Salting Public Traces with Attack Traffic to Test Flow Classifiers

Z. Berkay Celik[1], Jayaram Raghuram[2], George Kesidis[1,2], and David J. Miller[2]

[1]Department of Computer Science and Engineering
[2]Department of Electrical Engineering
Pennsylvania State University
University Park, PA, 16802
{*zbc102,jzr148,gik2,djm25*}*@psu.edu*

Abstract

We consider the problem of using flow-level data for detection of botnet command and control (C&C) activity. We find that current approaches do not consider timing-based calibration of the C&C traffic traces prior to using this traffic to salt a background traffic trace. Thus, timing-based features of the C&C traffic may be artificially distinctive, potentially leading to (unrealistically) optimistic flow classification results. In this paper, we show that round-trip times (RTT) of the C&C traffic are significantly smaller than that of the background traffic. We present a method to calibrate the timing-based features of the simulated botnet traffic by estimating eligible RTT samples from the background traffic. We then salt C&C traffic, and design flow classifiers under four scenarios: with and without calibrating timing-based features of C&C traffic, without using timing-based features, and calibrating C&C traffic only in the test set. In the flow classifier, we strive to use features that are not readily susceptible to obfuscation or tampering such as port numbers or protocol-specific information in the payload header. We discuss the results for several supervised classifiers, evaluating botnet C&C traffic precision, recall, and overall classification accuracy. Our experiments reveal to what extent the presence of timing artifacts in botnet traces leads to changes in classifier results.

1 Introduction

The principal network-based methods of detecting botnets either rely on passive DNS analysis (*e.g.*, [7] targeting fast flux activity) used to hide the identities of botmaster servers, or rely on deep packet inspection targeting unencrypted malware in-transit (the latter often part of a more comprehensive NIDS system, *e.g.*, [9]). Intercepted malware may be quickly inspected to infer behavioral signatures of the botnet. Netflow information has also been exploited to detect botnets (*e.g.*, [10, 8, 16]). Such analysis can lead to rankings or a reputation system for Internet domains (or individual IP addresses). These detection methods are imperfect, *e.g.*, they may suffer from false positive detections with coordinated software distribution/update systems. So recently, analysts have proposed jointly using netflow and DNS analytical methods (*e.g.*, [27]).

Detection based purely on netflows, particularly using timing information, is the focus of this paper. However, we consider neither any specific behavioral signatures that may be given nor use timing information to detect anomalies that may indicate communication by stepping stones. Rather, our objective is to study detection of botnet C&C activity by flow-based classification and clustering, as in [8].

Flow-based classification and clustering methods examine traffic statistics of packet flows to identify attacks and malicious behavior in the network [26]. Recently, Speratta *et al.* [28] presented an extensive survey of state of the art flow-based IDS with attack classifications and proposed solutions. Among these attack classifications, a few botnet detection techniques were adapted using machine learning (ML) algorithms. These techniques extensively perform botnet detection using timing-based flow statistics. [19], [30] and [29] extract duration, average packet rate, and bit rate per second characteristics of flows for performing supervised classification. A recent approach [8] similarly uses the number of flows per hour and the average bytes per second for performing unsupervised clustering. These techniques use representative attack traffic for both training and testing their classifiers to provide confidence in their proposed solutions. However, there are not many realistic datasets publicly available with background and attack traffic naturally present together. For these reasons, researchers often generate synthetic malicious traffic to salt real enterprise packet traces for evaluating the effectiveness of the proposed classifiers. As an example, [19] uses a testbed to obtain

This material is based upon work supported by the National Science Foundation under Grant No. 0915552 and a Cisco Systems URP gift.

the botnet traces, and [29, 30] use more representative botnet traces by hosting bots inside the internal network and a C&C center on an external network. The BotMiner system [8] salts two IRC and two HTTP botnet traces, which were captured by running benign versions of bots in a simulated environment to the background traffic.

In this paper, we explore how the timing-based features determined during the salting process may affect classification performance. Timing-based features may vary depending on the position of the packet-trace monitor and, of course, on the geographical separation of communicating points including bots and their masters (and intermediate stepping stones if any). In both situations, "directly" salting botnet traces into the background traffic may introduce timing inconsistencies with the background traffic which may affect classification performance. Hence, it is necessary to achieve timing consistency between the traces to mitigate specific artifacts of timing-based features.

