Detecting Intruders and Preventing Hackers from Evasion by Tor Circuit Selection

Conference Paper · August 2018

DOI: 10.1109/TrustCom/BigDataSE.2018.00074

2 authors, including:

Stephen Huang
University of Houston
94 PUBLICATIONS **631** CITATIONS

SEE PROFILE

2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering

Detecting Intruders and Preventing Hackers from Evasion by Tor Circuit Selection

Zechun Cao
Department of Computer Science
University of Houston
Houston, Texas, USA
zcao3@uh.edu

Shou-Hsuan Stephen Huang
Department of Computer Science
University of Houston
Houston, Texas, USA
shuang@cs.uh.edu

***Abstract*—The widely-used Tor network has become the most popular anonymous network that supports circuit-based low-latency internet connections. However, recent security breach incidents reveal SSH have been used to launch attacks by malicious users. Although a server-side blocking mechanism which can identify SSH connections individually has been proposed, we have found that it is restricted to certain Tor circuit protocol versions and not for all SSH protocol implementations. The prior method is based on the difference of latency in the Tor network which may be subject to hacker manipulation by circuit selection in the Tor network. In this paper, we first present a set of attributes that can be used to detect SSH connection through Tor for all SSH handshake between client and server, by observing the network packets exchanges of the SSH protocol. In the second half of this paper, we show that the geographical location of the nodes in Tor circuit has an impact on the effectiveness of our metrics. If hackers know our detection algorithm, they may be able to evade the detection. We demonstrate the effectiveness of our attacks detection by analyzing multiple Tor circuit selections. Finally, we identify and evaluate our detection algorithm and demonstrate that our algorithm achieves 98% accuracy under the most stringent condition.**

***Index Terms*—Intrusion Detection, Network Security, SSH, Tor Network, Circuit Selection**

I. Introduction

Back in 2011, Visa issued a security alert that states: "Insecure remote access, ranging from command-line based (SSH, Telnet) to visually driven packages (pcAnywhere, VNC, MS Remote Desktop), continues to be the most frequent attack method used by intruders to gain access to a merchant's point-of-sale (POS) environment" [1]. Retailer Target has also reported data breach costs of $248 million incurred from its 2013 massive data breach, the largest in U.S. history at the time [2]. Because SSH (Secure SHell) software is widely available on virtually all Unix-inspired operating systems as well as Microsoft Windows, it has become one of the most popular applications to administrate remote servers and transfer files [3]. Moreover, SSH tunneling provides even more applications for users to hide their network traffic or identities. These functionalities leave SSH host at a greater risk of being attacked by insecure remote connections [4], [5].

This project was supported in part by an NSF grant (NSF-1433817) and a DHS Grant (D15PC00185).

Unlike SSH attacks launched through stepping-stones [4], [5], which requires hackers compromise the intermediate hosts to reach the victim server, it is easier for the hackers to take advantage a low latency anonymity network such as "The Onion Router" (Tor). Tor is a distributed overlay network designed to anonymize TCP-based applications like web browsing, secure shell, and instant messaging [6]. It has a broad range of users who want to hide their identities for legal reasons, but recent research conducted by IBM indicated that the Tor network is becoming a source of the malicious traffic that arises from Tor exit nodes [7]. To defend against the malicious Tor users, more and more websites and web service providers are seeking effective methodologies to block all connections from Tor exit nodes.

However, under-blocking and over-blocking are common problems when people seek solutions to block Tor connections by IP address, due to either referring to an outdated Tor exit relay list or blocking an exit node with dynamic IP service. Recent researches show about 3.67% of the top 1,000 Alexa websites block Tor users at the application layer. The problem becomes even more serious when "bottleneck" web services (e.g. CloudFlare, Akamai) whose components are used by many popular website block or discriminate against Tor users by blacklist-based approaches [8]. Examples of the outcomes resulting from these issues include: deterring people from contributing their computer as exit nodes, blocking non-Tor users from popular websites and allowing Tor-originated abuse accessing critical web services. To encourage more and more hosts to donate their resources to Tor network, and prevent over-blocking or under-blocking Tor users if blocking Tor connections is preferred, we propose a per connection- based Tor detection methodology.

