Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

A flow-based approach for Trickbot banking trojan detection

Ali Gezer[a,*], *Gary Warner*[b], *Clifford Wilson*[b], *Prakash Shrestha*[b]

[a] *Electronic and Telecommunication Technology, Erciyes University, Kayseri, Turkey*
[b] *Computer Science, University of Alabama at Birmingham, Birmingham, AL, US*

ARTICLE INFO

Article history:
Received 29 August 2018
Revised 3 March 2019
Accepted 13 March 2019
Available online 15 March 2019

Keywords:
Trickbot
Banking trojan
Machine learning
Anomaly traffic detection
Dynamic analysis
Random Forest

ABSTRACT

Nowadays, online banking is an attractive way of carrying out financial operations such as ecommerce, e-banking, and e-payments without much effort or the need of any physical presence. This increasing popularity in online banking services and payment systems has created motivation for financial attackers to steal customer‘s credentials and money. Banking trojans have been a way of committing attacks on these financial institutions for more than a decade, and they have become one of the primary drivers of botnet traffic. However, the stealthy nature of financial botnets requires new techniques and novel systems for detection and analysis in order to prevent losses and to ultimately take the botnets down. TrickBot, which specifically threatens businesses in the financial sector and their customers, has been behind man-in-the-browser attacks since 2016. Its main goal is to steal online banking information from victims when they visit their banking websites.

In this study, we utilize machine learning techniques to detect TrickBot malware infections and to identify TrickBot related traffic flows without having to analyze network packet payloads, the IP addresses, port numbers and protocol information. Since command and control server IPs are updated almost daily, identification of TrickBot related traffic flows without looking at specific IP addresses is significant. We adopt behavior-based classification that uses artifacts created by the malware during the dynamic analysis of TrickBot malware samples. We compare the performance results of four different state-of-the-art machine learning algorithms, Random Forest, Sequential Minimal Optimization, Multilayer Perceptron, and Logistic Model to identify TrickBot related flows and detect a TrickBot infection. Then, we optimize the proposed classifier via exploring the best hyperparameter and feature set selection. Looking at network packet identifiers such as packet length, packet and flag counts, and inter-arrival times, the Random Forest classifier identifies TrickBot related flows with 99.9534% accuracy, 91.7% true positive rate.

1. Introduction

In recent years there has been considerable growth in the number of users in online banking (Szopinski, 2016, Lopez and Martin, 2017). The increasing trend in online banking operations has made it a focus of interest for fraudulent online banking activities. The number of malware and exploits focused on online banking system vulnerabilities has also been steadily growing during past years (Carminti et al., 2018). Cybercriminals target online banking using different techniques such as phishing emails, drive-by downloads, key logging, and infecting victims with trojanized software with the purpose of committing financial fraud by stealing user credentials and, eventually the money of those customers. In this context,

* Corresponding author.
E-mail address: aligezer@uab.edu (A. Gezer).
https://doi.org/10.1016/j.cose.2019.03.013
0167-4048/© 2019 Elsevier Ltd. All rights reserved.

financial botnets have been used to support a range of malicious activities including DDoS attacks, keylogging, spam, phishing, and web injections (Ono et al., 2007). A financial botnet is a distributed network of compromised machines that can be remotely controlled by the same command and control server with the aim of attacking financial customers (Khattak et al., 2014). Banking trojans have been the primary drivers of botnet traffic and malicious activity, and they have become a well-known threat for banking institutions all around the globe (Peotta, 2011).

Once the customer device is infected with the banking trojanized software, it has become a zombie and has gained the capability of being monitored or even controlled by the threat actor. Generally, financial bots choose the following ways for carrying out their goals.

- Modify the source code of targeted websites via injecting HTML code or JavaScript code to monitor access to the modified webpage.
- Redirect the victim to a fake website mimicking the genuine banking webpage.
- Steal the data entered into HTML forms such as user banking account credentials and other personally identifiable information (PII)
- Download and execute additional binaries such as configuration files, modules, or other malware to obtain extra functionalities.

Specifically threatening businesses in the financial sector, TrickBot has been behind man-in-the-browser (MitB) attacks since 2016 via injecting malicious code into ongoing browser sessions (Kremez and Burbage, July 2017). It first became popular in October 2016 when it hit corporate and business accounts through a targeted malvertising campaign (Keshet, 2016). Its main aim is stealing online banking information from browsers when victims are visiting banking webpages. TrickBot has strong code similarities with the Dyre banking trojan, a botnet that was used in numerous spam and phishing campaigns creating damage worth tens of millions of dollars to Western banks and businesses in the US, the UK, and Australia (Pauli, 2016). Dyre related financial crimes eased off with the arrest of its authors in February 2016.

An important task of network management is the accurate identification of anomaly traffic in the networks to mitigate or block malicious traffic. Due to the stealthy nature of financial malwares, it is difficult to identify and complex to analyze in an automated way (Riccardi et al., 2010). Machine learning based classification is a well-known technique for the identification of anomalies in the network traffic. There are two common techniques applied for the classification of network traffic: signature-based and behavior-based. In signature-based classification, a common sequence of bytes that appear in the binary code is used to identify and detect the family of malware, but this requires inspection of the payload of every packet (Mohaisen and Alrawi, 2013). If the packet payload is not encrypted, the mentioned approach might be a promising way to detect malicious traffic. However, most applications, especially malicious ones, use encrypted communication which makes signature-based classification much more difficult. This approach also raises privacy concerns while examining user generated data and demands high computational resources and storage (Alshammari and Zincir-Heywood, 2007).

In this study, we adopt the behavior-based approach for the classification of TrickBot related network flows. Behavior-based classification uses artifacts created by the malware during the execution of the malware. We attempt to provide some insights into the identification of a TrickBot infection via the evaluation of network traffic flows through a machine learning approach. A test classifier is proposed to distinguish between benign traffic streams and the malicious TrickBot traffic streams. Malicious traffic captures were generated using real-world TrickBot malware infections, while benign traffic captures are generated through the network interaction with popular Internet domains. During the dynamic analysis of TrickBot, we also visited many popular websites in Alexa top 500 list to generate benign HTTP traffic. The designed classifier can be used to determine the class of unknown traffic flows to reveal any infection with a TrickBot trojan and provides a promising way for classifying TrickBot related traffic flows without taking into account application payloads, specific IP addresses, port numbers, and protocol information. TrickBot C&C server IPs are updated very frequently and utilized port numbers could be assigned dynamically. Therefore, taking into account IP addresses, port numbers and protocol information could overfit the built classifier and may cause a biased output in real life scenarios when we come across new version of TrickBot. Besides, TrickBot banking Trojan is updated very frequently, almost nearly every day. Therefore, training the classifier via taking into account IP addresses, port numbers may cause a biased output that relies on particular IP addresses and port numbers. We carried out dynamic analysis of more than 300 TrickBot malware samples over a 9 month period and captured their network traffic with a protocol analyzer. Later, the captured files were utilized as training data to build our TrickBot classifiers and test data to evaluate the performance of the built model.

2. Related work

Exploiting the inherent vulnerabilities of botnet's command and control servers might be a way of lessening the impact of banking trojans. In Watkins et al. (2014), an offensive measure is proposed to fight with the command and control (C&C) server, or bot-herder of the Zeus botnet. The Zeus botnet family has been on the stage for more than a decade and is one of the oldest banking trojans. In 2017, the Zeus botnet and its variations made up about 28% of all banking related malware (Top 10 Banking trojans for 2017). The leakage of Zeus source code in May 2011 has given motivation for financial malware authors to implement Zeus features in their cybercrime tools. ICE IX and Citadel are a couple of malware families which are based on the leaked Zeus source code. Like Zeus, the primary goal of Citadel is to deliver web injects. The modified website with the injected JavaScript code or new form fields can be used to commit complex man-in-the-browser attacks (Milletary, 2012).