Our objective is, before salting simulated botnet traffic traces into background traffic traces, to provide consistency between the timing-based features of the two traces. We analyze the factors that contribute to the timing-based features (particularly goodput *cf.,* Section 3.1) of flow classifiers. After simulating a benign version of the Kaiten bot to acquire botnet C&C traces, we estimate eligible RTT samples from the background traffic. We calibrate the simulated botnet traffic timestamps to provide consistency of timing-based features between the traces. Then, background traffic is salted to the simulated botnet traces, and flow features are extracted. We perform experiments using supervised ML algorithms to distinguish the botnet C&C traffic from the background traffic, and discuss the classification results both with and without calibrating timing-based features of botnet C&C traffic.

The remainder of this paper is organized as follows: In Section 2, we begin by describing the LBNL public packet trace, flow generation procedure, flow features we extracted, the way ground-truth flow-class labels were derived, and how the Kaiten bot was simulated. In Section 3, RTT calibration and botnet trace timestamp modification is described. In Section 4, we present detailed analysis of classification results. Finally, we summarize in Section 5.

2 Background

2.1 Packet Trace Data

In our experiments, we use publicly available Lawrence Berkeley National Laboratory (LBNL or now just LBL) packet traces [4]. According to [23], the LBNL trace represents internal enterprise traffic recorded at a medium-sized site. The measurement system at LBNL simultaneously monitored the traffic out of two of the (more than twenty) router ports and collected packet traces that spanned more than 100 hours of activity from a total of several thousand internal hosts. Thus, the packet traces are from a successive sampling of subnets. All together, 11GB of packet header traces from October 2004 through January 2005 are available for analysis. This data was publicly released and anonymized by the system described in [24, 23]. The authors used a prefix-preserving scheme to remap the external and internal IP addresses. Additionally, the subnet and host portions of the internal addresses were further transformed in a one-to-one fashion. However, the port numbers in the TCP headers are intact, which provides information about application type at the time the trace was recorded. The authors also provided meta-data allowing us to obtain anonymized subnets and gateway addresses which are used for RTT estimation in our experiments.

The raw trace used in our experiments is a combination of those recorded on Oct. 4, 2004, router port 19, and on Dec. 15, 2004, router port 8[1].

2.2 Flow Generation and Flow-level Features

A typical flow exporter system takes flow metrics (protocol type, number of packets and flow timeout) as an input and automatically starts processing the PCAP files. First, the system analyzes bi-directional session between endpoints depending on their protocol type, then it automatically collects the set of packets between the two endpoints which have a set of common properties: Source IP, destination IP, source port, destination port and protocol (IP 5-tuple) [25]. Since the simulated Kaiten Bot uses TCP-based attacks, we restrict our attention to TCP flows. Following [18], we extracted features from first 5 packets after the 3 way handshake with 60 seconds inactive timeout value[2] to prevent significant processing load of capturing and exporting per flow information at the monitoring device. In addition, the first 5 packets provide feasibility of detection in early stage of a connection [6].

In [22], the authors describe 248 flow-level features. Among these features, we focus on the eight features: cnt-data-pkt, min-data-size, avg-data-size, init-win-bytes, RTT-samples, IP-bytes-med and frame-bytes-var (see Table 1 for definitions). According to [18], these features were selected from among 248 features after applying a correlation-based filtering mechanism on each of the datasets. We also identified the following two "derived" features as promising ones not strongly dependent on TCP:

- *IP-ratio:* The ratio of the maximum to the minimum IP packet size (client to server and server to client),
- *Goodput:* Total non-duplicate number of frame bytes divided by total flow time (client to server and server to client).

Using these twelve features, we extracted a total 7266

Abbreviation	Description
cnt-data-pkt	The count of all the packets with at least a byte of TCP data payload (client to server)
min-data-size	The minimum payload size observed (client to server)
avg-data-size	Data bytes divided by number of packets (server to client)
init-win-bytes	The total number of bytes sent in initial window(server to client & client to server), see [22]
RTT-samples	The total number of round-trip time (RTT) samples found (client to server), see [22]
IP-bytes-med	Median of total bytes in IP packet (client to server)
frame-bytes-var	Variance of bytes in (Ethernet) packet (server to client)
IP-ratio	Ratio between the maximum packet size and minimum packet size (server to client & client to server)
goodput	Total number of frame bytes divided by the flow duration (server to client & client to server)

Table 1: Flow feature definitions and descriptions

LBNL TCP flows. Note that the twelve features we extract do not consider fields that can be readily tampered with, such as *port numbers* or protocol-specific information in the payload header, *e.g.*, count of packets with the push bit set in the TCP option field [19]. Therefore, a classifier built based on these features should tend to be more "tamper-resistent" than one based on highly TCP-dependent features [34].