Although blocking Tor connections by exit nodes' IP addresses is not desirable as we stated, the research related to other blocking methods are limited. Springall et al. [9] proposed a method to detect HTTP and SSH through Tor network per connection. Their algorithm takes protocol level timing signature for HTTP and SSH prospectively and creates metrics to identify inbound traffic through Tor. In their analysis, the authors measured three different metrics during SSH handshake process. However, we find that the measurements used in their paper are not generic across different SSH

2324-9013/18/31.00 ©2018 IEEE
DOI 10.1109/TrustCom/BigDataSE.2018.00074

implementations. Especially with performance aggressive SSH applications, there are far fewer packets exchanges with no order guaranteed between client and server. In this paper, we first propose a more generic and robust method that is evaluated by the data extracted from real SSH connections across different platforms and applications. Then we improve our algorithm to prevent possible attacks launched by manipulating Tor circuit selection.

There are several projects aim at evaluating and improving Tor's performance or anonymity with different circuit selection strategies. Annessi et al. [10] proposed a Round Trip Time (RTT) based method named Circuit-RTT to reduce Tor circuit's latency by actively measuring the RTT. The algorithm drops the circuit if its RTT is above a client-specific threshold, therefore minimize the delay of the Tor circuit. Akhoondi et al. [11] leveraged geographical distance as a proxy for latency and concluded that the geographic locations of relays have a significant impact on the latency of Tor circuit. LASTor, their proposed method, selects the lowest cost path to the destination and improved the performance of the circuit by 25% compared to the original Tor design. Sun et al. [12] presented a suite of new attacks, called Raptor, that can be launched by Autonomous Systems (ASes) to compromise the anonymity of Tor users. Meanwhile, the authors advocated that favoring closer guard relays is one of the approaches that can help prevent the threat of the AS-level attack. This results in having fewer ASes in the Tor circuit on the entry side of the traffic, and lower the chance that a hacker can observe on both ends of the traffic.

The remainder of this paper is organized as follows: We review the handshake process of both direct and via Tor network SSH connection in Section 2. We present the details of our detecting metrics in the SSH handshake process, including the experiments and discussion, followed by detection algorithm in Section 3. In Section 4, we define three Tor circuit selecting methods, and validate our findings by showing experimental results under the worst possible situation. The last section concludes our findings and provides directions for future work.

II. SSH Handshake and Tor Impact

We first present an overview of the SSH handshake process, focusing on the packets ordering and their timing information. Then we introduce Tor network and the way it changes SSH handshake process to provide an anonymous connection for the users.

A. SSH Handshake

Since SSH runs on top of TCP protocol, a TCP connection must be established between client and server before SSH handshake process. As shown in Figure 1, we depict the packets exchanges in order based on protocol description for both TCP and SSH handshake process. TCP connection requires a three-way handshake process, and the server must wait for the last packet from the client, ACK-ACK packet, to complete the TCP three-way handshake process [13]. After the TCP connection is established, both sides start the SSH handshake process immediately. In the rest of this section, we focus on three major SSH handshake processes and introduce the packets we collect for the measurements of our algorithm.

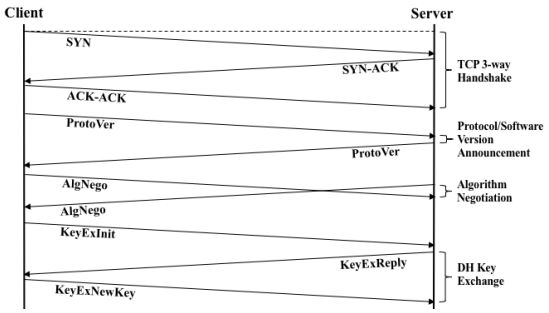

Fig. 1: SSH Handshake via a Direct Connection.