A behavior-based classification technique is used in Mohaisen and Alrawi (2013) that uses artifacts created by the malware during the execution for the identification of the Zeus banking trojan. For the classification, they extract 65 features that are unique and robust for identifying malware families. They show that artifacts like file system, registry, and network features can be used to identify distinct malware families with high accuracy, in some cases as high as 95%. However, they didn't perform any feature selection to optimize their model in this study. Existing attack techniques and the vulnerabilities that affect current internet banking systems are classified in (Peotta, 2011). Based on the elaborate analysis of current security models, they propose a guideline for designing secure internet banking systems which are not affected by the present attacks and vulnerabilities. In the mentioned study, the authors didn't focus on any particular banking trojan to reveal their particular attack vector and infection strategy.

A cyber kill chain (CKC) based taxonomy for banking trojan features is proposed in Kiwia et al. (2018) which can be used to inform detection and mitigation strategies. Their taxonomy is built based on the analysis of 127 banking Trojan samples collected from December 2014 to January 2016 from a real-world banking organization in the United Kingdom. They didn't address any specific signature to any particular banking Trojan in the mentioned study.

A set of malware behaviors that are present in the selected banking malware families are identified in Black et al. (2018). The aim of the paper is to examine the similarity in the behaviors of this selected set of malware which serve similar purposes but have different and somewhat unknown origins. This automatic classification of the behaviors present in a malware sample presents a representation of malware capability at a higher level of abstraction. A system called Zarathustra is presented in Criscione et al. (2014) to automatically characterize the web inject-based behaviors, regardless of the underlying implementation. They evaluated their method against 213 real, live URLs of banking websites and 56 distinct samples of the Zeus banking trojan. None of these studies have characterized TrickBot banking Trojan behavior.

Machine learning techniques are frequently utilized to detect the particular flows of network traffic. In Alshammari and Zincir-Heywood (2007, a Ripper learner is used to identify SSH traffic with 99% detection rate. They exclude TCP and UDP port numbers from the feature set to avoid any biased model that rely on dynamically assigned port numbers. Haddadi et al. compared some flow exporters to increase the botnet traffic classification with machine learning techniques. They performed their test with Citadel, Cutwail, Kelihos, Zeus and Conficker botnet dataset and show that Tranalyzer flow exporter gave the best result (Haddadi and Zincir-Heywood, 2016). However, they didn't perform any test with Trickbot botnet.

In Soniya and Wilscy (2016), a bot detection system is proposed which aims to detect randomized bot C&C traffic for early bot detection with Multi-Layer Perceptron classifier. Temporal persistence that spans time intervals larger than a user session is used to differentiate benign and malicious bot traffic. They performed a test with 43 bot samples such as Banbra, Dedler, Ramnit, and Sasfis. In Zhao et al. (2013), a botnet detection system is proposed based on traffic behavior and flow intervals with machine learning technique. They didn't use application layer data as an attribute. However, they exploited IP addresses, source and destination ports and protocol information in their feature set which may result in a biased model that rely on specific IP addresses, and dynamically changeable port numbers. In Black et al. (2018), a survey study is performed to analyze the similarities in banking malware behaviors. They focused on assembling a dataset of malware behavior using static methods which differentiates from our study that applies dynamic analysis to detect TrickBot banking trojan.

To the best of our knowledge, this is the first study which focuses on the detection of TrickBot banking trojan infection and identification of its abnormal traffic in networks. In this study, we employ a behavior-based machine learning approach to identify TrickBot related traffic flows within a network. The built classification model could detect any TrickBot infection without needing to evaluate the application payloads of packets or IP addresses.

3. TrickBot banking trojan

TrickBot is the first and only banking trojan which targets the customers of major banks that cover many geographies and language zones around the world. In the beginning, TrickBot targeted the financial institutions located in the UK, Australia, and Switzerland. Nowadays, TrickBot operators have been carrying out their redirection attacks against banks in 19 different countries (Kessem, 2017). According to the strings found inside the code, TrickBot authors named it TrickBot or Trickloader. Its internal makeup, methods for infecting new endpoints, and modular structure show a high resemblance to the Dyre banking trojan. It tricks users into entering their credentials such as login details, personally identifiable information, and financial authentication codes into phishing and fraudulent banking websites. The TrickBot malware has been continually upgraded with new modules and configuration files improving its ability to steal credentials, to expand distribution vectors, and to prevent detection by defenders.

TrickBot infects victim's host with an initial malware binary. Then, this binary starts downloading different modules which are responsible for various operations. At this point in the infection, if Chrome, Firefox, Internet Explorer, or Edge is opened, it will be exploited to inject malicious code on targeted websites. It performs a highly skilled redirection of browsing. When the victim visits a targeted banking page, it injects HTML or JavaScript code into the page to redirect user's data to another server. While the malware keeps a live connection with the bank‘s genuine webpage, the online banking users are displayed a webpage updated with the attacker's malicious code. Meanwhile, the alternate webpage on the user‘s screen displays the bank‘s correct URL in the address bar with the bank‘s genuine digital SSL certificate. The bot achieves persistence by adding itself as a task in Windows Task Scheduler. If an attempt is made to kill the process or if the computer is restarted, TrickBot is automatically restarted by the Task Scheduler Engine.

The TrickBot binary is composed of several layers as shown in Fig. 1. The first layer is used for protection. It carries the encrypted payload and tries to hide it from AV software. The second layer is the main bot loader which determines

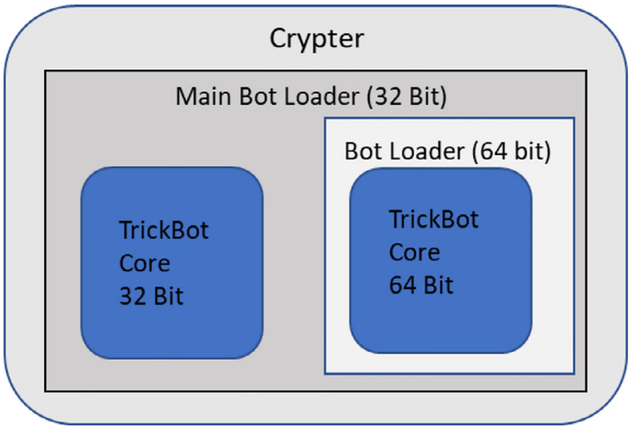

Fig. 1 – TrickBot layers.

whether to deploy a 32-bit or 64-bit payload, depending on the current system's architecture. At the beginning this malware fetches information about the victim‘s operating system in order to use and download the correct architecture for the related resource files. Most of the communication with its main C&C is encrypted. The downloaded modules and configuration files are encrypted by AES (Malwarebytes Labs 2016).

3.1. *Behavioral analysis*

Once TrickBot binary is executed, it dynamically extracts code from itself, puts it into a heap space, and then calls its entry point. TrickBot's initial network activity is an HTTP request to public service sites, such as "myexternalip.com" to retrieve the victim‘s public IP address for the aim of sharing this information with its command and control server (Zhang, 2016). After this, it will continually load encrypted module and configuration files from the botnet command and control (C&C) servers. These files define how the malware will modify the contents of targeted bank webpages. Upon decryption, it can be seen that the main configuration file (config.conf) includes information about its version, group tag, and IP addresses of its C&C servers. The config.conf file contains a list of IPs from where TrickBot will download related resource files. In this context, the configuration file contents are important because system administrators can block access to these IPs. The downside of this approach is that it's reactive, opposed to proactive. TrickBot releases a new version of this main configuration file on (almost) a daily basis, meaning this list of blocked IPs would need to be constantly updated. In this study, we initially utilized this list of C&C server IPs found in TrickBot's configuration files to label network flows for training and testing purposes.

When a system is infected with TrickBot Trojan, the following steps are carried out.

- Duplicates itself into a subdirectory within the%AppData %\Roaming folder.
- Creates identifier files (group_tag and client_id) and a folder named Modules, intended for the storage of various modules and the inject configuration files.
- Creates a task in Windows Task Scheduler to ensure persistence upon termination or a system restart.
- Connects with a server (via an HTTP GET request) to reveal victim's public IP.
- Goes through its list of C&C server IPs to download modules and configuration files.
- The main process opens multiple svchost.exe (usually between 4 and 7) each focusing on different tasks correlating with the different downloaded modules.
- When a targeted bank‘s webpage is visited, an inject of code is placed in the browser to redirect user to a falsified webpage.

