Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

BotDet: A System for Real Time Botnet Command and Control Traffic Detection

IBRAHIM GHAFIR[1,2], VACLAV PRENOSIL[1], MOHAMMAD HAMMOUDEH[3], THAR BAKER[4], SOHAIL JABBAR[5], SHEHZAD KHALID[6], SARDAR JAF[2]

[1]Faculty of Informatics, Masaryk University, Brno, Czech Republic
[2]Department of Computer Science, Durham University, Durham, UK
[3]Faculty of Science and Engineering, Manchester Metropolitan University, Manchester, UK
[4]Department of Computer Science, Liverpool John Moores University, Liverpool, UK
[5]Department of Computer Science, National Textile University, Faisalabad, Pakistan
[6]Department of Computer Engineering, Bahria University, Islamabad, Pakistan

Corresponding author: Ibrahim Ghafir (e-mail: ibrahim.ghafir@durham.ac.uk).

ABSTRACT Over the past decade, the digitization of services transformed the healthcare sector leading to a sharp rise in cybersecurity threats. Poor cybersecurity in the healthcare sector, coupled with high value of patient records attracted the attention of hackers. Sophisticated advanced persistent threats and malware have significantly contributed to increasing risks to the health sector. Many recent attacks are attributed to the spread of malicious software, e.g., ransomware or bot malware. Machines infected with bot malware can be used as tools for remote attack or even cryptomining. This paper presents a novel approach, called BotDet, for botnet Command and Control (C&C) traffic detection to defend against malware attacks in critical ultrastructure systems. There are two stages in the development of the proposed sytsem: (i) we have developed four detection modules to detect different possible techniques used in botnet C&C communications; (ii) we have designed a correlation framework to reduce the rate of false alarms raised by individual detection modules. Evaluation results show that BotDet balances the true positive rate and the false positive rate with 82.3% and 13.6% respectively. Furthermore, it proves BotDet capability of real time detection.

INDEX TERMS Critical infrastructure security, healthcare cyber attacks, malware, botnet, command and control server, intrusion detection system, alert correlation.

I. INTRODUCTION

COUNTRY'S national security, economic vitality and daily life rely on a safe, stable, and resilient cyberspace. We depend on this vast array of networks to provide healthcare services, transport and communication, power our homes and run our economy [1]. Over the last decade, cyber attacks and intrusions have increased substantially, disrupting critical operations, resulting in business downtime and exposing sensitive personal and business information.

Statistics draw a grim picture about the cybersecurity challenges and digital risks in the healthcare industry. A report by the US Department of Health and Human Services [2] reveals that the healthcare sector has suffered from approximately four data breaches a week in 2016. To put this into perspective, one in every three American citizens was a victim of a breach in the healthcare sector. One of the primary reasons behind targeting healthcare organizations is that these organizations do not set protecting patient data as a priority, hence they under invest in qualified IT security personnel. The lack of solid information security infrastructure makes healthcare organizations an easy target. For instance, the recent attack on the National Health Service (NHS) in the UK showed that some hospitals and care providers systems were obsolete or have not been patched against well-known vulnerabilities. Additionally, patient records contain a wealth of information that can be used for identifying theft, financial/insurance fraud and even blackmailing. In 2017, 15,000 medical records have been stolen from Beverly Hills plastic surgery clinic to bully several high-profile celebrities.

Today, intelligence agencies and governments military are actively preparing for cyber warfare. Global activities against software, hardware, or data are referred to as cyber attack in the field of computer networks or systems. These activities lead to degrading, disrupting, destroying or denying access to network/system services or resources. Activities that target gathering intelligent are referred to as cyber exploitation [3].

The main objective of these activities is to gain unauthorised access to information and data.

Over the last decade, malicious software or malware has increased, particularly in the healthcare industry. They have become one of the main reasons for the majority of the (distributed) denial-of-service (Dos) activities [4], direct and scanning attacks [1], [5], [6]. Noticeably, the motivation from fame seeking and curiosity has been shifted to unlawful financial attainment, which resulted in the sophistication of malicious software [7]. Moreover, the availability of easy-to-use toolkits to build malware will probably keep these malwares a threat to individuals, business and governments in the foreseeable future.

Generally, there are two classes of malware: (a) malware that targets the general population and (b) customised information-stealing malware that targets particular organizations [7] such as healthcare providers. Zombies, which refers to those machines infected with bot malware, can be used as tools for remote attack or can be part of a botnet, which is completely controlled by the botnetmaster [7]. Bots are "enslaved" host computers in botnets (networks formed by bots). One or more botmasters control bots in botnets and the intention is to perform malicious activities [8], [9]. The essential goal of botnets is to control organized crime syndicate, criminal, or group of criminals to use compromised machines for performing illegal activities. Experts mention that about $16-25\%$ of the machines connected to the Internet are parts of botnets [10], [11]. Bots are different from the other malware. They are capable to create Command and Control (C&C) channels. Bots recognize themselves by their C&C channels through which they can be controlled, updated and instructed. The C&C servers are usually machines that have been exploited and sorted in a distributed form to limit traceability.

The detection of botnet C&C traffic is challenging for current Intrusion Detection Systems (IDS) for several reasons: (1) it is a benign traffic and follows normal protocol usage; (2) their volume of traffic is small; (3) the number of bots may be very small in the monitored network; and (4) Bots' communications may be encrypted [12].

This paper aims to contribute to IDS research, particularly to botnet C&C traffic detection. The proposed approach, called BotDet, undergoes two main phases. The first phase runs various modules to detect different possible techniques used in botnet C&C communications. The second phase uses a framework for alert correlation to reduce the number of false positives. The main contributions of this work are:

1) The development of four methods for the detection of various attack techniques used in botnet C&C communications. Although methodologies exist in the literature for blacklist-based detection modules, their implementation and validation in real traffic are significant contributions to the field.
2) The development of a framework to correlate results from individual detection methods to reduce the false alarms.
3) The automation of blacklists, used in some of the detection modules, based on different intelligent feeds. This allows BotDet to offer real time attack detection.
4) The evaluation results show that BotDet balances the true positive rate and the false positive rate with 82.3% and 13.6% respectively.

The remainder of this paper is structured as follows. Section II presents the related work to botnet C&C traffic detection. The proposed approach, including the detection modules and correlation framework, is presented in Section III. Section IV shows the evaluation results and Section V concludes the paper.

II. RELATED WORK

There are two main approaches for botnet C&C traffic detection in the literature. The first one is based on setting up honeynets in the network [13]. This approach is often used to understand and analyse a botnet technology and characteristics. However, honeynets are not always capable of detecting bot infection. The second approach is based on passive traffic monitoring [14]. These approaches can be classified into signature-based and anomaly-based methods, respectively. Signature-based detection methods make use of known signatures and behaviour of existing botnets, therefore it can be used for detecting only known botnets [15]. Anomaly-based detection methods are able to detect unknown botnets as they try to detect botnets based on network traffic anomalies like traffic on unusual ports, high volumes of traffic, unusual system behaviour and high network latency [16]. Detection methods can be further classified into host-based and network-based methods. Host-based method detects botnets by monitoring and analysing the internals of a computer system [17]. Whereas the network-based method monitors the network traffic to detect botnets [18].