2.3 Ground Truth Labels

For supervised classifier training, it is necessary to ascertain ground-truth class labels for both training and test data. However, LBNL traces have only the anonymized raw packets publicly available, containing only the TCP/IP header information without any payload. Fortunately, port numbers were unaffected by the anonymization process, with the exception of traffic associated with one particular port used as an internal security monitoring application. We extracted the client and server port numbers for each flow, and then applied the Internet Assigned Numbers Authority (IANA)'s list of registered ports [3] to determine the application. We observed that the LBNL trace includes a large variety application types with different communication patterns. As an example, web, e-mail and name services include both enterprise network and wide-area network traffic. Windows services and network file services are the only within enterprise applications. We labeled every flow in the dataset with the corresponding application category, following the same procedure as [21]. While port-based application identification might seem ineffective, the use of non-standard or dynamic port numbers is unusual enough in the LBNL enterprise traffic that it does not affect our label definitions significantly [21]. The application category breakdown of TCP traffic is shown in Table 2.

Class	Protocols
bulk	FTP, HPSS
email	SMTP, IMAP4, IMAP/S,POP3, POP/S, LDAP
interactive	SSH, telnet, rlogin, X11
name-srv	DNS, netbios-NS, SrvLoc
net-file	NFS, NCP
net-mgnt	DHCP, ident, NTP, SNMP, NAV-ping, SAP, NetInfo-local
web	HTTP, HTTPS
windows	CIFS/SMB,DCE/RPC, Netbios-SSN, Netbios-DGM
misc	Steltor, MetaSys, LPD, IPP, ORACLE-SQL, MS-SQL

Table 2: Flow class definitions and sample applications

2.4 Botnet Simulation

In order to obtain traces of actual botnet C&C traffic, an IRC-based bot Kaiten whose source code was available on the web [1] was simulated. A virtual network of one bot controller, three infected machines (bots) and a bot controller was implemented by reverse engineering the bot code. We collected C&C flows associated with our botnet using wireshark [5] at the simulated IRC server. The complete simulation is described in the technical report [33].

Bots which connect to the IRC server using a predetermined C&C channel, and send a NICK IRC message to convey that the user is online with a certain ID. The controller then replies to verify that the bot is alive and has joined the C&C channel. After the verification phase, the bots wait for commands from the controller. Upon the arrival of a command, the bots act accordingly. If no command is received for 20 minutes, a bot reopens a connection to the controller. Our goal here is to generate C&C heartbeat traffic of an already established botnet that is *readily* standing for commands. Our goal is not intrusion prevention or detecting of bots after an overt attack *e.g.*, denial-of-service (DOS) or email spam has been launched. Instead, we wish to simulate and detect a botnet that is either in a sleep or "stealth" state where, as an example of the latter, the botnet could be slowly/discreetly exfiltrating private data.

3 Methodology

3.1 RTT Calibration

In our flow classifier implementation, we used a timing-based goodput feature in both directions (client to server and server to client) defined as the total non-duplicate number of frame bytes divided by the total flow duration. From Figure 2, we split bi-directional flows into two unidirectional flows which contain k nonduplicate packet series, $P_{cs} = (P_i, P_{i+1}, \ldots, P_k)$, with a corresponding timestamp, t_i, of the i^{th} packet at the monitoring point. In our experiments, we calculate the goodput feature after the 3 way handshake. Therefore, we do not include RTT

samples of the SYN-SYNACK segments and the corresponding SYNACK-ACK segments. We can calculate the goodput feature from client to server as follows (the goodput feature from server to client can be computed in a similar way):

$$Goodput_{cs} = \frac{\sum_{i=1}^{k} frame\ bytes_{cs}}{flow\ duration} \quad (1)$$

Equation (1) depends on the flow duration which includes the RTT and inter-packet arrival time of the connections. RTT is the difference between the capture time of the sending packet with a certain sequence number (SEQ) and the corresponding follow up acknowledgement (ACK) from the receiver. To avoid the ambiguity of computing the RTT, retransmitted packets are omitted in our goodput calculation. The packet inter-arrival time is the difference between the time of the i^{th} packet and the $(i-1)^{th}$ packet at the monitoring point. Ideally, it is characterized by the network, application process time and the size of the congestion window. In addition, for specific applications, it includes "thinking" or "reading" time of the users. In this paper, we only deal with the differences of the RTT samples between botnet and background traffic[3].

