
Downloaded from: http://sure.sunderland.ac.uk/id/eprint/10451/

Usage guidelines

Please refer to the usage guidelines at http://sure.sunderland.ac.uk/policies.html or alternatively contact sure@sunderland.ac.uk.
BotDet: A System for Real Time Botnet Command and Control Traffic Detection

IBRAHIM GHAFLIR1,2, VAACLAV PRENOSIL1, MOHAMMAD HAMMOUDEH3, THAR BAKER4, SOMAIL JABBAR5, SHEHZAD KHALID6, AND SARDAR JAF2
1Faculty of Informatics, Masaryk University, Brno 602 00, Czech Republic
2Department of Computer Science, Durham University, Durham DH1 3LE, U.K.
3Faculty of Science and Engineering, Manchester Metropolitan University, Manchester M1 5GD, U.K.
4Department of Computer Science, Liverpool John Moores University, Liverpool L3 5UA, U.K.
5Department of Computer Science, National Textile University, Faisalabad 37610, Pakistan
6Department of Computer Engineering, Bahria University, Islamabad 44220, Pakistan

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 system: 1) we have developed four detection modules to detect different possible techniques used in botnet C&C communications and 2) 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 unauthorized 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) customized 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 analyze 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 behavior 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 behavior 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 analyzing 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 analyzing 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 behavioral 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 categorizing 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 behavior 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 analyzed 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 servers 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 in 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.

 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 analyzer.

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 utilized 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.](image)

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

**Algorithm 1 Implementation Pseudo-Code of MIPD**

```plaintext
1: Input: blacklist of malicious IP addresses (t_ip_blacklist table)
2: Input: new_connection event
3: Check if the connection is to a malicious IP:
4:     if the connection destination IP is in t_ip_blacklist then
5:         if the connection is established by one of the network’s hosts then
6:             if MIPD didn’t raise the same alert during the past 24 hours then
7:                 goto Check if the connection is from a malicious IP:
8:             else
9:                 Raise an alert (ip_alert)
10:                Record the generated alert
11:                Notify the network security team via email
12:                Deny repeating the same alert during the next 24 hours
13:         else
14:             end if
15:     else
16:         goto Check if the connection is from a malicious IP:
17:     end if
18: end if
19: Check if the connection is from a malicious IP:
20:     if the connection source IP is in t_ip_blacklist then
21:        if the connection is oriented to one of the network’s hosts then
22:            if MIPD didn’t raise the same alert during the past 24 hours then
23:                goto End
24:        else
25:            end if
26:     else
27:         Raise an alert (ssl_alert)
28:         Record the generated alert
29:         Notify the network security team via email
30:         Deny repeating the same alert during the next 24 hours
31:     end if
32: else
33:     goto End
34: end if
35: goto End
36: end if
37: End
```

When the connection is from a malicious IP, the same procedure as in when the connection is to a malicious IP 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.

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.
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) SHA1 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].

![Methodology of the malicious SSL certificate detection.](image)

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, 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 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.

**Algorithm 2 Implementation Pseudo-Code of Intelligence-Based MSSLD**

1: **Input**: blacklist of certificates’ hashes
2: for each SSL encrypted connection do
3: Compute the certificate hash
4: if the certificate hash matches with the blacklist then
5: if the connection is established by one of the network’s hosts then
6: if MSSLD didn’t raise the same alert during the past 24 hours then
7: Raise an alert (ssl_alert)
8: Record the generated alert
9: Notify the network security team via email
10: Deny repeating the same alert during the next 24 hours
11: end if
12: else if the connection is oriented to one of the network’s hosts then
13: if MSSLD didn’t raise the same alert during the past 24 hours then
14: Raise an alert (ssl_alert)
15: Record the generated alert
16: Notify the network security team via email
17: Deny repeating the same alert during the next 24 hours
18: end if
19: else
20: goto End
21: end if
22: else
23: goto End
24: end if
25: end for
26: End

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 analyzed and, for detecting domain flux attacks and identifying infected hosts, the same IP address is constrained by a threshold for DNS query failures [40].

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_message 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 (c$dns$rcode_name=="NXDOMAIN"). If these two conditions are met, the source IP address that searches 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 analyzed and, for detecting domain flux attacks and identifying infected hosts, the same IP address is constrained by a threshold for DNS query failures [40].