Snort is a signature-based IDS [19] capable of monitoring and analysing network traffic to match signatures of known botnets. Snort consists of many components working together in order to detect malicious patterns in the traffic. Packets from network interfaces are captured by the packet decoder and they are prepared to be preprocessed or sent to the detection engine. Then, packets are checked against specific plugins by a processor, and if anomalies are found, the processor raises an alert.

In [20], Balram and Wilscy propose a host-based approach for botnet C&C communication detection. This approach analyses suspicious flows produced by filtering out benign traffic from the traffic created by a host. A normal profile of the host traffic is used for the filtering. The behavioural pattern of flows to all destinations is examined in a bid to generate the host profile. This approach achieved a detection rate of 100% and false positives of 8%.

In [21], Fedynyshyn et al. present a host-based detection method able to detect the existence of botnet C&C traffic on the observed machine, and also categorize the type of C&C communication used by the bot, e.g., peer-to-peer (P2P) based, HTTP-based or IRC-based. As it does not examine the

packets payloads, their detection method is independent of the content of the C&C messages. Their method for detecting and categorising botnet C&C connections is based on three hypotheses: (1) it is possible to distinguish between botnet C&C communication and botnet non-C&C communication, (2) it is possible to distinguish between botnet C&C communication and valid communication and (3) there are shared characteristics between different styles of C&C and different botnet families.

An approach for bot-infected machines detection was presented by Wurzinger et al. [22], which requires no previous knowledge of the way a bot spreads. It depends on the characteristic behaviour of a bot, particularly: (a) receiving commands from the botmaster, and (b) responding to these commands by carrying out some activities. Both commands and responses can be monitored in the network traffic and detection models can be built. The authors ran a bot in a controlled network to record its traffic and then they examine the received commands and responses activities. For this purpose, they proposed techniques to determine points in the network that were involved in the response activity. Afterwards, the traffic had been observed before this response is analysed to find the corresponding command. By these detection models the network traffic is scanned for similar actions aiming to detect bot-infected machines.

Giroire et al. [23] presented another host-based detection method for botnet C&C traffic detection. This method is based on the fact that the infected machines should stay in contact with C&C severs to be instructed and controlled by the botmaster. It is assumed that those connections are persistent and established regularly. A white-list of benign destinations that the user regularly contacts is built and all the user outbound traffic is monitored. When a connection is persistent enough and the destination is not white-listed, an alert is generated and the user is informed and asked to decide. If the destination is legitimate, the user can easily add it to the white-list, otherwise the connection is deemed as C&C communication and blocked.

A network-based botnet detection system, BotSniffer, was proposed in [12]. This system is based on anomaly-based detection algorithms to detect both HTTP and IRC based C&Cs with no previous knowledge of C&C server addresses or signatures. The main goal in BotSniffer is to identify spatial-temporal similarity patterns and correlation in network traffic that are generated between the infected hosts and botnet C&C servers. They study two common styles usually used for botnet control, “push” and “pull”. An example for the push style is IRC-based C&C is where the commands are sent or pushed to the infected hosts. In the pull style, the commands are downloaded (or pulled) by the infected hosts, as in HTTP-based C&C. When a set of hosts is found to carry out the same actions in response to similar messages from the same server, it is considered to be part of a botnet.

Several works have attempted to detect Botnet C&C traffic. However, they have limitations in achieving real time detection, and they cannot balance between false positive and false negative rates. Therefore, the significance of this work is we propose methods that achieve real time detection and produce a balance between false positive and false negative rates.

III. BOTDET DESIGN AND SPECIFICATIONS

BotDet runs through two main phases. The first phase hosts four modules to detect different techniques used in botnet C&C communications. The second phase requires using a framework for alert correlation, based on voting among individual detection modules. Figure 1 shows the architecture of the proposed BotDet.

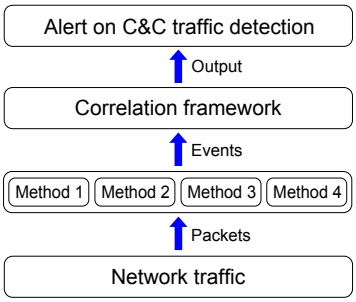

FIGURE 1. Architecture of the proposed approach for C&C traffic detection.

Initially, sniffed data traffic is scanned to detect techniques used in botnet C&C communications. To this end, four detection modules have been developed which are: malicious IP address detection module (MIPD), malicious SSL certificate detection module (MSSLD), domain-flux detection module (DFD) and Tor connection detection module (TorD). The output of this phase is alerts, also known as events, triggered by individual modules. Alerts raised by individual detection modules are then fed into the correlation framework (CF), which aims to find links between alerts to increase the confidence of botnet traffic detection and decrease the rate of false alarms.

The four detection modules operate in real time, as BotDet can process the sniffed network traffic live and does not have to store it. Some of the detection modules are blacklist-based, where some of these blacklists are publicly published or privately maintained. Information on different intelligence feeds at once is used to automatically update all used blacklists within BotDet. All detection modules are implemented on top of *Bro* [24], [25], which is a passive and open-source network traffic analyser.

A corresponding event is generated as an output for each detection module. This event is to be used in the correlation framework as explained later in Section III-F. In addition to this, an alert email is sent to the Request Tracker (RT) [26]. The network security team then conduct further forensics and reply to the alert–a reply from the team is assumed to be within 24 hours from the alert generation. This method allows the detection module to suppress all the alerts with the same infected host and the same malicious item into one

alert per day. Also, this method reduces the number of email alerts sent to the network security team and computational cost on the correlation framework. After generating an alert, the triggered alert is added to a specific corresponding table by the module and stored for 24 hours. This way, generating the same alert within the same 24 hours by the module is avoided. During the detection of an APT technique but prior to generating an alert, the module queries the corresponding table in order to check if the same alert exists in the table. If the same alert is found then it is ignored. Otherwise, information about the new alert and the malicious connection (such as *alert_type, timestamp, src_ip, src_port, dest_ip, dest_port, infected_host, malicious_item*) is recorded in a specific log file (individual log for each technique detection), which is used for keeping historical record of the monitored network.

The following subsections present the four detection modules MIPD, MSSLD, DFD and TorD. The automatic update details of the utilised blacklists are provided as well. Then, the correlation framework methodology is given.

A. MALICIOUS IP ADDRESS DETECTION (MIPD)

The MIPD module detects any connection between the infected host and a C&C server. The detection is based on a blacklist of malicious IPs of C&C servers [27]–[30]. As depicted in Figure 2, MIPD processes the network traffic to search for a match in the source and destination IP addresses for each connection with the IP blacklist [31].