1) SSH Handshake — Protocol/Software Version Announcement: This is the first packets exchange happens between client and server to establish an SSH connection. The purpose of this process is to make sure two peers are having a compatible version of SSH protocol. Two packets in total are exchanged in this section: the server sends its protocol version and the client replies with its own version.

Since the Secure Shell (SSH) Transport Layer Protocol [14] does not clearly specify the order of exchanging the packets that contain the protocol versions, either server or client can send out the packet before receiving from the other end. Most of the SSH client and server implementations, including commonly used OpenSSH, adopt performance aggressive approach to handle this problem. Instead of waiting for the arrival of the packet, either side should send out the necessary packets immediately after the completion of the last step. Based on the discussion about the TCP handshake process before, the client always learns of the completion of the TCP connection earlier than the server as depicted in Figure 1. This "delay of acknowledgment" is usually significant, and causes client's protocol version packet arrives at server side slightly earlier than the server sends out its own copy.

2) SSH Handshake — Key Exchange Algorithm Negotiation: The primary objective of this step is to negotiate the algorithm for encryption key exchange, bulk data encryption, message integrity, and compression. Each peer presents its supported and preferred methods in comma separated list form [14]. Similarly, key exchange algorithm negotiation consists of two packets in which the client and server wrap supported algorithms' list. Since there is no order guaranteed for sending the algorithms' list packets based on the protocol, the actual time of sending and receiving the algorithms' list depends on the previous protocol/software version announcement process. However, in a direct SSH connection, the server side sends out its algorithm's list packet earlier than receiving the one from the client. This is because the server receives protocol version packet earlier than the client during the previous step, hence it sends out the algorithms' list packet

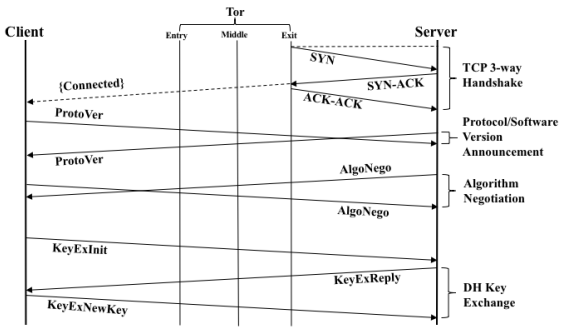

Fig. 2: SSH Handshake over Tor Network.

without waiting.

3) SSH Handshake — Diffie-Hellman (DH) Key Exchange: The DH key exchange provides a shared secret that cannot be determined by either party alone [14]. Upon the successful completion of the algorithm negotiation processes, both sides should start the key exchange immediately. Although we found the number of exchanged packets varies among different implementations, there are four common data packet exchanges in this process. In contrasts to the previous two stages in the SSH handshake process, DH key exchange process guarantees the order of packets for the computing and verification purposes. Therefore, each side should wait until receiving the packet from the other end to proceed to the next step.

B. SSH Connection via Tor Network

Tor is a popular open source project and provides anonymity service for TCP applications. By default, Tor network constructs a three-node circuit for the connection between client and server, as illustrated in Figure 2.

When the SSH client connects to the server through Tor, it first contacts the software called Onion Proxy (OP), which is implemented as a SOCKS proxy locally. The OP learns the destination IP address and port. Since OP has all the nodes information in the chosen circuit, it sends the destination IP address and port to the exit node through Tor. Then the exit node establishes a TCP connection with destination server and sends a success message back to the client. Then the OP starts accepting data from client's application, packs it into 512-byte fixed cells and transmits to the server through Tor [15].