Group_tag and client_id are generated locally and used to identify the individual bot and the campaign name to which it belongs. Group_tag and client_id files are not encrypted, both files contain text in Unicode. Client_id is a unique identifier for the host system which contains the Windows computer name of the infected machine, the build version of the Windows operating system, and an arbitrarily generated 32-character string. The created Modules folder will be used for storing encrypted plug-and-play modules. In the memory strings of the running malware binary, the "built in" main configuration (similar to the "config.conf" file downloaded later) can be found by searching for <mcconf>. This string contains the version number of the config, group tag, and the list of C&C IPs. TrickBot will (seemingly randomly) attempt to connect with these different IPs, but when an active C&C is found, a TCP connection is established to download additional resource files: an updated main configuration (config.conf), inject configuration files, and additional modules. As these modules are downloaded, the main TrickBot process will spawn malicious svchost.exe sub-processes corresponding with each module and its functionality. After downloading config. conf, dinj, sinj, and dpost files, we can use the TrickBot decoder tools to decrypt the resource files (TrickBot-Toolkit).

Common modules and functionality:

- systeminfo32 - gather information about the infected system
- injectdll32 - injects into the browser and performs web injects
- mailsearcher32 - searches through personal files to gather a list of email addresses
- sharedll32 - allows the malware to move laterally via network shares

Common configuration files and purpose:

- config.conf - main TrickBot config which includes the latest server list, version, and group tag
- dinj - dynamic web inject configuration
- sinj - static web inject configuration
- dpost - server which the dynamic web injects will send intercepted requests to
- mailconf - server to send harvested email list to

4. Methodology

4.1. *Machine Learning approach for identifying Trickbot flows*

As discussed earlier, we focus on the detection of TrickBot infection and identification of TrickBot related network flows. When TrickBot trojan infects a system, its initial behavior is a network activity with an HTTP request to some public service sites, such as “myexternalip.com”, “icanhazip.com” or “ip.anysrc.net” to retrieve the victim‘s public IP address. Afterwards, the compromised system tries to connect with one of the TrickBot C2 servers which is active for the time being. Some of the C2 server IPs are hardcoded into malware’s binary. Once the connection established, it will obtain new C2 IPs with the aim of continually downloading encrypted modules and configuration files, and exfiltration of stolen data. However, Ramnit banking trojan uses a domain generation algorithm (DGA) to contact with its C2 servers. Upon the infection, the trojan starts carrying out DNS queries for generated domain names. Another popular banking trojan, IcedID which exhibits totally a different network communication pattern, sets up a local proxy and redirects the Internet traffic through the proxy server.

TrickBot uses webinjects data in its dinj and sinj configuration files which consist of some URL links that include webinjects scripts. The webinjects consist of mostly Javascript code which is injected into the webpage while users are visiting legitimate online banking services. Dinj and sinj files are encrypted with AES so it is impossible to reveal the content of these files without decryption. TrickBot trojan uses TLS/SSL network communications to avoid detection by intrusion detection systems (IDS) and intrusion prevention systems (IPS) for downloading module and configuration files. Sometimes it uses different ports for TLS/SSL communication other than 443. However, custom TCP/IP ports for HTTP and HTTPS are frequently used to download configuration files or post the stolen data. TrickBot exploits virtual network computing (VNC) which gives the attacker the possibility of taking control of the victim’s system. TrickBot also utilizes screenshot behavior to reveal any use of virtual keyboard for entering user credentials into a banking session. Mostly, these facilities are gained through the execution of downloaded module and configuration files from C2 servers. After executing these modules, they also need a network communication to fulfill their destructive objectives. However, due to the encrypted content of exchanged packets, it is difficult to comprehend their motive. Therefore, revealing the pattern of TrickBot network communication would help us to detect any TrickBot infection on a system. For this aim, we focus on the communication patterns between TrickBot C2 servers and compromised system. In our implementation, the machine learning technique is used to map the instances of network flows into the class of TrickBot flows and non-TrickBot traffic flows. Each traffic flow is described by a set of statistical features which can be calculated from one or more packets. So, each flow is characterized by the same set of feature names, however different feature values. In our methodology, we utilize a supervised machine learning approach to classify traffic flows into the class memberships. In supervised classification, the classes should be pre-defined before training the system. First, a classification model is built using a training data set which includes instances of each class (Williams et al., October 2006). Then, this model is used to predict the class membership for new traffic flows represented as statistical features.

As could be tracked from Malwarebytes Labs (2018), the TrickBot banking trojan is updated almost every day with new functionalities in its defense and infection strategy. Through the content of decrypted config.conf files, we observe that IPs never seen before might be used as an intermediate TrickBot C&C server with new TrickBot versions. Therefore, we built a TrickBot traffic classifier without taking into account newly discovered specific C&C IPs. As mentioned, the C&C IPs might change over the course of version updates. Because of that, only looking at previously seen IPs makes the detection of new versions of TrickBot futile. The classifier may be overfitted in the training and testing phase. In this case, it could produce biased outputs when faced new TrickBot versions which use updated C&C IPs, never been used before. Therefore, we propose a supervised machine learning classification which identifies TrickBot flows without taking into account specific IP addresses and packet payloads. We also don’t use protocol information and port numbers as attributes for the training of the proposed classifier. Port numbers could be assigned dynamically, and protocols could be hided via using VPN. Therefore, any protocol and port number dependent model may produce biased outputs.

4.2. *Data Collection via dynamic analysis*

During a 9-month period, we dynamically analyzed many TrickBot malware samples from version 2 to version 184. Once the sample is executed, network communication occurs between the compromised computer and a website to reveal the infected computer‘s public IP address. During the dynamic analysis, we observe the%AppData%\Roaming folder to see if there is any newly created folder related to the TrickBot infection. The names of this newly created folder have recently been “winapp,” “netdefender,” and “services” according to TrickBot version.

Before the execution of a TrickBot malware sample on a virtual machine, Wireshark protocol analyzer is set to capture the network traffic with some pre-defined filtering rules to not to capture ARP and some local broadcast packets. Our implementation for the dynamic analysis of TrickBot is illustrated in Fig. 2. We also run Process Hacker to track running processes in the virtual machine. In this way, it could be easily observed when the TrickBot process starts, terminates, deploys in Windows Task Manager, and how many svchost processes are launched by the main executable to run the downloaded module DLL files. We analyzed more than 300 different TrickBot samples over 9 months to observe how this trojan develops and to observe new behavioral patterns. Does it send any spam? Is there any web-injection when visiting a banking web page? Is it directing users to any phishing pages? While we are looking for the answers to these questions through dynamic analysis, we also capture its network traffic via Wireshark protocol analyzer to observe the interaction between the host and TrickBot C&C servers. Especially,

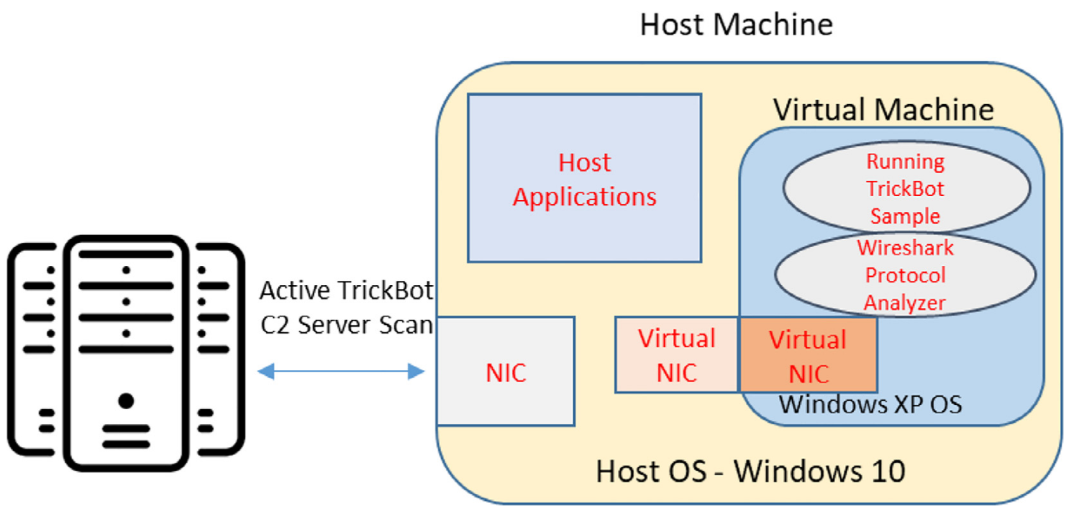

Fig. 2 – The infrastructure used for executing TrickBot samples in secure environment.