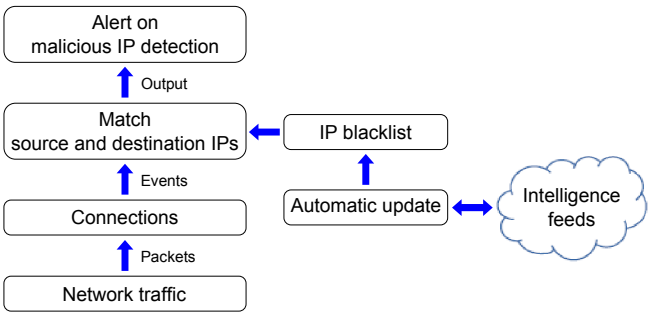

FIGURE 2. Methodology of the malicious IP detection.

Algorithm 1 shows the implementation pseudo-code of the MIPD module.

Algorithm 1 Implementation pseudo-code of MIPD

Input: blacklist of malicious IP addresses (*t_ip_blacklist* table)
Input: *new_connection* event
Check if the connection is to a malicious IP:
if the connection *destination IP* is in *t_ip_blacklist* **then**
  if the connection is established by one of the network's hosts **then**
    if MIPD didn't raise the same alert during the past 24 hours **then**
      goto Check if the connection is from a malicious IP:
    else
      Raise an alert *(ip_alert)*
      Record the generated alert
      Notify the network security team via email
      Deny repeating the same alert during the next 24 hours
    end if
  else
    goto Check if the connection is from a malicious IP:
  end if
else
  goto Check if the connection is from a malicious IP:
end if
Check if the connection is from a malicious IP:
if the connection *source IP* is in *t_ip_blacklist* **then**
  if the connection is oriented to one of the network's hosts **then**
    if MIPD didn't raise the same alert during the past 24 hours **then**
      goto End
    else
      Raise an alert *(ssl_alert)*
      Record the generated alert
      Notify the network security team via email
      Deny repeating the same alert during the next 24 hours
    end if
  else
    goto End
  end if
else
  goto End
end if
End

The network traffic is monitored to identify any *new_connection* type event generated by Bro. This event is generated for every new connection and raised with the first packet of a previously unknown connection [32]. Through the *new_connection* event, MIPD checks both connection sides IP addresses to detect if the connection is to or from a malicious IP. If the connection destination IP exists in the *t_ip_blacklist* table, this means, the connection is to a malicious IP. MIPD then checks the connection source IP through the *is_local_addr* function to determine if the connection is established by a host from the monitored network. On detecting a malicious connection, but prior to raising an alert, MIPD checks the *t_suppress_ip_alert* table to determine if the same *ip_alert* has been generated in the last 24 hours. If the same *ip_alert* has not been generated, MIPD does the following: (i) an *ip_alert* event is generated and information about the malicious connection is written to a *blacklist_detection_ip.log* file, (ii) an email alert about the malicious IP detection is sent to RT and, (iii) the current detected set (host, ip) is added to the *t_suppress_ip_alert*

table.

When the connection is from a malicious IP, the same procedure as in when the connection is to a malicious IP followed by paying attention to the source and destination IP addresses as shown in Algorithm 1.

B. MALICIOUS SSL CERTIFICATE DETECTION (MSSLD)

Secure Sockets Layer (SSL) encryption is used for protecting C&C communications because it makes it difficult to identify it as malicious traffic. A blacklist of malicious SSL certificates for detecting C&C communications is used by MSSLD [33], [34]. There are two forms of the certificate in the blacklist: (a) *SHAI fingerprints* and (b) *serial & subject*. Malware and malicious activities are associated with both. As depicted in Figure 3, the main processes are: (1) processing the network traffic, (2) filtering the secure connections and (3) of each secure connection the SSL certificate is matched with the SSL certificate blacklist [35].

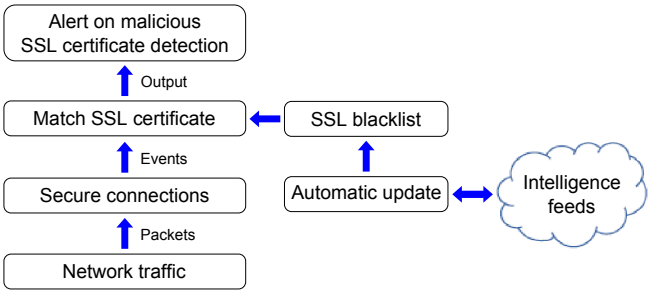

FIGURE 3. Methodology of the malicious SSL certificate detection.

Two methods are implemented for malicious SSL certificate detection because the blacklist comprises two types of malicious SSL certificates (which are *SHA1 fingerprints* and *serial & subject*): (1) intelligence-based MSSLD which is shown in Algorithm 2 and, (2) event-based MSSLD, as in Algorithm 3.

The Intelligence-based MSSLD uses the *Bro Intelligence Framework* [36] and it is configured to monitor the hashes of all secure connections SSL certificates, where data from different data sources can be consumed by MSSLD for hash matching. The framework is connected to *blacklist.intel* file, which contains the SSL certificate blacklist. The intelligence framework receives the SSL certificates after extracting the secure connections traffic. The framework checks the SSL certificates against the intelligence data set *blacklist.intel*. If a match is found with any of the *indicator_type* of the intelligence data, an *Intel::match* event is generated. The *Intel::match* event is examined to check if the *indicator_type* is *CERT_HASH*. If that is the case then this indicates the connection has a malicious SSL certificate. Next, *is_local_addr* function is used for checking the source and IP addresses of the connection sides to determine whether the connection to or from the monitored network is established. In order the avoid generating identical alerts within any 24-hour period,

Algorithm 2 Implementation pseudo-code of intelligence-based MSSLD

```
Input: blacklist of certificates' hashes
for each SSL encrypted connection do
    Compute the certificate hash
    if the certificate hash matches with the blacklist then
        if the connection is established by one of the network's hosts then
            if MSSLD didn't raise the same alert during the past 24 hours then
                Raise an alert (ssl_alert)
                Record the generated alert
                Notify the network security team via email
                Deny repeating the same alert during the next 24 hours
            end if
        else if the connection is oriented to one of the network's hosts then
            if MSSLD didn't raise the same alert during the past 24 hours then
                Raise an alert (ssl_alert)
                Record the generated alert
                Notify the network security team via email
                Deny repeating the same alert during the next 24 hours
            end if
        else
            goto End
        end if
    else
        goto End
    end if
end for
End
```

t1_suppress_ssl_alert table is checked. No alert will be generated if the table contains the same detected [host IP address, SSL certificate hash] set.

The processes of the MSSLD module are: (1) generating an *ssl_alert* event, (2) logging the malicious connection information to the *blacklist_detection_ssl.log* log file, (3) composing and sending an email alert about the malicious SSL certificate detection to the RT, and (4) updating the table *t1_suppress_ssl_alert* with the current detected set [host IP address, SSL certificate hash].