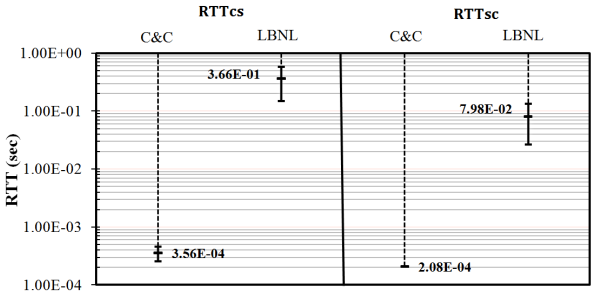

Figure 1: Comparison of background traffic and simulated C&C RTTs before calibration

Figure 1 shows a logarithmic scale plot of mean and standard error of the mean[4] for botnet and background traffic RTT samples before calibrating the simulated botnet C&C traffic. There is a significant difference between the mean values of C&C traffic and background traffic $RTTs$. This explains that goodput features of the botnet traces are likely to change significantly after C&C RTT calibration. RTT samples of the C&C traffic have less within-group difference, since network conditions do not have any effect on the virtual simulated environment used to generate C&C traffic. Additionally, LBNL traffic RTT_{sc} values are considerably smaller than that of RTT_{cs} values, because the position of the LBNL packet-trace monitor is close to the LBNL clients. Now, we are interested in estimating eligible

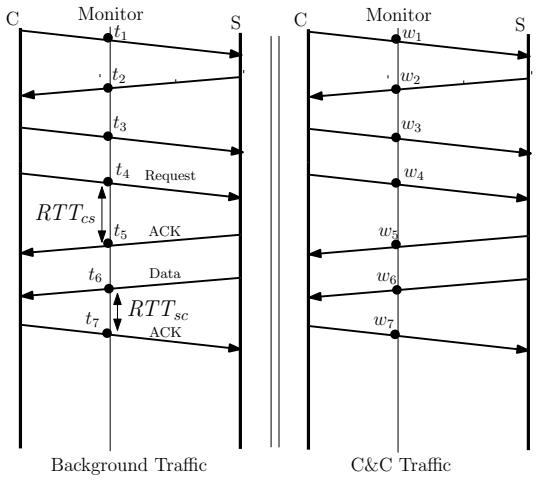

Figure 2: Timestamp modification procedure

RTTs from background traffic to calibrate the botnet traffic t_i values before salting. There are various methods suggested by the research community to estimate passive RTT values from a packet trace. In [15], the authors proposed SYN-ACK (SA) and Slow-Start (SS) estimation techniques for unidirectional flows. In [31], two different techniques are offered, both of which depend on the TCP timestamp option by adding timestamp value (TSval) and timestamp echo reply (TSecr) values [13] to the TCP header option; however, these methods require a *priori* deployment, and implementation of timestamp increment should be consistent across different hosts. Other techniques [20, 14] generally depend on more complex deduction methods based on the sender's congestion window size, or maximum-likelihood estimation for matching data segment with ACK segment. In this study, we measure the RTT samples using provided timestamp values recorded at the monitoring point. We will use the following procedure to estimate the RTT samples from background traffic:

- We take the IP addresses which reside inside the enterprise as client and the external IP addresses as server. We simulate the external destination IP addresses as a C&C center and IP addresses inside the enterprise as bots.
- We construct RTT vectors of LBNL background traffic $V_{RTT} =< RTT_1, \ldots, RTT_n >$ for both directions corresponding to the eligible subset of the RTTs used in flow feature goodput generation.
- We calculate the empirical cumulative distribution function (CDF) from the RTT samples of the background traffic and use it to randomly generate the RTT_i values for the simulated botnet C&C traces.