Consequently, the client has to wait until receiving the relay "connected" command cell from the exit node to send out the protocol/software version packet. Because transmitting relay "connected" command from the Tor exit node back to the client introduces extra network latency, it takes longer to send out the protocol/software version packet from the client side compare to the direct SSH connection. Although the client always initiates the protocol version exchange process in direct SSH connection, it is not necessarily the case in the Tor connection scenario. Since Tor exit nodes act like "client" in the connection, the TCP connection is only between the exit node and the server. Hence once the TCP connection is completed at the exit node, it will send out a Tor command cell "connected" back to the client. After receiving this Tor command cell, the client acknowledges the establishment of TCP connection with the server, and start sending the next packet.

III. Detecting SSH Connection through Tor Network

In this section, we extend our algorithm, which can detect SSH connections through Tor prior to any of the data packets being transmitted between the client and server, to a more generic form that accommodates various situations during the SSH handshake process.

A. TCP Round Trip Time

As SSH protocol rides on top of TCP protocol, establishing SSH connection starts with TCP three-way handshake process. We first extract the server system timestamp of the *SYN-ACK* packet sent to the client as $t_{SYN-ACK}$, then $t_{ACK-ACK}$ as the timestamp of the server receives client's *ACK-ACK* packet. To estimate the overall latency of the TCP handshake process, we define a round-trip time named RTT as the elapse time between timestamps $t_{SYN-ACK}$ and $t_{ACK-ACK}$:

$$RTT = t_{ACK-ACK} - t_{SYN-ACK}$$

RTT usually varies among different connections, because the clients connected to the server often reside in different network environments. Although RTT is computed during the TCP handshake, it provides a latency estimation for the following SSH handshake process.

B. Detection Metrics and Algorithm

For each stage in SSH handshake process, the actual number of packets exchanged between the client and server is dictated by client and server's operating systems, as well as the SSH applications currently being used. To ensure this detection method is applicable to different scenarios, we only pick one required pair of packets by the SSH protocol in each stage of the handshake process. Since our algorithm is designed to be implemented on the server side, packet timestamps discussed in this paper are all based on server's system local time.

We first define a timestamp T_c on the server side as the moment it receives the packet from the client. Then, we define another timestamp T_s when the server sends out its packet to the client. Therefore, we can define a real number G, as the time gap measured in seconds on the server side. Additionally, G also indicates the order of sending and receiving the pair of packets on the server side. We then use set $\{p, a, k\}$ to denote Protocol/Software Version Announcement (p), Algorithm Negotiation (a) and Diffie-Hellman Key Exchange (k) respectively, and compute $G^{(i)}$ for each of the SSH handshake stage. Thus, we find $G^{(i)}$ as:

$$G^{(i)} = T_c^{(i)} - T_s^{(i)},\ i \in \{p, a, k\}$$

For $G^{(p)}$ in the Protocol/Software Version Announcement stage, $T_c^{(p)}$ is the timestamp of the packet sent from the client

describing its protocol/software version, while $T_s^{(p)}$ is taken from the packet sent from the server that contains server's protocol/software version. Following Protocol/Software Version Announcement, the server and client must exchange their supported list of algorithms by a single pair of packets. To compute $G^{(a)}$, we extract $T_c^{(a)}$ from the timestamp of the algorithms list packet from the client and $T_s^{(a)}$ from the corresponding packet originated from the server. During the DH Key Exchange process, it may involve various packets that need to be exchanged, depending on the algorithm agreed by both parties in the preceding stage. Regardless of the chosen algorithm, this stage ends with both parties sending "SSH_MSG_NEWKEYS" messages. Therefore, we measure time point $T_c^{(k)}$ and $T_s^{(k)}$ as the timestamps of the client's and server's ending messages respectively.