The process in the event-based MSSLD involves processing and filtering the network traffic into secure connections traffic. Then, for the encountered X509 certificates [37], the *x509_certificate* event is generated, where the serial and subject of the X509 certificate are checked for the existence of the certificate in the *bad_ssl* group. This group contains serials and subjects of malicious X509 certificates. MSSLD determines if there is a connection to or from one

Algorithm 3 Implementation pseudo-code of event-based MSSLD

```
Input: blacklist of certificates' hashes
Input x509_certificate event
Obtain the serial and subject of the certificate
if serial and subject match with the blacklist then
    if the connection is established by one of the network's hosts then
        if MSSLD didn't raise the same alert during the past 24 hours then
            Raise an alert (ssl_alert)
            Record the generated alert
            Notify the network security team via email
            Deny repeating the same alert during the next 24 hours
        end if
    else if the connection is oriented to one of the network's hosts then
        if MSSLD didn't raise the same alert during the past 24 hours then
            Raise an alert (ssl_alert)
            Record the generated alert
            Notify the network security team via email
            Deny repeating the same alert during the next 24 hours
        end if
    else
        goto End
    end if
else
    goto End
end if
End
```

of the monitored network hosts if a match is found. The source and destination IP addresses are checked through the *is_local_addr* function. The *t2_suppress_ssl_alert* table is checked before generating the *ssl_alert*. This is to avoid generating an alert that may have been generated in the past 24-hour period. As in the previous intelligence-based method, the process involves: (1) MSSLD generating an *ssl_alert* event; (2) recording the malicious connection information in the *blacklist_detection_ssl.log* file; (3) sending an alert to the RT; and (4) updating the *t2_suppress_ssl_alert* table with the current detected set [host IP address, SSL certificate hash].

C. DOMAIN FLUX DETECTION (DFD)

One common technique used for C&C communications is the domain flux technique, where each infected machine separately uses a Domain Generation Algorithm (DGA) to generate a list of domain names [38]. By using the domain flux technique, the infected host attempts to query and connect to a large number of generated domain names, which are expected to link the host to the C&C servers. This technique makes it difficult for law enforcement to successfully shut down a large number of domains. To prevent infected hosts from connecting to the C&C servers, law enforcement needs to pre-register all the domains that an infected host queries every day before the attacker registers them [39].

The DFD module detects algorithmically generated domain flux, where the infected host queries for the existence of a large number of domains, whilst the owner has to register only one. This leads to the failure of many DNS queries. DFD utilizes DNS query failure to detect domain flux attacks. Figure 4 depicts the way the network traffic is processed, particularly DNS traffic. All DNS query failures are analysed and, for detecting domain flux attacks and identifying infected hosts, the same IP address is constrained by a threshold for DNS query failures [40].

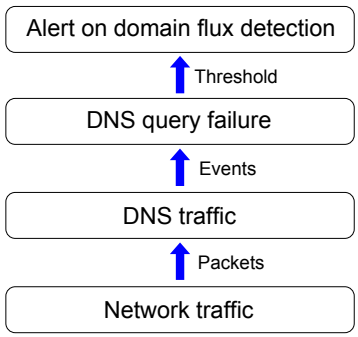

FIGURE 4. Methodology of the domain flux detection.

Algorithm 4 shows the implementation pseudo-code of the DFD module. DNS traffic is extracted and processed; DFD waits for the *dns_message* event to be generated by Bro. This event is generated for any DNS message and provides information regarding the connection to the DNS server [41].

Through the *dns_massage* event, DFD checks for two conditions: (1) if this connection is established by a host from the monitored network using the *is_local_addr* function; (2) if the *dns_message* is due to DNS error of NXDOMAIN, which indicates the non-existence of the domain name where it is either invalid or not registered. This information can be extracted from the *dns_message* event (*cdnsrcode_name=="NXDOMAIN"*). If these two conditions are met, the source IP address that searches for unregistered domain names is saved in the *t_dns_failure* table. This table counts the number of DNS query failures of the same IP address. The counter is increased by one (++ *t_dns_failure[cidorig_h]*) if the *t_dns_failure* table contains the current IP address. When the number of DNS query failures exceeds a specified threshold, *dns_failure_threshold*, the current IP address is deleted from the *t_dns_failure* table to reset the counter of this IP address to zero. Because recent malware can generate $50,000$ domain names every day [42], the threshold is set to 50 DNS query failures per 5 minutes. Then, if the IP address of the potentially infected host does not exist in the *t_suppress_domain_flux_alert* table, DFD generates a *domain_flux_alert* event.

Algorithm 4 Implementation pseudo-code of DFD

```
Input: dns_failure_threshold
Extract DNS traffic
Input: dns_message event
if the connection is established by one of the network's hosts then
    if dns_message event is due to DNS error of NXDOMAIN then
        if the host IP is not in t_dns_failure table then
            write host IP into t_dns_failure
            host IP counter ← 1
        else
            Increase host IP counter by 1
            if host IP counter > dns_failure_threshold then
                Delete host IP from t_dns_failure
                Reset host IP counter to zero
                if DFD raised the same alert during the past 24 hours then
                    goto End
                else
                    Raise an alert (domain_flux_alert)
                    Record the generated alert
                    Notify the network security team via email
                    Deny repeating the same alert during the next 24 hours
                end if
            else
                goto End
            end if
        end if
    else
        goto End
    end if
else
    goto End
end if
End
```

DFD also sends a domain flux detection email alert to the RT, and updates the *t_suppress_domain_flux_alert* table by adding the current detected host IP address.

D. TOR CONNECTION DETECTION (TORCD)

Tor [43] [44] is an anonymous communication network that provides user privacy by encrypting the connection through an overlay network. Tor uses onion routing to direct client's traffic over a circuit of different relays, denying any single relay to know the complete path of the traffic [45]. Tor is often misused by criminals and hackers to remotely direct and instruct infected machines [46].

The TorCD module detects any connection to a Tor network. It is based on a list of Tor servers which is publicly published [47]. As shown in Figure 5, the network traffic is processed and the source and destination IP addresses for each connection are matched with Tor servers list [48].

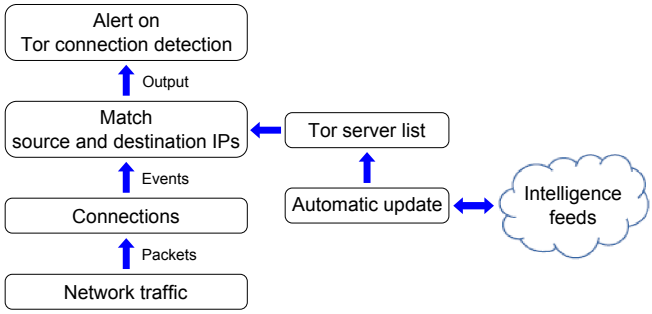

FIGURE 5. Methodology of the Tor connection detection.

Algorithm 5 shows the implementation pseudo-code of the TorCD module. The network traffic is processed, when a SYN-ACK packet is seen in response to a SYN packet during a TCP handshake [49] a *connection_established* event is generated by Bro.