After RTT estimation, we calibrate 50 conversations between the bots and the C&C Channel traffic by modifying their timestamp values. From Figure 2, we divide bi-directional packet flows into two unidirectional flows, P_{cs} and P_{sc}, which contain k nonduplicate packet series of C&C traffic. The initial timestamps of the C&C packets after 3 way handshake, w_3, stays same and the remaining timestamps, $w_4, \ldots, w_k$, of the packets are only calibrated when the ACK of sending packet is received. We calibrate the RTT samples (RTT_{cs} and RTT_{sc}) of the C&C traffic from the corresponding V_{RTTcs} *and* V_{RTTsc} vectors of the LBNL background traffic. As an example the timestamp of the first acknowledgement packet of the C&C traffic, w_5, is calibrated by calculating $w_4 + RTT_{cs}$, where RTT_{cs} is estimated from the LBNL background traffic. The calibration algorithm can be summarized as follows:

Algorithm 1 RTT Calibration

$k \leftarrow number\ of\ packets$
$i \leftarrow 1$
repeat
  if $(P_i^{sc}$ *and* $P_{i+1}^{sc})||(P_i^{cs}$ *and* $P_{i+1}^{cs})$ **then**
    $w_{i+1}^{new} = w_i + (w_{i+1}^{old} - w_i)$
  else if $(ACK^{sc} \in P^{cs})$ **then**
    Randomly select $RTT_{cs},\ RTT_{cs} \in V_{RTTcs}$
    $w_{i+1} = w_i + RTT_{cs}$
  else if $(ACK^{cs} \in P^{sc})$ **then**
    Randomly select $RTT_{sc},\ RTT_{sc} \in V_{RTTsc}$
    $w_{i+1} = w_i + RTT_{sc}$
  end if
  $i \leftarrow i+1$
until $i = k$
$saltToBackgroundTraffic()$

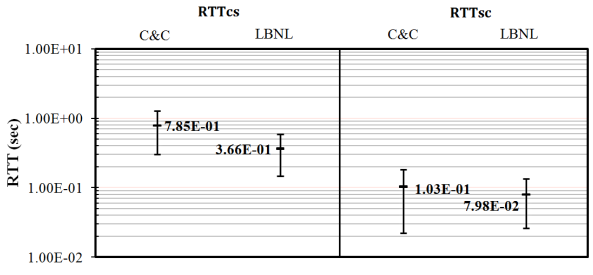

Figure 3: Comparison of background traffic and simulated C&C RTTs after calibration

As a result of application of Algorithm 1 to the RTTs of the botnet C&C traffic, there is better agreement between RTTs of simulated botnet traffic and background traffic as shown in Figure 3.

4 Experimental Results

4.1 ML Algorithms

We now evaluate how sensitive the supervised classification results are to timing-based features ($goodput_{cs}$ and $goodput_{sc}$). We use the true positive (*TP*) rate to show the percentage of flows correctly classified as C&C traffic, the false positive (*FP*) rate to represent the percentage of legitimate applications flagged as C&C traffic, and false negative (*FN*) rate to represent the percentage of C&C traffic flagged as other legitimate applications. We calculate the overall accuracy of the classifier, and recall and precision rate of the C&C traffic.

- *Recall*, which is the percentage of flows in an application class or botnet C&C traffic class, *i*, that are correctly identified,is defined as $TP_i/(TP_i + FN_i)$.
- *Precision*, which is the percentage of flows assigned to the each application class or botnet C&C traffic class, *i*, that are correctly, is defined as $TP_i/(TP_i + FP_i)$.
- *Overall accuracy*, which is the ratio of correctly classified flows to the total number flows in dataset, defined as $\sum_{i=1}^{n} TP_i \ / \ \sum_{i=1}^{n}(TP_i + FP_i + FN_i + TN_i)$ where *n* is the number of classes.

In our experiments, to perform supervised learning we used a Naive Bayes classifier, decision tree using the C4.5 algorithm, multinomial logistic regression, and logistic model trees (LMT). We give a brief description of these methods and our experimental settings. A Naive Bayes classifier assumes that the features are conditionally independent given the class of origin, and learns a model for the class conditional probability distribution of each feature. The Bayes rule is then used to compute the class posterior probability given the features, which is used for classification. The C4.5 algorithm creates a decision tree, where at each node of the tree the feature(s) with largest information gain is used to split the data into sub-groups, ending at the leaf nodes. A decision tree should have the property that at each leaf node, a strong majority of the samples belong to one class, which is also chosen as the predicted class for samples belonging to that leaf node. In multinomial logistic regression, we have a linear discriminant function associated with each class, and the log ratio of the probability of each class given the features, to the probability of a reference class given the features is modeled as a linear function of the features. The weights of the linear discriminant function associated with each class are determined using a maximum a *posteriori* (MAP) estimation. A LMT classifier learns a decision tree using the C4.5 algorithm, but the class prediction at each leaf node is done with a logistic regression model, unlike the standard decision tree, which uses voting [17]. The classifiers described above