while it is downloading configuration and module files, a high rate of packet exchange occurs. We recognized a network communication pattern and determined that this deduction could be used to identify TrickBot related flows in a network to identify infections by this trojan. During the capturing process of network traffic while the TrickBot sample is running, we intentionally generate benign HTTP traffic through the interaction with popular domains. Such benign traffic is generated via visiting some well-known websites such as university domains, online newspaper, and some well-known social media websites such as Facebook, Instagram, Twitter. So captured .pcap files also include normal web traffic opposed to containing only TrickBot specific traffic. We run each sample with different durations. Sometimes, it is executed only a few minutes to observe the network traffic pattern at the beginning or sometimes more than 3 h to observe the injection when visiting banking web pages. We captured more than 300 .pcap files which include both TrickBot related traffic and benign traffic. This variance in traffic captured in the .pcap files is greatly beneficial for training and test data for our proposed model.

4.3. *Feature Extraction and labeling*

NetMate toolset is used to process captured .pcap files, then compute features according to the flows. NetMate is an open source tool for *nix systems and used to calculate network flow statistics. At the end of processing, it generates feature values regarding flows in the captured trace files. A flow is a sequence of packets from one source IP and source port to another destination IP and destination port over a certain period of time. Each network flow is described by a set of statistical features and associated feature values. A feature is a descriptive statistic that can be calculated from one or more packets that belongs to a flow (Riccardi et al., 2010). Once the flows are generated, source/destination IPs, port numbers, and protocols are excluded from the future set. For UDP flows, a flow timeout value is determined to terminate connection. In our study, we adopt two criteria for the termination of TCP flows. One is the normal tear down of connection, another is flow time out value. Whichever happens first, the connection is terminated according to that criteria. 600 s flow timeout value is chosen where this corresponds to IETF real-time traffic flow measurement working group architecture (IETF). For UDP and TCP flows, we take into account the flows which have more than one packet in each direction, and have more than one-byte payload. So, it excludes flows with no payload or unsuccessful flows. Following statistical features are extracted from captured trace files.

Data link layer portion is excluded from the packet length. Packet lengths are calculated according to IP layer length. Inter arrival times are in microsecond precision. Totally, 37 flow features are taken into account for building a classifier. Flows are bidirectional, and the first packet determines the direction of the flow. At the end of flow statistic conversion process with Netmate, in total 334.529 unique flows are obtained to be provided as an input vector to the machine learning algorithms. So, each network flow is described by a set of statistical features and associated feature values. The source IP and destination IP are excluded from the feature set to ensure that the built model is not dependent on specific IPs. Port numbers and protocol information are also excluded from feature set to not to create a bias model to avoid any overfitting. Any classification that relies on port numbers or protocols may cause building a biased classifier that doesn't work well with updated TrickBot versions.

For supervised classification problem, a classification model must be built on according to the previously labeled flow instances. For class labeling process, we built a library file which contains all intermediate C&C server IPs from all analyzed TrickBot versions up to 184 which was the latest version at the time. A python script was written to compare the IPs in each flow with known TrickBot C&C server IPs, and each flow was labeled according this determination. As discussed before, one way might be to use memory strings to extract C&C intermediate server IPs for class labeling. Another option is for every downloaded config.conf file, TrickBot resource file decoder could be utilized to extract C&C server IPs. In both ways, TrickBot sample should be executed for each version to obtain

IPs from memory strings or the content of the config.conf files. Extraction of IPs in this way is a troublesome operation and sometimes problematic. For example, we have observed that during the execution of the sample, used C&C IPs for downloading TrickBot resource files could be erased from memory strings after some time. Therefore, some IPs might be missed in this way.

We decided to build a TrickBot C&C IP library file using the config.conf files at Malwarebytes Labs (2018). This site is maintained by a separate researcher and includes all decrypted config files up to the current version, 184. We found the data contained in this repository was consistent with our own findings, but did offer addition configurations on versions we had missed. Therefore, we wrote a Python script to download all of these encrypted config.conf and ServConf files and extract C&C intermediate IPs from the files downloaded. We carry out this operation for each botnet of TrickBot. After the elimination of repeated IPs, a TrickBot C&C IPs library file was created which contains unique 2095 C&C IPs. For our class labeling process, we used this library.

4.4. *Classification Models*

As a classifier, we used several state-of-the-art machine learning algorithms that used commonly in machine learning and knowledge discovery applications namely, Random Forest, Multilayer Perceptron (MLP), Logistic model, and Sequential Minimal Optimization (SMO) to determine an efficient classifier for the problem at hand. While we are testing mentioned classifiers, we utilized the feature set given in Table 1. Then, we perform a hyper parameter optimization and feature selection to increase overall accuracy and TP rate for the discovered classifier. We also utilized Weka, an open source machine learning tool, for our research. In this section, we provide a brief description on each of the machine learning algorithms we benefited from in our study.

4.4.1. *Random Forest*

Random Forest is an ensemble learning approach for classification that consists of a set of decision trees. It generates each of these decision trees from a randomly selected subset of training data samples. While classifying a new data instance, each tree participates and predicts, and a final result is computed based on the majority voting from the trees. In laymen's terms (Breiman, 2001), consider training set of five data instances [T1, T2, T3, T4, T5] with corresponding labels as [L1, L2, L3, L4, L5]. Random Forest may create three decision trees based on three different subset of training instances as follow – [T1, T2, T3], [T1, T3, T5], [T2, T4, T5]. Each tree of Random Forest makes a prediction while classifying a new instance, and a final result is computed as the majority of votes from its decision trees. Since the final decision of Random Forest is based on the majority voting from its multiple decision trees, it works well even in the presence of "noise" (abnormal data). A single decision tree may be influenced and affected by the noise, but using and aggregating the results from multiple decision trees reduces the effect of noise, thereby making it robust even with noise present.

Table 1 – Considered features for the classifier.

Total forward packets	Std. of backward inter-arrival time
Total forward volume	Duration
Total backward packets	Min time flow was active before idle
Total backward volume	Mean time flow was active before idle
Min forward packet length	Max time flow was active before idle
Mean forward packet length	Std. of time flow was active before idle
Max forward packet length	Min time flow was idle before active
Std. of forward packet length	Mean time flow was idle before active
Min backward packet length	Max time flow was idle before active
Mean backward packet length	Std. of time flow was idle before active
Max backward packet length	Sub-flow forward packets
Std. of backward packet length	Sub-flow forward bytes
Min forward inter-arrival time	Sub-flow backward packets
Mean forward inter-arrival time	Sub-flow backward bytes
Max forward inter-arrival time	Forward push counter
Std. of forward inter-arrival time	Backward push counter
Min backward inter-arrival time	Total header bytes in forward direction
Mean backward inter-arrival time	Total header bytes in backward direction
Max backward inter-arrival time	

4.4.2. *Multilayer Perceptron*

Multilayer Perceptron (MLP) is a class of artificial neural network that uses back propagation to classify instances. It consists of at least three layers of nodes. Specifically, it consists of an input layer to receive the input feature vector, an output layer that makes a decision or prediction based on the supplied feature vector, and in between those two, an arbitrary number of hidden layers that are the true computational engine of the Multilayer Perceptron. Generally, a set of input-output pairs is used to train the MLP model through which it learns the correlation (or dependencies) between those inputs and outputs. During the training phase, parameters of the MLP model, specifically the weights and biases, are adjusted in order to minimize the error. Backpropagation is used to make those weight and bias adjustments relative to the error. The error can be measured in a variety of ways, including by root mean squared error (RMSE).