Algorithm 5 Implementation pseudo-code of TorCD

```
Input: Tor servers list (t_tor_server table)
Input: connection_established event
Check if the connection is to a Tor network:
if the connection destination IP is in t_tor_server then
    if the connection is established by one of the network's hosts then
        if TorCD raised the same alert during the past 24 hours then
            goto Check if the connection is from a Tor network:
        else
            Raise an alert (tor_alert)
            Record the generated alert
            Notify the network security team via email
            Deny repeating the same alert during the next 24 hours
        end if
    else
        goto Check if the connection is from a Tor network:
    end if
else
    goto Check if the connection is from a Tor network:
end if
Check if the connection is from a Tor network:
if the connection source IP is in t_tor_server then
    if the connection is oriented to one of the network's hosts then
        if TorCD raised the same alert during the past 24 hours then
            goto End
        else
            Raise an alert (tor_alert)
```

```
27:                Record the generated alert
28:                Notify the network security team via email
29:                Deny repeating the same alert during the next
    24 hours
30:            end if
31:        else
32:            goto End
33:        end if
34:    else
35:        goto End
36:    end if
37:    End
```

Through the *connection_established* event, TorCD checks both sides of the connection to detect if the connection is to or from a Tor network. The process proceeds in the same way as in the MIPD module (see Section III-A), i.e., a *tor_alert* event is generated, and an alert email is sent to the RT.

E. AUTOMATIC UPDATES

Based on different intelligence feeds, the blacklists of blacklist-based detection modules are automatically updated at once. The automatic update and detection processing run in parallel and it is not required to halt or restart BotDet. This way, it is possible to monitor the live network and support real-time detection.

There are two automatic update mechanisms. Figure 6 shows the automatic update of the blacklists used by the MIPD, MSSLD and TorCD modules. The user *crontab file* is configured to run the *blacklist_update.sh* each day at 3:00am, where the shell script connects to the intelligence feeds via the Internet and downloads updated blacklist of malicious IPs, malicious SSL certificates and Tor servers into the *ip_blacklist.txt*, *ssl_blacklist.txt* and *Tor_servers_list.txt* files respectively. The *Input Framework* [50], built in Bro, enables the four modules to use those text files as an input to BotDet. The Input Framework reads the *ip_blacklist.txt*, *ssl_blacklist.txt* and *Tor_servers_list.txt* files into the *t_ip_blacklist* table, the *bad_ssl* group and the *t_tor_server* table, respectively.

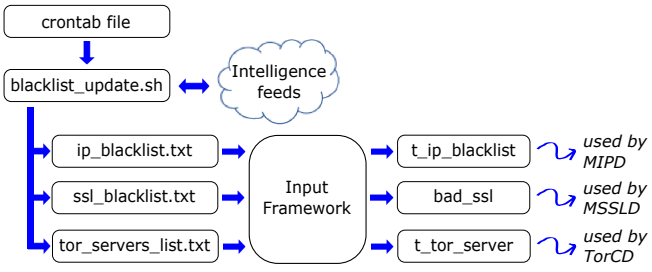

FIGURE 6. Automatic update of the blacklists used by the MIPD, MSSLD and TorCD modules.

Figure 7 shows the automatic update of the blacklist used by the MSSLD module. The *crontab file* user is configured to run *blacklist_update.sh* daily at 3:00am. This shell script, which utilises the Internet to connect to the data source servers, downloads updated blacklist of malicious SSL certificate hashes into a new *blacklist.intel* file. The *Intelligence Framework* consumes the text file, described in Section III-B.

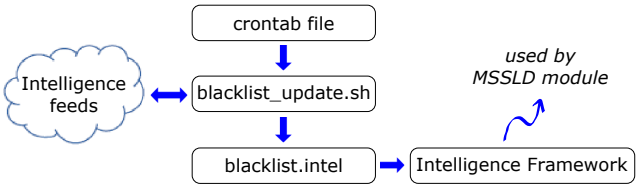

FIGURE 7. Automatic update of the blacklist used by the MSSLD module.

F. BOTDET CORRELATION FRAMEWORK (CF)

CF alert correlation time is configured to one day. Based on the number of the linked alerts, CF can generate four types of alerts, which are *C&C-1 alert*, *C&C-2 alert*, *C&C-3 alert* and *C&C-4 alert*. For example, the *C&C-3 alert* is raised when CF finds three different alerts about the same infected host during the previous day. For the *C&C-1 alert*, CF does not raise a *tor_alert* because Tor is a legitimate service that can be used legally by some users on the network.

IV. EVALUATION RESULTS

Three scenarios were implemented to evaluate BotDet. In the first one, third-party pcap files were analysed. In the second scenario, a virtual network was used. In the third scenario, the university campus live traffic was monitored.

In the first scenario, BotDet was run on pcap files that contained captured malware traffic. Four groups of pcap files were used, PCAP1, PCAP2, PCAP3 and PCAP4. PCAP1 files contain *Trojan-Spy.Win32.Zbot.oowo* traffic that spans 27 days in two files with total size of 11.7 GB and connections to 6339 IP addresses. The studied malware uses domain flux for C&C communication [28]. PCAP2 files contain *Trojan-Spy.Win32.Zbot.rfnx/.sbfp/.sbcq* traffic in 3 files, each spanning 10 hours and having total size of 28 MB. This malware connects to a blacklisted IP address and uses domain flux [29]. PCAP3 files contain recorded traffic, in 11 files, of multiple types of malware that all use the domain flux technique [30]. PCAP4 files contain recorded traffic of the *Trojan.Tbot* (Skynet Tor Botnet) malware in 6 files with a total size of 31.6 MB. *Trojan.Tbot* uses Tor network to communicate with its C&C centre [51]. None of the analysed pcap files contained known bad SSL certificates, this module was tested in the second scenario. All pcap files had been analysed by the providers, so the ground truth was known.

The detection modules were configured to consume the pcap files and produce log files. Then CF was used to correlate the individual modules' alerts to detect C&C communications. Results from individual modules and CF were comparable to the ground truth, and the values of True Positive Rate (TPR) and False Positive Rate (FPR) were calculated. Table 1 shows the results.