In the effort of separating direct SSH connection from SSH through Tor by $G^{(i)}$, we use TCP round trip time RTT to cancel out the latency heterogeneity among connections during this brief period. By computing the ratio relative to RTT, we define a ratio R as our detection metric. Similar to $G^{(i)}$, the metrics $R^{(i)}$ for all three handshake stages as in set $\{p, a, k\}$ can be computed as:

$$R^{(i)} = \frac{G^{(i)}}{RTT},\ i \in \{p, a, k\}$$

With metrics $R^{(i)}$ defined for each SSH handshake stage, we formally propose our algorithm as in Algorithm 1.

Algorithm 1 Detecting SSH Connection via Tor Network

Input: Inbound SSH connection; threshold value v
Output: Class label $l \in \{DirectSSH, \textit{SSH Through Tor}\}$

$t_{SYN-ACK} \leftarrow$ TCP handshake packet *SYN-ACK*
$t_{ACK-ACK} \leftarrow$ TCP handshake packet *ACK*
$RTT \leftarrow t_{ACK-ACK} - t_{SYN-ACK}$
$T_c^{(i)}, T_s^{(i)}, i \in \{p, a, k\} \leftarrow$ SSH handshake packets
$R^{(i)} \leftarrow \frac{T_c^{(i)} - T_s^{(i)}}{RTT}$
if $R^{(i)} > v$ **then**
    $l \leftarrow$ *SSH Through Tor*
else
    $l \leftarrow$ *Direct SSH*

To acquire the threshold value v described in Algorithm 1, a training process for the metric $R^{(i)}$ is required prior to the detection process. In the training process, we select v value that achieves the maximum accuracy in detection for the training dataset.

Then we set up the preliminary experiments to validate the effectiveness of metrics $R^{(i)}$ on separating direct SSH connections from SSH connections through Tor.

C. Experiment Setup

To simulate a network environment as in the real world, we chose 4 Amazon AWS servers that are geographically scattered by large distances. Two of them act as SSH servers, located in Ontario, Canada and Frankfurt, Germany. The others, deployed at Tokyo, Japan and Sydney, Australia, initiate the connections as SSH clients. Therefore, we have 4 different routes for the experiments, as each client can connect to a server to form a unique path.

We installed different operating systems, Windows Server 2016 and Ubuntu 14.04.1, on our 4 computers described above, so the algorithm can be tested on both UNIX-like and Windows platforms. Since SSH is natively supported by Unix-like systems, we use OS's default SSH application OpenSSH on the computers running Ubuntu OS. For Windows system computer that does not support OpenSSH, we install commonly used SSH application Putty to initiate SSH connections in the experiments. In this experiment, Tor software has default configuration on the client computers. Specifically, Tor network selects an exact 3-relay circuit by balancing traffic loads on all available relays.

We collected the incoming SSH packets at the server side by using Tshark, a terminal-based Wireshark, to capture and analyze network packets. Each inbound SSH connection is individually collected and saved into a file for analysis. Note that the algorithm can be modified into a real-time detection algorithm. Considering network environment may change during a day, we take "time variable" into account while collecting data in this experiment. Based on servers' local time, we picked three 6-hour windows and evenly conducted our experiments within those windows. The time windows we picked are 2 am - 8 am in the morning, 12 pm - 6 pm in the afternoon and 7 pm - 1 am at night. Within each time window, we collected 12 direct SSH connections and 12 connections through Tor network with default circuit configuration. Therefore, the dataset we obtained consists of 288 SSH connections (72 per route) for all 4 routes.

D. Experiment Results and Analysis

For each SSH connection collected, we measured and computed the ratios $R^{(i)},\ i \in \{p, a, k\}$ as defined previously for the SSH handshake process. Consequently, we have a two-class, 288 instances dataset for each metric in $R^{(i)},\ i \in \{p, a, k\}$. To further evaluate our algorithm with $R^{(i)},\ i \in \{p, a, k\}$ for the classification performance separately, we consider each metric as a binary classifier with connection type as class label. Then we compare all three classifiers' ROC curves, to evaluate their detection success rate (true positive) over false positive rate.