	LMT				C4.5				Logistic Regression				Naive Bayes			
	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
web	4	2	-	4	-	-	-	-	3	2	2	3	-	-	-	-
windows	-	-	-	-	2	2	2	2	3	3	2	3	-	-	-	-
email	1	3	1	1	-	-	-	-	13	13	13	13	-	-	-	-
name-srv	-	2	-	-	-	-	-	-	7	1	5	7	-	-	-	-

Table 3: Comparison of the classiffier C&C false positive counts

	LMT				C4.5				Logistic Regression				Naive Bayes			
	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
web	-	-	-	-	-	-	-	-	1	4	2	3	-	-	-	-
email	-	-	-	-	-	-	-	-	-	-	-	-	1	1	-	1
name-srv	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 4: Comparison of the classiffier C&C false negative counts
(1) C&C traffic timing-based features are not calibrated (2) C&C traffic timing-based features are calibrated
(3) classification does not use timing-based features (4) timing-based features of the C&C traffic are not calibrated in the training set, but are calibrated in the test set

are suitable for multiclass problems and are commonly applied to machine learning problems.

4.2 Classifier Evaluation

We investigated flow classification to one of ten classes, including the nine classes defined in Table 2 and the C&C class, *i.e.*, the C&C class was treated as a predefined class, with the classifier trained in a supervised fashion to assign flows to one of these ten classes, based on labeled training examples from all classes (including the C&C class). We salt a total of 7266 LBNL TCP flows of 9 classes described in Table 2 with the 50 botnet C&C traffic class under the following four scenarios: (1) when the timing-based features of the botnet C&C traffic are not calibrated with the background traffic in both training and test set; (2) when they are calibrated with the background traffic by the application of Algorithm 1 in both training and test set; (3) when the timing-based features are not used for classification, and (4) when the timing-based features of the C&C traffic are not calibrated in the training set, but are calibrated in the test set.

Another motivation for scenario (2) is that the attacker may deliberately arrange the timing-based features to conceal the C&C activity. Scenario (4) represents a realistic case where C&C traffic in the training set is provided by the simulated environment and tested with calibrated botnet traffic, whose timing-based features are consistent with the background traffic. This scenario begs other scenarios where the C&C traffic is not calibrated to the background traffic in the training set, and is calibrated differently in the test set in two ways: different from the background traffic or different from the C&C traffic in the training set.

Algorithm	Dataset	C&C Precision (%)	C&C Recall (%)	Overall Accuracy (%)
LMT	(1)	93.89	100	98.442
	(2)	89.99	98	98.057
	(3)	98.33	100	99.048
	(4)	93.89	100	98.442
C4.5	(1)	96.66	100	99.317
	(2)	96.66	100	99.317
	(3)	96.66	100	99.289
	(4)	96.66	100	99.317
Logistic Regression	(1)	67.17	98	95.381
	(2)	73.4	92	95.434
	(3)	70.63	96	95.422
	(4)	67.17	94	95.312
Naive Bayes	(1)	100	98	90.405
	(2)	100	98	90.405
	(3)	100	100	91.655
	(4)	100	98	90.405

Table 5: Comparison of the classiffier C&C precision, recall and overall accuracy rates

The implementation of the classifiers was obtained from the open source data mining tool Weka [11]. We use stratified 10-fold cross-validation by dividing the dataset into 10 folds of approximately equal size. The proportion of classes is roughly the same in all 10 folds. We fit the model on 90% of the training data and then predict the class labels of the remaining 10% of the test data. This procedure is repeated 10 times, with each fold in turn playing the role of the test samples, and the errors

on all 10 folds averaged together to compute the precision, recall and overall accuracy. Note that in creating the cross validation folds for the 4 scenarios, we use the same training and test folds, and modify only the timing-based features of the C&C traffic, where needed (*e.g.*, in scenario (4) only the timing-based features of the C&C traffic in the test folds are modified).