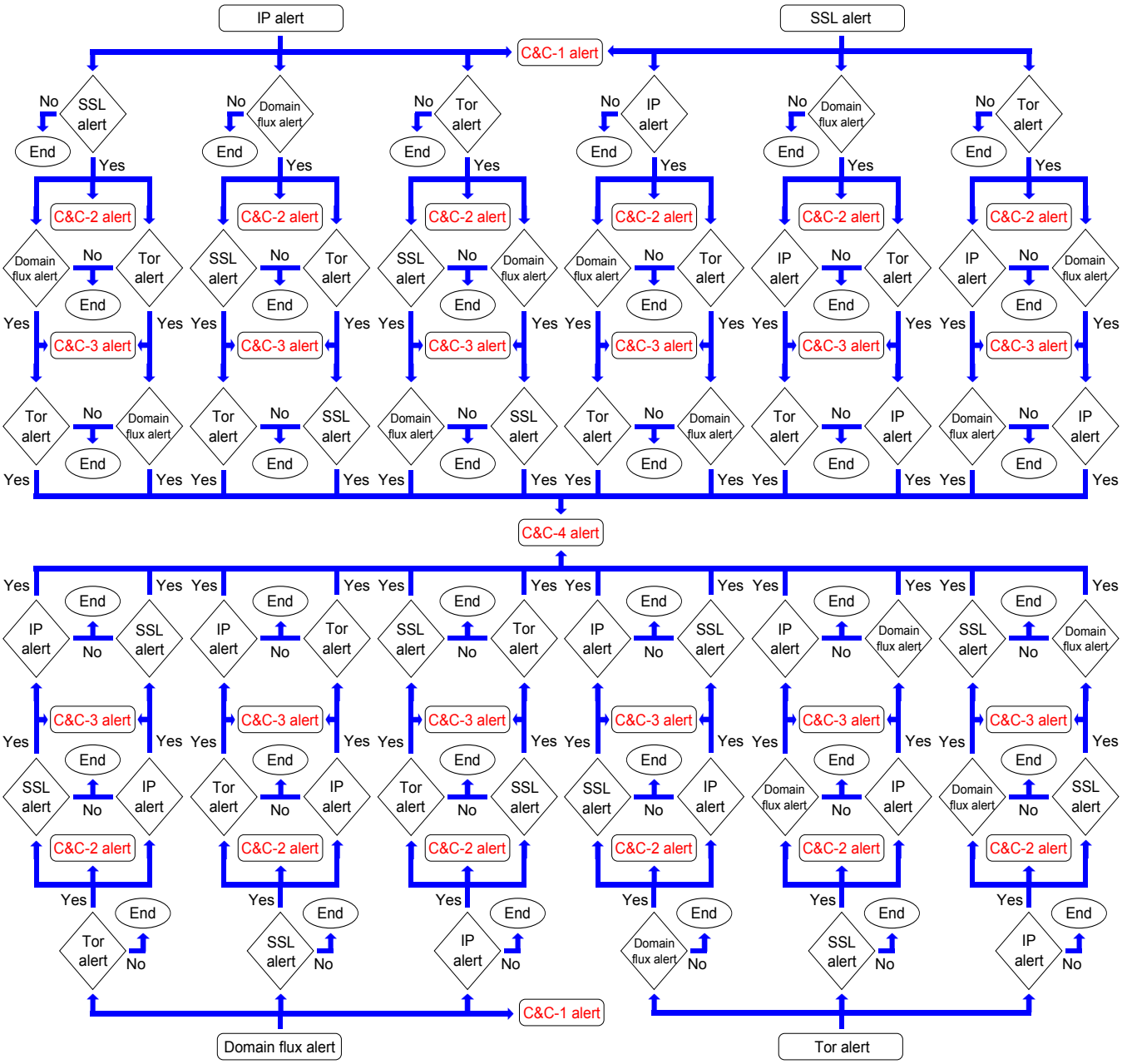

FIGURE 8. The architecture of the correlation framework.

TABLE 1. The TPR and FPR of the individual detection modules.

Detection module	TPR	FPR
Malicious IP	63%	39.2%
Malicious SSL	–	–
Domain flux	86.4%	24.9%
Tor connection	65%	33.8%

Among the individual modules, the domain flux detection module has the best results, with TPR of 86.4% and FPR of 24.9%. The results of IP and Tor detection modules are affected by the quality of IP blacklist and Tor server list respectively. We argue that IP and Tor detection modules can only work hand in hand with infrastructure that intelligently updates the blacklist and Tor server list in real time, which is not the case in this scenario. Figure 9 shows part of a log produced by Tor connection detection module.By correlating the detection modules alerts, CF increases the TPR and decreases the FPR. Table 2 shows the TPR and FPR of CF based on one and multiple modules.

Although the correlation based on one detection module

```
#fields    timestamp      alert_type  infected_host    tor_server
#types     time           string      addr             addr
1349621090.423721  tor_alert  172.16.253.130  208.83.223.34
1349621090.945925  tor_alert  172.16.253.130  86.59.21.38
1349621102.634850  tor_alert  172.16.253.130  74.120.13.132
1349621102.628802  tor_alert  172.16.253.130  96.47.226.20
1349621102.670488  tor_alert  172.16.253.130  96.44.189.102
#close  2015-03-24-18-06-02
```

FIGURE 9. Part of a log produced by Tor connection detection module.

TABLE 2. The TPR and FPR of the correlation framework

Correlation framework	TPR	FPR
Based on one detection module	93%	47.9%
Based on two detection modules	82.3%	13.6
Based on three detection modules	41.4%	9%
Based on four detection modules	–	–

has the highest TPR, it also has the highest FPR. The best results are for the correlation based on two detection modules, with TPR of 82.3% and FPR of 13.6%. While the correlation based on three detection modules has the best FPR value, TPR is the lowest. We argue that the acceptable TPR and FPR values are specific to the environment and purpose in which the modules are used. The correlation framework allows for a selection between different TPR/FPR trade-offs. The best selection is the one that yields the best balance between TPR and FPR among the available choices. In comparison to the best results of individual modules (86.4% for TPR and 24.9% for FPR), the correlation based on two detection modules has increased the detection rate and decreased the false alarms. Despite the fact that TPR is slightly less, FPR has been considerably reduced. Figure 10 shows part of a log produced by the correlation framework.

```
#fields timestamp      CandC_detection     infected_host
alert_1     malicious_addr_1  time_1     orig_h_1  orig_p_1  resp_h_1  resp_p_1
alert_2                       time_2     orig_h_2  orig_p_2  resp_h_2  resp_p_2
#types time            string              addr
string    addr         time                addr      port      addr      port
string                 time                addr      port      addr      port
1425360376.712895  CandC_two_alerts   10.0.2.15
ip_alert  54.83.43.69  1425325993.160545  10.0.2.15  49171  54.83.43.69  80
domain_flux_alert     1425360376.712895  10.0.2.15  60544  8.8.8.8      53
#close 2015-03-21-20-55-08
```

FIGURE 10. Part of a log produced by the correlation framework.

In the second scenario, a virtual network connected to the Internet was built, the network was injected with malware samples, and the network traffic was recorded into pcap files. As in the first scenario, those pcap files were used to evaluate the detection modules and CF performance. The analysed malware samples were *Trojan.Win32.Inject.sbqz*, *Trojan.Win32.Staser.bazr*, *HEUR:Trojan.Win32.Generic* and *Trojan-Spy.Win32.Zbot.qvcn*.

As shown in Figure 11, two Windows XP SP3 virtual machines were connected to a router. The virtual machines, which provided internet connection, mimicked physical computers in a home network where they can communicate with each other.

The nictrace VirtualBox functionality [52] was used for recording the virtual machines traffic to two pcap files, one pcap file per virtual machine. Because the virtual machines has no applications installed and the operating system updates was disabled, the majority of the virtual machine's traffic was initiated by the installed malware to easily establish the ground truth.

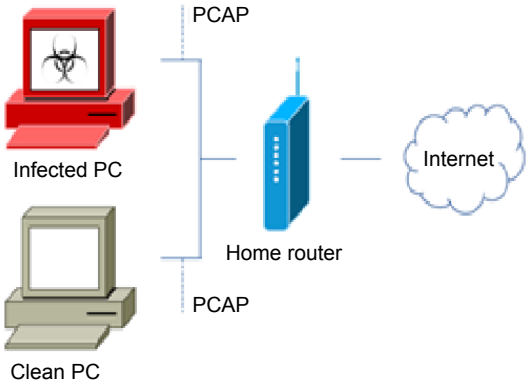

FIGURE 11. Topology of the implemented virtual network.