From the ROC curves obtained from $R^{(i)},\ i \in \{p, a, k\}$, their true positive rates all reach over 0.75 when the false positive rates are at 0. Moreover, $R^{(a)}$ has the worst performance among all three classifiers, it only achieves 0.83 on the true positive rate along the y-axis. On the other hand, classifiers $R^{(p)}$ and $R^{(k)}$ both outperform $R^{(a)}$ in our experiment. $R^{(p)}$'s true positive rate reaches 1.00, while its false positive rate maintains as low as 0; Meanwhile, $R^{(k)}$ only trails in true positive rate by a little bit compared to $R^{(p)}$, achieves 0.99 with false positive rate kept at 0.

IV. Tor Circuit Selection and Impacts on Detection

In this section, we are trying to answer the question: do the effectiveness of three detection algorithms presented earlier in the paper depends on the Tor's connection path? If they do, the hacker may be able to evade the detection by manipulating the connection path of the Tor circuits.

A. Tor Circuit Selection Strategies

To analyze if malicious users can defeat us by manipulating Tor circuit selection settings, this section elaborates on the impacts of different Tor circuit selection strategies and how our algorithm should cope with them.

In addition to the default Tor's circuit selection, Tor users have the flexibility to select their routers' (sometimes called relays) geographic location. Therefore, we discovered and defined three different routing strategies that Tor users could choose from to build a Tor circuit, they are TorDefault (TD), TorNearClient (TNC) and TorNearServer (TNS) respectively. With thousands of active Tor relays around the world, TD doesn't provide any restriction to the locations of the relays. It is also the strategy we adopted in the experiments in Section 3. On the other hand, TNC strictly restricts all relays in the circuit to be physically close to the client, while TNS restricts them close to the server. For example, a Tor circuit is considered as a TNC circuit if the client is trying to connect to an overseas server (such as the U.S.), but all Tor relays are located within client's city (such as Hong Kong).

Although we have evaluated the metrics $R^{(i)},\ i \in \{p, a, k\}$ for three stages in the SSH handshake process in Section 3, their performance under the influences of Tor circuit selection strategies remains unknown. In the rest of this section, we will further extend our preliminary experiments in Section 3 to re-evaluate the performance of three metrics in the worst possible scenarios with Tor circuit manipulation.

B. Tor Circuit Selection Experiments

To find the extreme situations in which the users take advantage of Tor circuit selection methods, we extend our experiments in Section 3 by initiating new SSH connections through Tor by TNC and TNS strategies for the same set of routes. As Tor users can manipulate relays location by excluding undesired country codes, we set all relays within the same country with the client or server. Compared to the original dataset in Section 3, TNC and TNS have the same number of connections collected in 3 identical time windows as in Direct and TD. Thus, we obtained an extended dataset that contains 144 connections for each connection type, and 576 connections for all 4 routes.

As we discussed at the end of Section 3, the binary classifiers built by $R^{(p)}$ and $R^{(k)}$ both outperform the one by $R^{(a)}$. With extended dataset including TNC and TNS connections, we plot $R^{(p)}$ values in ascending order for all 4 connection types in Figure 3.

In Figure 3, $R^{(p)}$ values are sorted from small to large for all 4 connection types: Direct SSH, TD, TNC and TNS. Due to the space limitation, only $R^{(p)}$ is shown with a full scale of all 144 connection on the x-axis in Figure 3. We found that SSH direct connection has the lowest $R^{(p)}$ values on the graph, while TNC, TD, and TNS are clearly showing higher $R^{(p)}$ values with noticeable gaps in between. Although not plotted in this paper, $R^{(a)}$ and $R^{(k)}$ are showing the identical pattern with $R^{(p)}$ for the 4 connection types. Among 4 connection types curves plotted in Figure 3, it is interesting to observe that Direct SSH and TNC have the smallest gap between them compared to other curves. Under closer observation on the lower range of both curves, we found it is impossible to separate them by a single cutoff threshold. In other words, it is the most challenging task to detect SSH through Tor from Direct SSH connections if the clients deliberately choose all Tor relays geographically close to themselves. Therefore, we further evaluate the performance of the classifiers build by $R^{(i)}$ in ROC curve shown in Figure 4 with focus on Direct SSH and TNC as class labels.