4.4.3. *Sequential Minimal Optimization (SMO)*

It is a class of classifier implemented in Weka tool that we utilize for our classification modeling. At its core, it implements John Platt's Sequential Minimal Optimization algorithm (SMO) to train a support vector machine (SVM) classifier. The training of the SVM classifier may raise the quadrative programming problem, and SMO is an algorithm to solve such quadrative problems. Specifically, SMO is an iterative algorithm for solving the optimization problem. It breaks a complex and huge problem into a series of the smallest possible sub-problems, which are then solved analytically. Since the memory that

SMO requires is linear to the size of the training set, it allows SMO to handle very large training sets. Unlike standard SVM algorithms, that scale somewhere between linear and cubic in the training set size, since SMO avoids the matrix computation, an expensive computation, SMO scales somewhere between linear and quadratic in the training set size for various test problems. As the computation of SMO is dominated by SVM evaluation, it is fastest for linear SVMs and sparse data sets.

4.4.4. *Logistic Model*

The logistic regression model is one member of the supervised classification algorithm family. Logistic regression classifier is more similar to a linear classifier which uses the calculated logits (score) to predict the target class. Specifically, it measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function. The Simple Logistic classifier implemented in Weka uses Logit Boost with simple regression functions as base learners in order to fit the logistic models. Simple Logistic determines the optimal number of iterations for Logit Boost through cross-validation, and leads to automatic attribute selection for better classification performance.

4.5. *Performance metrics*

In our classification model, TrickBot instances correspond to positive class, and non-TrickBot instances correspond to negative class. Therefore, True Positive (TP) indicates the number of times the TrickBot instances are correctly classified as TrickBot, true negative (TN) indicates the number of times non-TrickBot instances are predicted as non-TrickBot, false positive (FP) indicates the number of times the non-TrickBot instances are classified as TrickBot, and false negative (FN) indicates the number of times TrickBot instances are classified as non-TrickBot.

To evaluate the performance of the classifiers, we used True Positive Rate (FPR), False Negative Rate (FNR), Precision, Recall, and *F*-Measure (as shown in Eqs. (1)–(5) as performance measures. Precision and FPR measure the accuracy of the classification model in correctly predicting non-TrickBot instances. Recall and FNR capture the error rate, for incorrectly predicting non-TrickBot instances. *F*-Measure is the harmonic mean of precision and recall that considers both the accuracy and the error of the classifier. To make the system accurate and robust to error, we would like to have recall, precision, and *F*-measure to be as close as 1.

$$TPR = \frac{TP}{TN + TP} \tag{1}$$

$$FPR = \frac{FP}{TP + FN} \tag{2}$$

$$precision = \frac{TP}{TP + FP} \tag{3}$$

$$recall = \frac{TP}{TP + FN} \tag{4}$$

$$F - Measure\ (F1 - Score) = 2 * \frac{precision * recall}{precision + recall} \tag{5}$$

4.6. *Results of tested classifiers*

Overall accuracy is one critical metric which should be focused on during the classification with machine learning technique. In some situation, especially when we are dealing with intrusion detection, dataset might me highly imbalanced. In real world network environment, sometimes minority flows are more dangerous than the majority flows. Mostly, the class of interest is often a rare one, sometimes less than 10% of the total classes (Khoshgoftaa et al., 2007). Data mining algorithms usually try to minimize the overall error via focusing on the majority class instead of minority class. However, this approach may cause a severe problem in intrusion detection because wrong identification of class of interest might cost much than the majority class.

There are 332,821 non-TrickBot flow connections and 1708 TrickBot flow connections in our dataset. Even if the built classifier finds all of the flows as non-TrickBot, still we could reach 99.48% accuracy. However, in this circumstance, the built classifier would not catch any Trickbot flows which may put the system at a risk. Therefore, this highly imbalanced data set is a big challenge that could be handled well with the built classifier. In similar cases, subsampling would be a solution to make the instance number equal for both classes. However, this is not a good solution in NIDSs, which has to handle mostly imbalanced dataset due to minority intrusions. In the case of subsampling, equal number of instances would overfit the training and testing data, and will perform poorly in real world scenarios. Therefore, we trained the classifier without any subsampling operations with the data. Because of that, we particularly focus on TP rate to evaluate our classifier instead of focusing only overall accuracy.

We developed a Python script to label all the aforementioned feature vectors obtained from dynamic analysis of more than 300 TrickBot samples. Malicious traffic content was generated using real-world TrickBot malware samples while benign traffic was created through the interaction with popular Internet domains. We visit popular websites in Alexa top 500 global sites during dynamic analysis to populate benign traffic. At the end of the labeling process with the developed script, 332,821 not-TrickBot and 1708 TrickBot network flows were obtained from all datasets.

WEKA, which is open source tool for data mining tasks, is employed for our classification problem. It has a user-friendly interface and includes many built-in machine learning algorithms. First, we utilize some efficient classifiers to build the model using 10-fold cross validation such as Random Forest, Multilayer Perceptron, Naive Bayes, Sequential Minimal Optimization, and Logistic Model. In this step, we aim to determine a robust classifier for the detection of TrickBot flows without any hyper parameter optimization. In 10-fold cross validation, the complete feature data set is randomly partitioned into 10 equal sized data sets. Each time, nine of the ten data sets are utilized as training data, and the remaining data set is chosen for the validation. This process is repeated 10 times until each data set is used one time for the validation and the remaining 9 are used for training. At the end, 10 results are averaged to obtain one singular estimation.

Table 2 – Confusion matrix for Random Forest Classifier.

Non-TrickBot flow	TrickBot flow	Classified as
332,813	8	**Non TrickBot flow**
190	1518	**TrickBot flow**

Once the aforementioned feature vectors are computed for all the data sets, Random Forest, Multilayer Perceptron, Sequential Minimal Optimization, and Logistic Model classifiers were tested for this classification one by one. A better result was obtained for the evaluation of the performance of the tested classifier, as true positive results get closer to 1, and false positive results get closer to 0. First, Random Forest classifier is employed over the obtained feature data set vector using 10-fold cross validation. According to tested classifier results, 334,331 instances were classified correctly, this is 99.9408%. Overall, 198 flow instances were misclassified. The confusion matrix for Random forest approach is given in the Table 2. It takes 460.82 s to build the model with the mentioned classifier.

The obtained results demonstrate that 8 Non-TrickBot flows are classified as TrickBot flow. The computed performance parameter for the built model is given in Table 3. The weighted average true positive rate of both classes is 0.999.

Second, we built a classifier with Multilayer Perceptron algorithm. The build time for this model is 1772.62 s which is slower than the Random Forest performance. The chosen classifier is run with learning rate 0.3, momentum 0.2, and training time 500. According to obtained results with Multilayer Perceptron, 99.7417% of the flow instances are classified correctly (333,665 out of 334,529). The obtained confusion matrix is given in Table 4. Still we have a very high accuracy; TP rate is very low which is 0.508. Approximately, our classifier could not catch half of the TrickBot flows. According to the confusion matrix, it could be observed that 23 non-TrickBot flow are classified as TrickBot flows, which indicates the false positive number for non-TrickBot flow. The performance parameter of the built model can be computed easily with the obtained confusion matrix. The obtained performance parameters for Multilayer Perceptron is given in Table 5.

Third, we employ Sequential minimal optimization (SMO) algorithm for the classifier with 1.0E-12 epsilon and 0.001 tolerance parameter. It takes 2200.84 s to build the model. Besides the build time performance of the model, other performance results are also not as good as previous models. We got 332,821 correctly classified instances and 1708 incorrectly classified instances which give 99.4894% classification accuracy. Obtained confusion matrix and performance results are given in Table 6 and Table 7, respectively for this classifier.