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_message 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 (c$dns$rcode_name=="NXDOMAIN"). If these two conditions are met, the source IP address that searches 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 analyzed and, for detecting domain flux attacks and identifying infected hosts, the same IP address is constrained by a threshold for DNS query failures [40].
Algorithm 4 Implementation Pseudo-Code of DFD

1: Input: dns_failure_threshold
2: Extract DNS traffic
3: Input: dns_message event
4: if the connection is established by one of the network’s hosts then
5: if dns_message event is due to DNS error of NXDOMAIN then
6: if the host IP is not in t_dns_failure table
7: write host IP into t_dns_failure
8: host IP counter ← 1
9: else
10: Increase host IP counter by 1
11: if host IP counter > dns_failure_threshold then
12: Delete host IP from t_dns_failure
13: Reset host IP counter to zero
14: if DFD raised the same alert during the past 24 hours then
15: goto End
16: else
17: Raise an alert (domain_flux_alert)
18: Record the generated alert
19: Notify the network security team via email
20: Deny repeating the same alert during the next 24 hours
21: end if
22: else
23: goto End
24: end if
25: else
26: goto End
27: end if
28: else
29: goto End
30: end if
31: end if
32: End

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].

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.

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 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 utilizes 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.

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
Algorithm 5 Implementation Pseudo-Code of TorCD

```
1: Input: Tor servers list (t_tor_server table)
2: Input: connection_established event
3: Check if the connection is to a Tor network:
4: if the connection destination IP is in t_tor_server then
5:   if the connection is established by one of the network’s hosts then
6:     goto Check if the connection is from a Tor network:
7:   elif TorCD raised the same alert during the past 24 hours then
8:     goto Check if the connection is from a Tor network:
9:   else
10:    Raise an alert (tor_alert)
11:    Record the generated alert
12:    Notify the network security team via email
13:    Deny repeating the same alert during the next 24 hours
14:   end if
15:   goto Check if the connection is from a Tor network:
16: end if
17: else
18:   goto Check if the connection is from a Tor network:
19: end if
20: Check if the connection is from a Tor network:
21: if the connection source IP is in t_tor_server then
22:   if the connection is oriented to one of the network’s hosts then
23:     goto End
24:   else
25:     Raise an alert (tor_alert)
26:     Record the generated alert
27:     Notify the network security team via email
28:     Deny repeating the same alert during the next 24 hours
29:   end if
30: else
31:   goto End
32: end if
33: end if
34: end if
35: end if
36: end if
37: End
```

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 analyzed. 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 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 center [51]. None of the analyzed pcap files contained known bad SSL certificates, this module was tested in the second scenario. All pcap files had been analyzed 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.

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 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

**FIGURE 8.** The architecture of the correlation framework.

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

<table>
<thead>
<tr>
<th>Detection module</th>
<th>TPR</th>
<th>FPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious IP</td>
<td>63%</td>
<td>39.2%</td>
</tr>
<tr>
<td>Malicious SSL</td>
<td>86.4%</td>
<td>24.9%</td>
</tr>
<tr>
<td>Domain flux</td>
<td>65%</td>
<td>33.8%</td>
</tr>
<tr>
<td>Tor connection</td>
<td>88%</td>
<td>12.5%</td>
</tr>
</tbody>
</table>
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.

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 analyzed malware samples were Trojan.Win32.Inject.sbbz, Trojan.Win32.Stasserbaaz, 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.

The HEUR: Trojan.Win32.Generic malware (MD5, fbb354f7673fb81927a59008d9fd3a6) 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 center 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, 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.sbbz, also known as Torrent-Locker, (with MD5 hash aabe2844ee61e1f2969a7a9e1355a99) and Trojan.Win32.Stasserbaaz 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.

<table>
<thead>
<tr>
<th>Detection module</th>
<th>TPR</th>
<th>FPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious IP</td>
<td>61%</td>
<td>34.7%</td>
</tr>
<tr>
<td>Malicious SSL</td>
<td>42.6%</td>
<td>0%</td>
</tr>
<tr>
<td>Domain flux</td>
<td>83%</td>
<td>27.3%</td>
</tr>
<tr>
<td>Tor connection</td>
<td>63.3%</td>
<td>29%</td>
</tr>
</tbody>
</table>

TABLE 4. The TPR and FPR of the correlation framework.

<table>
<thead>
<tr>
<th>Correlation framework</th>
<th>TPR</th>
<th>FPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Based on one detection module</td>
<td>91.8%</td>
<td>45%</td>
</tr>
<tr>
<td>Based on two detection modules</td>
<td>79%</td>
<td>16.8%</td>
</tr>
<tr>
<td>Based on three detection modules</td>
<td>52.6%</td>
<td>7%</td>
</tr>
<tr>
<td>Based on four detection modules</td>
<td>31.1%</td>
<td>0%</td>
</tr>
</tbody>
</table>


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.

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.

REFERENCES


Authors’ photographs and biographies not available at the time of publication.