Compared to the ROC curves obtained through TD connections in Section 3, the three curves in $R^{(i)},\ i \in \{p, a, k\}$

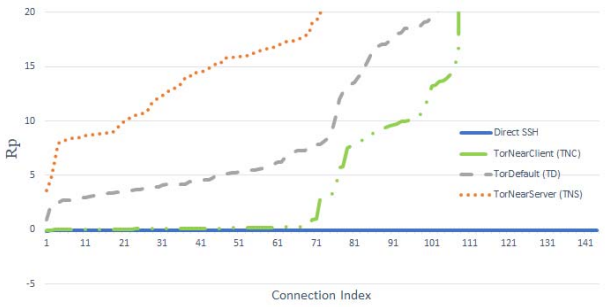

Fig. 3: $R^{(p)}$ in Ascending Order for 4 Connection Types

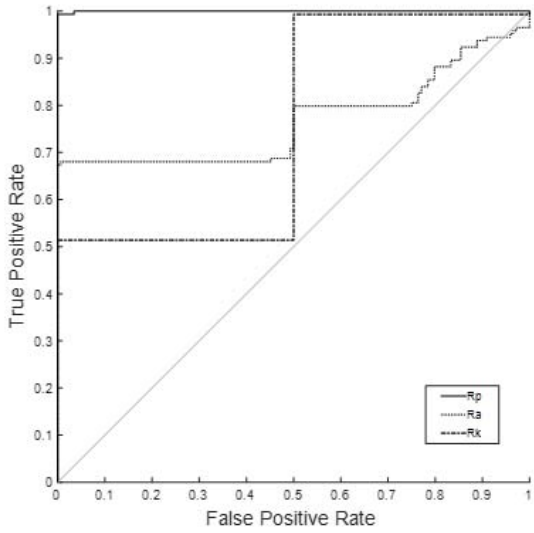

Fig. 4: Success Rate of Classifying SSH Connection via TNC Tor Network by Three Metrics.

show different success rates in classifying TNC and Direct SSH connections. With a 0 false positive rate, $R^{(k)}$ remains high true positive rate at 99% while $R^{(a)}$ and $R^{(k)}$ both go down to 68% and 51% respectively. Although this finding partially contradicts the conclusion from Springall et al. [9] where $R^{(p)}$ and $R^{(k)}$ both considered as good classifiers to use in the algorithm, we concluded that $R^{(p)}$ is the only reliable metric among the three with the influences of choosing all Tor relays close to the client, i.e. TNC.

To demonstrate the validity of our algorithm, we used 4-fold cross-validation on the extended dataset. Since TNC connection has the most similar feature with Direct SSH, we filter out TD and TNS instances in the dataset to leave the worst situation only for the detection. Therefore, we have 144 instances for each class and total of 288 instances for the entire dataset for the validation. The 4-fold cross-validation process first randomly split 288 instances dataset into 4 folds, and 3 of these folds are used as training set to generate the threshold value v to be evaluated on the other fold. This process iterates 4 times until all 4 folds have been evaluated independently for the accuracy. Then the accuracy of the algorithm will be determined by the average value of the accuracy across all 4 iterations. Following the procedure stated above, we validate our algorithm with a single-node decision tree model that reaches the accuracy of 98.26%.

V. Conclusion

In this paper, we have proposed several algorithms to detect SSH connections through Tor network. We eventually selected one of the three algorithms which have been shown to resist hacker's manipulation. Our method monitors the packet exchanges during SSH handshake process. By comparing the latencies of some of the packet exchange RTTs, we are able to identify the SSH connections that travel through Tor. Our experiments show that by tolerating a false positive rate of 3%, 99% of the test cases can be correctly detected. Moreover, the method has 98.26% accuracy during our 4-fold cross-validation over the dataset contains 288 instances.