Table 4 – Confusion matrix for Multilayer Perceptron Classifier.

Non-TrickBot flow	TrickBot flow	Classified as
332,798	23	**Non TrickBot Flow**
841	867	**TrickBot Flow**

According to the confusion matrix results, although we get a very high accuracy, all of the TrickBot instances are classified as non-TrickBot flows. It is well understood that, due to the imbalance dataset, overall accuracy is not an enough metric to exhibit the performance of the classifier.

Lastly, we choose the Logistic model classifier for the classification problem at hand. It takes 47.15 s to build the classification model with the Logistic model classifier. This result is far better, if our aim is to get fast classification result. At the end of building the model and testing it, we got 333,099 correctly classified instances which is 99.5725%, and 1430 incorrectly classified instances. Logistic classifier gives very low TP rate which is 0.278. 1234 out of 1708 TrickBot flows are classified as non-TrickBot. Although overall accuracy of classifier is still high, we get a poor TP rate.

Obtained confusion matrix and performance parameter for Logistic model are shown in Tables 8 and 9.

Graphical comparison of tested classifiers is given in Figs. 3 and 4. As could be seen in Fig. 4, Random Forest gives the best performance result in terms of classification accuracy at 99.9408% and Singular Minimal optimization gives 99.4894%, which is less efficient in tested classifiers in terms of correct classification accuracy. Another performance criteria, the true positive rate graphic is plotted, as shown in Fig. 4. These graphics also show that Random Forest gives efficient results for TrickBot detection and for identification of flows.

4.7. *Optimization of Random Forest in Trickbot detection*

Random Forest is an ensemble classification and regression approach, which is unsurpassable in accuracy among current data mining algorithms (Breiman, 2001). As mentioned in the previous section, learning from an imbalanced data set is a big challenge encountered in machine learning and knowledge discovery applications. However, Random Forest exhibited a good performance that has surpassed other learners. It handle the TrickBot dataset very well which has only 0.5% TrickBot flows and provide a relatively better classification result. The bagging (Breiman, 1996) and random subspace mechanism (Ho, 1998) in Random Forest show their strength in decreasing the correlation between each decision tree that helps to minimize the source of error such as bias and variance.

Table 3 – Performance results of Random Forest Classifier.

	TP Rate (TPR)	FP Rate (FPR)	Precision	Recall	F-Measure	Class
	1	0.111	0.999	1	1	**Non TrickBot Flow**
	0.889	0	0.995	0.889	0.939	**TrickBot Flow**
Weighted Average	0.999	0.111	0.999	0.999	0.999	**All**

Table 5 – Performance parameter results of Multilayer Perceptron Classifier.

	TP Rate (TPR)	FP rate (FPR)	Precision	Recall	F-Measure	Class
	1	0.492	0.997	1	0.999	**Non TrickBot**
	0.508	0	0.974	0.508	0.667	**TrickBot**
Weighted Average	0.997	0.49	0.997	0.997	0.997	**All**

Table 6 – Confusion matrix of Sequential Minimal Optimization Classifier.

Non-TrickBot flow	TrickBot flow	Classified as
332,821	0	**Non TrickBot Flow**
1708	0	**TrickBot Flow**

Table 8 – Confusion matrix for Logistic Model Classifier.

Non-TrickBot flow	TrickBot flow	Classified as
332,625	196	**Non TrickBot Flow**
1234	474	**TrickBot Flow**

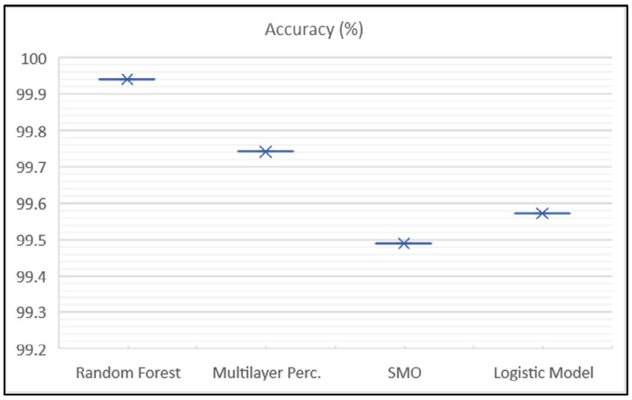

Fig. 3 – Classification accuracy of tested classifiers.

The outputs of randomized unpruned decision trees would produce a robust classification result in terms of outliers and noise. The classification mechanism of Random Forest is illustrated in Fig. 5.

There are two reasons that provide randomness for each decision tree in Random Forest. First, the randomness is achieved by selecting a different bootstrap sample (bootstrapping) from the overall dataset to build each decision tree. So the resemblance between each learner is diminished. After building each decision tree via using different bootstrap samples, the classifier would give a final decision according to the majority voting of all decision tree (bagging) when a new instance is presented to the classifier. The bagging mechanism also provides another advantage which is helpful in evaluating the test performance of the learner in the training step. Approximately, two-third of our dataset is utilized in the training of each tree, and the remaining one-third is used for testing purpose in the training phase. Therefore, out of bag (OOB) cases would give classification accuracy that would prevent overfitting in offline phase.

Another randomness is provided with the arbitrary selection of features in the feature set to be used in splitting each node at a decision tree. Random subspace mechanism would help to produce relatively uncorrelated decision trees which will also optimize out of bag error. The classification accuracy of the RF learner depends on the correlation between individual tree classifiers in the ensemble. In the case of all trees were identical, each tree in the forest would produce same decision that would cause a biased prediction at the end.

One of the challenges in machine learning applications is feature selection procedure. In many fields, feature selection needs a good expertise and domain knowledge. Taking into account irrelevant features would result in a high computational cost, and also increase the error rate of the classifiers. Avoiding overfitting, and reaching a cost effective model could be succeeded with a proper feature selection. However, the advantages of feature selection would introduce another complexity in the modeling task.

Particularly, extracting features from network capture files for the aim of network intrusion detection is a challenging task and needs a high computational cost. Furthermore, taking into account unessential network features not only increase computational cost, but also increase the error rate especially for some algorithms that are sensitive to the number of features. Besides, without feature construction the raw audit .pcap files are not suitable for building NIDSs. Therefore, we aim to decide upon the right set of features which are essential for the classification accuracy and minimization of computational cost.

For the aim of identifying which attributes in the data are the most predictive ones, we benefit from Weka learning tool

Table 7 – Performance results of Sequential Minimal Optimization Classifier.

	TP Rate (TPR)	FP Rate (FPR)	Precision	Recall	F-Measure	Class
	1	1	0.995	1	0.997	**Non TrickBot**
	0	0	0	0	0	**TrickBot**
Weighted Average	0.995	0.995	0.99	0.995	0.992	**All**

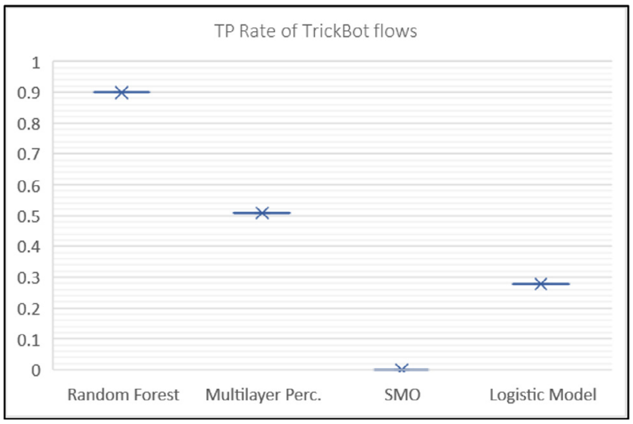

Fig. 4 – Averaged true positive rate of tested classifiers.