Table 5 shows the precision and recall rates of the C&C flows, and the average 10-fold cross-validation accuracy of each of the four classifiers. For LMT and Naive Bayes, when timing-based features are not used for classification, the C&C precision, recall rates, and the overall accuracy all increase. For the C4.5 classifier, the results are the same in all 4 scenarios. A possible explanation is that the timing-based features are not used by the C4.5 decision tree, or the rules based on the timing-based features are not sensitive to the feature calibration[5]. For the Naive Bayes classifier, it is interesting to note that even though the classification accuracy is not as high as the other classifiers, the C&C precision and recall rates are almost perfect. Again, the Naive Bayes classifier models the class conditional distribution of all the features and uses a Bayes rule to make class predictions. In this case, it is possible that we have an accurate model for the C&C class, which results in high precision and recall, but the model for other application classes may not be very accurate which results in the relatively low overall accuracy. For logistic regression and LMT, the precision and recall rates are different in all the scenarios. A possible explanation is that the class posterior probability depends directly on a linear combination of all the feature values, making the predictions sensitive to the timing feature calibration.

In Table 3, the number of false positives (*i.e.*, the number of legitimate applications that are classified as C&C traffic) are shown for all the scenarios. Note that for scenarios (1) and (4), since the training and test sets are the same for all application classes, the false positive counts are identical. For LMT and logistic regression, it is interesting to observe the changes in false positive counts in scenarios (1), (2), and (3). As an example, for LMT, the change from 4 to 2 for the web class, and from 1 to 3 for the email class; for logistic regression, the change from 3 to 2 for web class, and from 7 to 1 for the name-srv class. For C4.5 and Naive Bayes, the false positive counts are the same for all the scenarios. In Table 4, the number of false negatives (*i.e.*, the number of C&C flows classified as other legitimate applications) are shown. For example, for logistic regression classifier 4 C&C flows are classified as web class when timing-based features are calibrated in both the training and test sets, compared to only 1 C&C flow classified as web when the timing-based features are not calibrated.

5 Summary and Discussion

In this paper, we presented a method for salting *timing-calibrated* simulated C&C botnet traffic to the LBNL public traces. We experimented using supervised flow classification algorithms to evaluate the accuracy of detecting C&C traffic in four scenarios: (1) when the timing-based features of the botnet C&C traffic are not calibrated with the background traffic in both training and test set; (2) when they are calibrated with the background traffic by the application of Algorithm 1 in both training and test set; (3) when the timing-based features are not used for classification; and (4) when the timing-based features of the C&C traffic are not calibrated in the training set, but are calibrated in the test set. We demonstrated the changes in overall classification accuracy, C&C traffic recall, and precision rates for all scenarios. It was found that presence of any timing artifacts in botnet traces leads to changes in classifier results for some classifier models, and this could result in a serious impact on detection and false negative rates. Our experiments yielded several insights:

First, even though we designed a classifier using the first 5 packet statistics after the 3 way handshake, we detected misclassification of botnet flows by calibrating only a few RTT samples of 2 timing-based features. Methods relying more heavily on timing-based features of complete flows (that consist of more RTT samples) may be even more vulnerable, with smaller attack recall and precision rates. Second, by altering the network traffic patterns over time, a botmaster may change the distribution of the C&C traffic timing-based features. In this way, the botmaster may mimic the legitimate applications to disguise the botnet traffic [32]. So, we may interpret our detection results are for "worst case" calibration of timing features. Third, our experiments showed that the employed feature subset excluding the timing-based features is well predictive of the application classes. Using calibrated timing-based features with "external" feature selection procedures for a given classification algorithm may result in more classification errors. Fourth, when porting a flow classifier from one domain to another, it is possible that domain differences in the class-conditional timing-based feature distributions may lead to substantial losses in classification accuracy on the new domain [34]. Fifth, even if the performance of some classifiers (*e.g.*, C4.5, in our experiments) are not sensitive to timing-based features, using these classifiers as a part of ensemble of classifiers with a voting procedure may worsen the final decision. Finally, to overcome the site and time dependency of RTTs, new isolated timing-based features such as application process time may result in better classification performance [12].

References

[1] http://packetstormsecurity.org/irc/kaiten.c.

[2] Cisco IOS NetFlow configuration guide release 12.4. http://www.cisco.com.

[3] Internet Assigned Numbers Authority (IANA). http://www.iana.org/assignments/port-numbers.

[4] LBNL/ICSI enterprise tracing project. http://www.icir.org/enterprise-tracing.

[5] Wireshark homepage. http://www.wireshark.org.

[6] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *Proc. ACM CoNEXT* , 2006.

[7] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding malicious domains using passive DNS analysis. In *Proc. Network and Distributed System Security Symposium (NDSS)*, 2011.

[8] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol and structure independent botnet detection. In *Proc. USENIX Security Symposium*, 2008.