The *HEUR:Trojan.Win32.Generic* malware (MD5, *fbb354f6773fb81927a59008cd9fd3a6*) was run on the virtual machine A for 48 hours. Through manual traffic analysis, we found that the virtual machine A connected to the C&C centre and the virtual machine B did not become infected. The malware downloaded its payload from *dstkom.com/mandoc/lit23.pdf*. The infected computer then proceeded to use the domain flux technique to connect to the C&C servers over ports 1778, 3363, 3478 and 3479.

The *Trojan-Spy.Win32.Zbot.qvcn* malware (MD5 hash, *52d3b26a03495d02414e621ee4d0c04e*) was run on the same virtual network for 10 hours. The malware communicated solely through the Tor network and did not exhibit other activities. In the first two minutes, the malware initiated 67 connections to 67 addresses belonging to the Tor network and transferred 3698 kB of data. The flow of data dropped for the remaining time but new connections were still made. These findings were used to establish the ground truth and test the Tor detection module.

The *Trojan.Win32.Inject.sbqz*, also known as TorrentLocker, (with MD5 hash *aabe2844ee61e1f2969d7a96e1355a99*) and *Trojan.Win32.Staser.bazr* malware (with MD5 hash *e161a4d2716eb83552d3bd22ce5d603c*) were run on the same virtual network independently for 5 minutes each. The C&C servers for these two malware use SSL certificates for communication over *https*.

Table 3 and Table 4 show the results of individual detection modules and the correlation framework respectively. Similar to the first scenario, the correlation based on two detection modules has the best results with TPR of 79% and FPR of 16.8%.

TABLE 3. The TPR and FPR of the individual detection modules.

Detection module	TPR	FPR
Malicious IP	61%	34.7%
Malicious SSL	42.6%	0%
Domain flux	83%	27.3%
Tor connection	63.3%	29%

TABLE 4. The TPR and FPR of the correlation framework

Correlation framework	TPR	FPR
Based on one detection module	91.8%	45%
Based on two detection modules	79%	16.8
Based on three detection modules	52.6%	7%
Based on four detection modules	31.1%	0%

In the third scenario, part of the campus live traffic (200 Mbps, 200 users, 550 nodes) was monitored for one month. The four detection modules and CF were hosted on a server (2x 4-core Intel Xeon CPU E5530 @ 2.40GHz, 12 GB RAM) with passive access to the campus live traffic via an optical TAP (Test Access Port). Figure 12 shows part of a log produced by the developed system BotDet, hosted on a server with passive access to the campus live traffic.

```
#fields   timestamp      CandC_detection   infected_host
alert_1   malicious_addr_1 time_1          orig_h_1      orig_p_1 resp_h_1 resp_p_1
alert_2   malicious_addr_2 time_2          orig_h_2      orig_p_2 resp_h_2 resp_p_2
#types    time           string            addr
string    addr           time              addr          port     addr     port
string    addr           time              addr          port     addr     port
1427273858.680063   CandC_two_alerts
tor_alert 5.39.80.135  1427191426.664526                 56946    5.39.80.135 9001
domain_flux_alert  -   1427273858.680063                 47565    8.8.8.8     53

1427273995.014264   CandC_two_alerts
domain_flux_alert  -   1427201796.739411                 59119    147.251.4.33 53
ip_alert 77.247.181.162 1427273995.014264 77.247.181.162 51753    [redacted]   80
#close 2015-03-25-10-00-00
```

FIGURE 12. Part of a log produced by BotDet for live traffic.

Figure 13 shows an example of the *CandC_Traffic_Two_Alerts* ticket sent by BotDet via email to RT, where the network security team can perform additional forensics and respond to it.

V. CONCLUSION AND FUTURE WORK

This paper presents a novel approach called BotDet for botnet C&C traffic detection. The developed system (BotDet) runs through two main phases, the first one includes developed modules to detect possible techniques used in botnet C&C communications. The second phase uses a framework for alert correlation, based on voting between the detection modules. BotDet achieves detection rate and false alarm of 82.3% and 13.6% respectively. Additionally, the blacklists used in some of the detection modules are automatically updated based on different intelligent feeds, which gives BotDet the capability of real time detection.

As future work, more detection modules will be added to detect other techniques used in botnet C&C communications. Besides, alerts from external IDSs deployed on the network can be received and fed into BotDet, which can ultimately reduce the false positive rate of the system.

```
Greetings,

the security team CSIRT-MU detected involvement of the IP address
[redacted] into the following incident:

Incident type: CandC_Traffic_Two_Alerts
Time of detection: 2015-03-12 15:27:31 +0100
IP address [redacted]
Domain name: ---

Details of this incident can be found at this address:
https://reports.csirt.muni.cz/A4FE72DC-65A8-1C74-8627-5664BE43D472

Best regards,
CSIRT-MU, the security team of Masaryk University
http://www.muni.cz/csirt
            Date: Thu, 12 Mar 2015 15:27:39 +0100
```

FIGURE 13. CandC_Traffic_Two_Alerts ticket.

REFERENCES

[1] S. Belguith, N. Kaaniche, A. Jemai, M. Laurent, and R. Attia, "Pabac: A privacy preserving attribute based framework for fine grained access control in clouds," in Proceedings of the 13th International Joint Conference on e-Business and Telecommunications, 2016, pp. 133–146.

[2] "US Department of Health and Human Services report," https://ocrportal.hhs.gov/ocr/breach/breach_report.jsf, accessed: 07-01-2018.

[3] P. J. Denning and D. E. Denning, "Discussing cyber attack," Communications of the ACM, vol. 53, no. 9, pp. 29–31, 2010.

[4] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage, "Inferring internet denial-of-service activity," ACM Transactions on Computer Systems (TOCS), vol. 24, no. 2, pp. 115–139, 2006.

[5] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. P. Markatos, and A. D. Keromytis, "Detecting targeted attacks using shadow honeypots." in Usenix Security, 2005.

[6] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," Journal of Computer Security, vol. 10, no. 1, pp. 105–136, 2002.

[7] K.-K. R. Choo et al., "Cyber threat landscape faced by financial and insurance industry," 2011.

[8] P. Bacher, T. Holz, M. Kotter, and G. Wicherski, "Know your enemy: Tracking botnets," 2005.

[9] S. Belguith, N. Kaaniche, M. Laurent, A. Jemai, and R. Attia, "Phoabe: Securely outsourcing multi-authority attribute based encryption with policy hidden for cloud assisted iot," Computer Networks, vol. 133, pp. 141 – 156, 2018.

[10] W. Sturgeon, "Net pioneer predicts overwhelming botnet surge," ZDNet News, January, vol. 29, 2007.

[11] B. AsSadhan, J. M. Moura, D. Lapsley, C. Jones, and W. T. Strayer, "Detecting botnets using command and control traffic," in Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on. IEEE, 2009, pp. 156–162.

[12] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," 2008.

[13] S. Kumar, R. Sehgal, P. Singh, and A. Chaudhary, "Nepenthes honeypots based botnet detection," arXiv preprint arXiv:1303.3071, 2013.