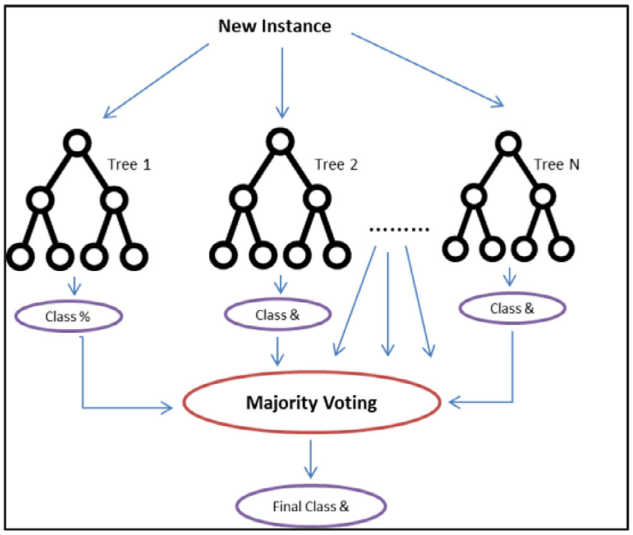

Fig. 5 – Random Forest Classifier.

which has a dedicated panel for the attribute selection. The "Select Attributes" panel includes several search methods and different evaluators for the selection of a feature set. Search methods include best-first, genetic algorithms, linear forward selection, ranking of attributes, and subset size forward selection. Evaluators include correlation, consistency and entropy-based methods. Different search and evaluation methods can be combined to find an optimum feature selection for the problem at hand. Table 10 provides the test results of some Weka's feature search methods with a combined feature evaluator for TrickBot intrusion detection.

According to obtained results, Subset Size Forward selection search method with a combined consistency subset evaluation technique gives the best overall accuracy and best TP rate. Therefore, we use the features determined by this combination to optimize our Random Forest classifier. The selected features with this combination are given in Table 11.

Table 11 – Selected feature list by Subset Size Forward Selection and Consistency Subset Evaluator.

Selected Features
Total forward volume
Total backward volume
Max forward packet length
Max backward packet length
Max backward inter-arrival time
Min time flow was active before idle
Mean time flow was active before idle
Max time flow was idle before active

Table 9 – Performance parameters of Logistic Model Classifier.

	TP Rate (TPR)	FP Rate (FPR)	Precision	Recall	F-Measure	Class
	0.999	0.722	0.996	0.999	0.998	Non TrickBot
	0.278	0.001	0.707	0.278	0.399	TrickBot
Weighted average	0.996	0.719	0.995	0.996	0.995	All

Table 10 – Comparison of search algorithm and feature evaluator for feature selection.

Search Algorithm+ Feature Evaluator	Correct Class. of TrickB. / 1708	TP Rate of TrickB. flows	Overall Accuracy	OOB Error
Without Feature Selection	1518	0.889	99.9408	0.0006
Best First + Consistency subset eval.	1533	0.89754	99.9447	0.0005
Genetic Search+ Consistency subset eval.	1479	0.86592	99.9283	0.0007
Linear forward selection+ Consistencysubseteval	1534	0.89812	99.945	0.0005
Ranker+ Chi Squared Attribute Eval	1526	0.89344	99.9435	0.0005
Ranker+ Info Gain Attribute Eval.	1535	0.89871	99.9465	0.0005
Ranker+ OneRAttributeEval	1517	0.88817	99.9399	0.0006
Subset size forward selection + Consistency Subset Eval.	1567	0.91744	99.9534	0.0005

Table 12 – Selected feature list by subset size Forward Selection and Consistency Subset Evaluator.

Tree Number (K)	Feature number (M)	Seed (S)	Tree depth (D)	TP Rate	Accuracy (%)	Time (Sec.)
10	4	1	0	0.903	99.9462	26.59
50	4	1	0	0.915	99.9519	119.49
100	4	1	0	0.917	99.9534	230.81
200	4	1	0	0.915	99.9519	483.85
300	4	1	0	0.914	99.9517	701.16
100	3	1	0	0.909	99.9495	186.52
100	5	1	0	0.916	99.9519	292.81
100	6	1	6	0.692	99.8359	328.87
100	6	1	10	0.816	99.902	387.86
100	6	1	15	0.912	99.9495	379.37
100	8	1	0	0.915	99.948	502.74
100	4	2	0	0.914	99.9513	238.72
100	4	3	0	0.913	99.9516	234.81
200	6	2	0	0.915	99.951	756.8
300	6	2	0	0.914	99.951	1198.92
300	6	3	0	0.917	99.9525	1174.78
300	8	3	0	0.912	99.9447	1504.77

Lastly, we will focus on optimizing Random Forest hyper parameters for TrickBot intrusion detection, namely the number of decision trees (K), randomly selected features (M), the seed (S), and depth parameter (D) of each tree. Optimizing the number of decision trees helps to minimize variance and finally decrease the tendency to any overfitting. However, increasing the number of classifiers would increase model built time and classification time. We observed that, 100 classification tree number is an optimum choice for our TrickBot classifier.

Another significant parameter is arbitrarily selected features at each tree for splitting nodes. This parameter arranges the randomness of each tree in the forest that helps to increase robustness and strength in the forest. According to Breiman's work, [$\log_2$ (Feature number)+1] attributes in building each tree in the ensemble is a good recommendation (Breiman, 2001). Obtained results prove this extraction. When we chose 4 as M parameter value, the best TP rate and classification accuracy is reached. Tree depth parameter is used to determine the depth of each tree. 0 value for depth parameter means an unlimited tree depth. It is observed that unlimited tree depth selection provides better results compared to other choices. Seed is a random number seed to be used to guarantee to produce same model when entered same value. 1 is an optimum choice for Seed parameter to get a better accuracy. The best hyper parameter set combination is given in the third row of Table 12 which is highlighted with yellow. We obtained a better overall accuracy and TP rate with this hyper parameter values.

4.8. *Conclusion and future work*

The TrickBot banking trojan family has threatened businesses in the financial sector and online banking customers for more than two years. TrickBot has been updated continually during this time by releasing new versions almost every day. With new modifications, it is getting more dangerous in its new infection strategy and attack vectors. We built a classifier which detects TrickBot infection via identifying TrickBot related flows. We compared the results of 4 different classifiers for the classification problem at hand. According to the obtained results, the Random Forest classifier gives the best classification results among four classifiers. We optimize the feature set via utilizing the Subset Size Forward Selection and Consistency Subset Evaluator to enhance the accuracy of Random Forest classifier. We also perform tests to identify the optimum hyper parameter set for TrickBot flow detection which is 100 classification trees, 4 random features for splitting each tree, 0 for Depth and 1 for Seed parameter. According to the results obtained with optimized feature set and hyper parameter set, we have reached 99.9534% classification accuracy with a 91.7% true positive rate. With our built model, TrickBot related flows and infection can easily be identified on networks.

In our future studies, we will focus on some up-to-date banking trojans such as Gozi (Ursnif), a prevalent trojan in 2018, and a Zeus variant, Game over Zeus, which has infected over 500.000 PCs worldwide, to detect infections on the network and to extract signatures related with these trojans. We will also continue to analyze TrickBot to detect any updated infection strategy and attack vectors with newer versions.

Acknowledgment

The authors would like to thank the editors and anonymous reviewers for their valuable suggestions and comments. The authors would like to thank Dr. Murat M. Tanik for his support, encouragement and guidance. I appreciate Ines Rozsa Villarrel for her review.

I would like to thank TUBITAK-Directorate of Science Fellowships and Grant Programmes (BIDEB) to support my postdoctoral research project coded with 1059B191600549 application number.

Conflict of Interest

The authors confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cose.2019.03.013.

REFERENCES

Alshammari R, Zincir-Heywood AN. A flow-based approach for SSH traffic detection. In: Proceedings of the IEEE international conference on systems, man and cybernetics. ISIC; 2007. p. 296–301.

Black P, Gondal I, Layton R. A survey of similarities in banking malware behaviours. Comput Secur 2018;77:756–72.

Breiman L. Bagging predictors. Mach Learn 1996;26(2):123–40.