[9] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through IDS-driven dialog correlation. In *Proc. USENIX Security Symposium*, 2007.

[10] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proc. Network and Distributed System Security Symposium (NDSS)*, 2008.

[11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[12] M. Jaber, R. Cascella, and C. Barakat. Can we trust the inter-packet time for traffic classification? In *Proc. IEEE International Conference on Communications (ICC)*, 2011.

[13] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for high performance. *RFC 1323*, *www.ietf.org*, 1992.

[14] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *Proc. IEEE INFOCOM*, 2004.

[15] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM SIGCOMM Computer Communication Review*, 32(3):75–88, 2002.

[16] V. Krmíček and T. Plesník. Detecting botnets with NetFlow. Presentation given at FloCon Conference, Salt Lake City, UT, 2011.

[17] N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Machine Learning*, 59(1):161–205, 2005.

[18] W. Li, M. Canini, A. Moore, and R. Bolla. Efficient application identification and the temporal and spatial stability of classification schema. *Computer Networks*, 53(6), 2009.

[19] C. Livadas, R. Walsh, D. Lapsley, and W. Strayer. Using machine learning techniques to identify botnet traffic. In *Proc. IEEE Workshop on Network Security*, 2006.

[20] G. Lu and X. Li. On the correspondency between TCP acknowledgment packet and data packet. In *Proc. ACM SIGCOMM Conference on Internet Measurement*, 2003.

[21] R. Mark, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A First Look at Modern Enterprise Traffic. In *Proc. ACM SIGCOMM Conference on Internet Measurement*, 2005.

[22] A. Moore, D. Zuev, and M. Crogan. Discriminators for use in flow-based classification. *Technical Report RR-05-13, University of Cambridge*, 2005.

[23] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. In *ACM SIGCOMM Computer Communication Review*, 2006.

[24] R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. In *Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 339–351, 2003.

[25] J. Quittek, T. Zseby, B. Claise, and S. Zander. RFC 3917 (Informational): Requirements for IP Flow information export: IPFIX. *www.ietf.org*, 2008.

[26] K. Scarfone and P. Mell. Guide to intrusion detection and prevention systems (IDPS). *NIST Special Publication*, 800:94, 2007.

[27] E. Soner. Integrating passive DNS and flow data. Presentation given at FloCon Conference, New Orleans, LA, 2010.

[28] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An overview of IP flow-based intrusion detection. *IEEE Communications Surveys & Tutorials*, 12(3):343–356, 2010.

[29] W. Strayer, D. Lapsely, R. Walsh, and C. Livadas. Botnet detection based on network behavior. *Botnet Detection*, pages 1–24, 2008.

[30] W. Strayer, R. Walsh, C. Livadas, and D. Lapsley. Detecting botnets with tight command and control. In *Proc. IEEE Conference on Local Computer Networks (LCN)*, 2006.

[31] B. Veal, K. Li, and D. Lowenthal. New methods for passive estimation of TCP round trip times. In *Proc. Passive and Active Network Measurement Conference*, 2005.

[32] C. Wright, S. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Proc. Network and Distributed Security Symposium (NDSS)*, 2009.

[33] X. Zang, A. Tangpong, G. Kesidis, and D. Miller. Botnet detection through fine flow classification. Technical Report CSE11-001, Penn State CSE Dept, Jan. 31, 2011.

[34] G. Zou, G. Kesidis, and D. Miller. A flow classifier with tamper-resistant features and an evaluation of its portability to new domains. In *Proc. IEEE JSAC Special Issue on Advances in Digital Forensics for Communications and Networking*, 2011.

Notes

[1] From [4], files lbl-internal.20041004-1458.port019.dump.anon and lbl-internal.20041215-0510.port008.dump.anon.

[2] The inactivity timer measures the length of time expired since the monitor recorded the last datagram for a given flow. When this threshold expires, the monitor exports the flow. As an example, the default value for NetFlow [2] is 15 seconds.

[3] Packet inter-arrival time statistics are widely used in network intrusion detection systems and Internet application identification. We leave it as a future work to analyze the effects of differences in packet of inter-arrival time on flow classifier accuracy.

[4] Standard error of the mean (SE) is the standard deviation of the sample mean, and defined as $\sigma/\sqrt{N}$ where σ is the standard deviation of samples and N is the sample size.

[5] According to our results, the C4.5 algorithm is invariant to how the timing-based features are handled (whether RTT calibration is used or not). A detailed explanation for this is beyond the scope of this paper.