[14] S. García, A. Zunino, and M. Campo, "Survey on network-based botnet detection methods," Security and Communication Networks, vol. 7, no. 5, pp. 878–903, 2014.
[15] S. Behal, A. S. Brar, and K. Kumar, "Signature-based botnet detection and prevention," http://www.rimtengg.com/iscet/proceedings/pdfs/advcomp/148.pdf, 2010.
[16] S. Arshad, M. Abbaspour, M. Kharrazi, and H. Sanatkar, "An anomaly-based botnet detection approach for identifying stealthy botnets," in Computer Applications and Industrial Electronics (ICCAIE), 2011 IEEE International Conference on. IEEE, 2011, pp. 564–569.
[17] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," Journal of Network and Computer Applications, vol. 36, no. 1, pp. 42–57, 2013.
[18] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," Journal of Network and Computer Applications, vol. 36, no. 1, pp. 16–24, 2013.
[19] P. Agarwal and S. Satapathy, "Implementation of signature-based detection system using snort in windows," 2014.
[20] S. Balram and M. Wilscy, "User traffic profile for traffic reduction and effective bot c&c detection." IJ Network Security, vol. 16, no. 1, pp. 46–52, 2014.
[21] G. Fedynyshyn, M. C. Chuah, and G. Tan, "Detection and classification of different botnet c&c channels," in Autonomic and Trusted Computing. Springer, 2011, pp. 228–242.
[22] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda, "Automatically generating models for botnet detection," in Computer Security–ESORICS 2009. Springer, 2009, pp. 232–249.
[23] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and D. Papagiannaki, "Exploiting temporal persistence to detect covert botnet channels," in Recent Advances in Intrusion Detection. Springer, 2009, pp. 326–345.
[24] V. Paxson, "Bro: a system for detecting network intruders in real-time," Computer networks, vol. 31, no. 23, pp. 2435–2463, 1999.
[25] The-Bro-Project, "The bro network security monitor," https://www.bro.org/, accessed: 15-02-2015.
[26] Best-Practical-Solutions, "Rt: Request tracker," https://www.bestpractical.com/rt/, accessed: 15-01-2018.
[27] J. B. Kowalski, "Tor network status," http://torstatus.blutmagie.de/, accessed: 07-04-2015.
[28] Malware-Capture-Facility-Project, "Analysis of ctu-malware-capture-1 (zbot.oowo)," http://mcfp.weebly.com/analysis/analisis-ofctu-malware-capture-1-zbotoowo, accessed: 07-04-2015.
[29] "Botnet malware pcaps," http://radkodimitrov.free.bg/, accessed: 07-04-2015.
[30] Network-Security-Blog, "Dns fast flux - analysis and detection," http://newtorksecurityblog.blogspot.cz/2015/02/dns-fast-flux-analysis-and-detection.html, accessed: 07-04-2015.
[31] I. Ghafir and V. Prenosil, "Blacklist-based malicious ip traffic detection," in Global Conference on Communication Technologies (GCCT). IEEE Xplore Digital Library, 2015, pp. 229–233.
[32] Bro-Project, "new_connection event," https://www.bro.org/sphinx/scripts/base/bif/event.bif.bro.html\#id-new_connection, accessed: 01-06-2017.
[33] Kaspersky-Lab-ZAO, "The inevitable move - 64-bit zeus enhanced with tor," http://securelist.com/blog/events/58184/the-inevitable-move-64-bit-zeus-enhanced-with-tor/, accessed: 07-04-2015.
[34] NETRESEC, "Detecting tor communication in network traffic," http://www.netresec.com/?page=Blog\&month=2013-04\&post=Detecting-TOR-Communication-in-Network-Traffic, accessed: 07-04-2015.
[35] I. Ghafir, V. Prenosil, M. Hammoudeh, L. Han, and U. Raza, "Malicious ssl certificate detection: A step towards advanced persistent threat defence," in Proceedings of the International Conference on Future Networks and Distributed Systems. ACM, 2017, p. 27.
[36] Bro-Project, "Intelligence framework," https://www.bro.org/sphinx/frameworks/intel.html, accessed: 15-02-2017.
[37] The-Bro-Project, "x509_certificate event," https://www.bro.org/sphinx/scripts/base/bif/plugins/Bro_X509.events.bif.bro.html\#id-x509_certificate, accessed: 01-06-2017.
[38] B. Stone-Gross, M. Cova, B. Gilbert, R. Kemmerer, C. Kruegel, and G. Vigna, "Analysis of a botnet takeover," Security & Privacy, IEEE, vol. 9, no. 1, pp. 64–72, 2011.
[39] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your botnet is my botnet: analysis of a botnet takeover," in Proceedings of the 16th ACM conference on Computer and communications security. ACM, 2009, pp. 635–647.
[40] I. Ghafir and V. Prenosil, "Dns query failure and algorithmically generated domain-flux detection," in International Conference on Frontiers of Communications, Networks and Applications (ICFCNA). IEEE Xplore Digital Library, 2014, pp. 1–5.
[41] Bro-Project, "dns_message event," https://www.bro.org/sphinx/scripts/base/bif/plugins/Bro_DNS.events.bif.bro.html\#id-dns_message, accessed: 01-08-2017.
[42] M. Mowbray and J. Hagen, "Finding domain-generation algorithms by looking at length distribution," in 2014 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, 2014, pp. 395–400.
[43] S. Doswell, N. Aslam, D. Kendall, and G. Sexton, "Please slow down!: the impact on tor performance from mobility," in Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices. ACM, 2013, pp. 87–92.
[44] S. Chakravarty, G. Portokalidis, M. Polychronakis, and A. D. Keromytis, "Detection and analysis of eavesdropping in anonymous communication networks," International Journal of Information Security, pp. 1–16, 2014.
[45] A. Kapadia, "Analysis of the tor browser and its security vulnerabilities," 2014.
[46] R. Jagerman, W. Sabée, L. Versluis, M. de Vos, and J. Pouwelse, "The fifteen year struggle of decentralizing privacy-enhancing technology," arXiv preprint arXiv:1404.4818, 2014.
[47] J. B. Kowalski, "Tor network status," http://torstatus.blutmagie.de/, accessed: 07-09-2017.
[48] I. Ghafir, J. Svoboda, and V. Prenosil, "Tor-based malware and tor connection detection," in International Conference on Frontiers of Communications, Networks and Applications (ICFCNA). IEEE Xplore Digital Library, 2014, pp. 1–6.
[49] Bro-Project, "connection_established event," https://www.bro.org/sphinx/scripts/base/bif/plugins/Bro_TCP.events.bif.bro.html\#id-connection_established, accessed: 01-11-2017.
[50] The-Bro-Project, "Input framework," https://www.bro.org/sphinx/frameworks/input.html, accessed: 01-06-2016.
[51] NETRESEC, "Detecting tor communication in network traffic," http://www.netresec.com/?page=Blog\&month=2013-04\&post=Detecting-TOR-Communication-in-Network-Traffic, accessed: 07-04-2015.
[52] Oracle, "Network tracing," https://www.virtualbox.org/wiki/Network_tips, accessed: 07-04-2015.