Breiman L. Random Forests. Mach Learn 2001;45(1):5–32.

Carminti M. Security evaluation of a banking fraud analysis system. ACM Trans Priv Secur 2018;21(3):11.

Criscione C, Bosatelli F, Zanero S, Maggi F. ZARATHUSTRA: extracting Web inject signatures from banking Trojans. In: Proceedings of the 12th annual international conference on privacy, security and trust; 2014. p. 139–48 July 23–24.

Haddadi F, Zincir-Heywood AN. Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification. IEEE Syst J 2016;10(4):1390–401.

Ho TK. The random subspace method for constructing decision forests. IEEE Trans Pattern Anal Mach Intell 1998;20(8):832–44.

IETF, http://www3.ietf.org/proceedings/97apr/97apr-final/xrtftr70.htm

Keshet L. An aggressive lauch: TrickBot Trojan rises with redirection attacks in the UK. Secur Intell November 2016. available: https://securityintelligence.com/an-aggressive-launch-TrickBot-trojan-rises-with-redirection-attacks-in-the-uk/.

Kessem L, Trojan Widens Its attack scope in Spain, Brings redirection attacks to local banks, July 19, 2017, Available: https://securityintelligence.com/TrickBot-habla-espanol-trojan-widens-its-attack-scope-in-spain-brings-redirection-attacks-to-local-banks/, Accessed 07 December 2018.

Khattak S, Ramay NR, Khan KR, Syed AA, Khayam SA. A taxonomy of botnet behavior, detection, and defense. IEEE Commun Surv Tutor 2014;16(2):898–924.

Khoshgoftaa TM, Golawala M, Hulse JV. An empirical study of learning from imbalanced data using random forest. Proceedings of the 19th IEEE international conference on tools with artificial intelligence, 2007.

Kiwia D, Dehghantanha A, Choo KR, Slaughter J. A cyber kill chain based taxonomy of banking Trojans for evolutionary computational intelligence. J Comput Sci 2018;27:394–409.

Kremez V, Burbage P, With a boost from Necurs, TrickBot expands its targeting to numerous U.S. financial instutions, July 2017, Available: https://www.flashpoint-intel.com/blog/TrickBot-targets-us-financials/, Accessed 07 December 2018.

Lopez PP, Martin H. Hardware trojans against virtual keyboards on e-banking platforms- a proof of concept. AEU-Int J Electron Commun 2017;76:146–51.

Malwarebytes Labs, TrickBot-Dyreza's Successor, October 24, 2016, Available: https://blog.malwarebytes.com/threat-analysis/2016/10/trick-bot-dyrezas-successor/, Accessed 07 December 2018.

Milletary J. Citadel trojan malware analysis. Dell SecureWorks Threat Unit Intelligence Services; 2012. Available https://blog.barkly.com/top-banking-trojans-2017. [Accessed 23 June 2018].

Mohaisen A, Alrawi O. Unveiling Zeus Automated Classification of Malware Samples. In: Proceedings of the 22nd international conference on world wide web ACM; 2013. p. 829–32 May 13–17.

Netmate, Available: https://github.com/DanielArndt/netmate-flowcalc, Accessed 29 May 2018.

Ono K, Kawaishi I, Kamon T. Trend of Botnet Activities. In: Proccedings of the 41st annual IEEE international Carnahan conference on Security Technology, Canada; 2007. p. 243–9.

Pauli D, Dyre malware re-surfaces as TrickBot, targets Australian bank, October 2016, Available: https://www.theregister.co.uk/2016/10/18/one_of_the_worlds_worst_trojans_feared_back_and_targeting_aussie_banks/ Accessed 07 December 2018.

Peotta L. A formal classification of Internet banking attacks and vulnerabilities. Int J Comput Sci Inf Technol 2011;3(1):186–97.

Riccardi M, Oro D, Jana J, Cremonini M, Vilanova M. A framework for financial botnet analysis. In: Proceedings of the ECrime researchers summit (ECrime); 2010. p. 1–7.

Soniya B, Wilscy M. Detection of randomized bot command and control traffic on an end-point host. Alex Eng J 2016;55(3):2771–81.

Szopinski TS. Factors affecting the adoption of online banking in Poland. J Bus Res 2016;69(11):4763–8.

Top 10 Banking trojans for 2017: What you need to know, Jonathan Crove, April 2017, Available: http://www.secureworks.com/cyber-threatintelligence/threats/top-banking-botnets-of-2013/ Accessed 07 December 2018.

Trick Bot config files, Available: https://github.com/JR0driguezB/malware_configs/tree/master/TrickBot, Accessed 07 December 2018.

TrickBot-Toolkit, Available: https://github.com/MalwareTech/TrickBot-Toolkit, Accessed 29 May 2018.

Watkins L, Kawka C, Corbett C, Robinson WH. Fighting banking botnets by exploiting inherent command and control vulnerabilities. In: Proceedings of the 9th international conference on malicious and unwanted software: the Americas (MALWARE); 2014. p. 93–100.

WEKA, Available: http://www.cs.waikato.ac.nz/ml/weka/, Accessed 05 May 2018.

Williams N, Zander S, Armitage G. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. ACM SIGCOMM Comput Commun Rev 2006;36(5):7–15.

Zhang X, Deep analysis of the online banking botnet TrickBot, December 2016, Available: https://www.fortinet.com/blog/threat-research/deep-analysis-of-the-online-banking-botnet-TrickBot.html, Accessed 07 December 2018.

Zhao D, Traore I, Sayed B, Lu W, Saad S, Ghorbani A, et al. Botnet detection based on traffic behavior analysis. Comput Secur 2013;39:2–16 Part A.

A. GEZER was born in Kayseri City, Turkey, in 1976. He received the B.S. degree in Electronic and Computer Education from Marmara University in 1999 and M.S. degree in computer engineering from Erciyes University in 2004, and the Ph.D. degree in electronic engineering from Erciyes University, Kayseri, TURKEY, in 2011.

He is an assistant professor with the Electronic and Communication Technology in Erciyes University. His research interests include internet traffic analysis, self-similarity, network traffic analysis and characterization, signal processing techniques, telecommunication technologies, IoT botnet investigations, malware analysis.

G. WARNER was born in Indiana and grew up in the Mid-West. He moved to Birmingham, Alabama to attend UAB, where he earned his Bachelor's in Computer Science. Warner has worked in mainframe operations, network security and design, and as the I.T. Director for an oil and gas company. He started the Birmingham InfraGard chapter in 2001, and has served on the national board of directors for both the FBI InfraGard program and the DHS Energy ISAC.

In 2007, he joined the University of Alabama at Birmingham to train future cyber-crime investigators. He currently directs a staff of 50 student researchers in the UAB Computer Forensics Research Lab where he works primarily on malware and botnet investigations, cyber-crime investigations, and the social media usage of criminals, hate groups, and terrorists.

C. WILSON was born outside of San Jose, California, and resides in Birmingham, Alabama. He is currently pursuing a Bachelor of Science degree in Digital Forensics at the University of Alabama at Birmingham (UAB). Wilson has been working as a Malware Analyst in the UAB Computer Forensics Research Lab, where he leads the Malware Research team and a Content Research team for a major social media company.

As a Malware Analyst, he has focused his research on the TrickBot banking trojan botnet. Upon graduation, he will be pursuing a career in Cyber Forensics with a specialty in Open Source Intelligence.

P. SHRESTHA was born in Nepal in 1987. He received the bachelor's degree in Computer Engineering from Kantipur Engineering College, Tribhuvan University, Nepal in 2011. He is currently persuading the PhD degree in computer science under the supervision of Dr. Nitesh Saxena. He is a member of the Security and Privacy In Emerging computing and networking Systems (SPIES) Research Laboratory, the University of Alabama at Birmingham (UAB). His research interests include exploring new authentication, particularly low-effort Two-Factor Authentication (TFA) and Zero-Effort Deauthentication (ZED), and privacy paradigms, from both offensive and defensive perspective, in/using mobile and wearable computing.