Our approach differs from previous work in that previous method is only applicable to certain operating systems and SSH applications. As detection method that deployed on the server side, it is not realistic to make any assumptions about the incoming SSH connections. The approach proposed in this method filters out other possible packets introduced by compatibility issues, and only focuses on the mandatory pairs of packets exchanged between client and server. Furthermore, we do not assume the order of the packets that are expected to receive, so our algorithm is more robust compared to the previous work. We also find the challenges to the previous method caused by locating Tor circuit relays by three different strategies, and prove that previous method could be invalid in certain scenarios. After further modifications to the original method, our algorithm remains high performance even in the worst situation possible.

We note that there are several limitations to this paper. First, our validation of the method is limited to a small-scale dataset that we had collected. The dataset contains 288 connections in two classes, which may not reflect all statistical characteristics precisely. Second, we have noticed that malicious Tor users could deliberately send out protocol/software announcement packet sooner than it supposed to be. Although we have not found any real system been attacked by this approach, it could impair our algorithm to a certain extent. In the next immediate step for this paper, we should defend against this type of attack to our method and prove its validity on a larger scale dataset.

References

[1] Visa. (2014, Jul.) Insecure remote access and user credential management. [Online]. Available: https://www.visakorea.com/dam/VCOM/download/merchants/Visa_Security_Alert_070114.pdf

[2] N. E. Weiss and R. S. Miller, "The target and other financial data breaches: Frequently asked questions," Congressional Research Service, Cambridge, MA, Congressional Research Service Reports, 2015.

[3] D. J. Barrett and R. E. Silverman, *SSH, The Secure Shell: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2001.

[4] W. Ding and S.-H. S. Huang, "Detecting intruders using a long connection chain to connect to a host," in *Proceedings of the 2011 IEEE International Conference on Advanced Information Networking and Applications*, ser. AINA '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 121–128. [Online]. Available: http://dx.doi.org/10.1109/AINA.2011.109

[5] W. Ding, K. Le, and S. H. S. Huang, "Detecting stepping-stones under the influence of packet jittering," in *2013 9th International Conference on Information Assurance and Security (IAS)*, Dec 2013, pp. 31–36.

[6] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251375.1251396

[7] IBM, "IBM x-force threat intelligence quarterly," IBM, IBM White Paper, Aug 2015.

[8] S. Khattak, D. Fifield, S. Afroz, M. Javed, and S. Sundaresan, "Do you see what i see? differential treatment of anonymous users." 2016.

[9] A. Springall, C. DeVito, Shou-Hsuan, and S. Huang, "Per connection server-side identification of connections via tor," in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, March 2015, pp. 727–734.

[10] R. Annessi and M. Schmiedecker, "Navigator: Finding faster paths to anonymity," in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, March 2016, pp. 214–226.

[11] M. Akhoondi, C. Yu, and H. V. Madhyastha, "Lastor: A low-latency as-aware tor client," in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 476–490.

[12] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "Raptor: Routing attacks on privacy in tor," in *Proceedings of the 24th USENIX Conference on Security Symposium*, ser. SEC'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 271–286. [Online]. Available: http://dl.acm.org/citation.cfm?id=2831143.2831161

[13] C. Kozierok, *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*. San Francisco, CA, USA: No Starch Press, 2005.

[14] C. M. Lonvick and T. Ylonen, "The Secure Shell (SSH) Transport Layer Protocol," RFC 4253, Jan. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4253.txt

[15] X. Fu, Z. Ling, J. Luo, W. Yu, W. Jia, and W. Zhao, "One cell is enough to break tors anonymity," in *Proceedings of Black Hat Technical Security Conference*, 2009, pp. 578–589.