Finding Proxy Users at the Service Using Anomaly Detection

Allen T. Webb
Dept. of ECE, Texas A&M University
College Station, Texas 77843
Email: allenwebb@tamu.edu

A. L. Narasima Reddy
Dept. of ECE, Texas A&M University
College Station, Texas 77843
Email: reddy@ece.tamu.edu

***Abstract*—Compromised machines or handsets can be used by attackers as stepping stones for accessing sensitive or protected information. We propose a class of detection methods based on anomaly detection at the service and present two lightweight methods of detecting proxies at the service: one for TCP and one for the application layer. These methods can potentially be deployed to monitor connections in real time so attackers may be stopped before accessing sensitive data. We evaluate these methods on local and wide area networks, with different proxy applications, and under different load conditions to show that the proposed techniques can provide high detection rates at low false positive rates. Our techniques are effective even when the client to proxy connections are out of scope of surveillance and resilient to attacks even during training.**

I. Introduction

A proxy server is a software that forwards network traffic. HTTP and SOCKS proxies [1] are used to hide the identity of the source by relaying data through the server. A typical configuration of how a proxy may be used by an attacker is shown in Figure 1. The three hosts shown are a sensitive service, a proxy server (a compromised machine), and a proxy client. The sensitive server provides some service the attacker has interest in. Some examples are internal databases, online banking websites, or corporate file servers. The proxy server acts as a middle man and initiates a connection to the sensitive service on a request from the proxy client. The proxy client connects to the proxy server to communicate with the sensitive service anonymously. From the point of view of a service, the IP address of the requester and TCP parameters may be identical between direct traffic and tunneled traffic. Although proxy servers have legitimate uses, they can also be utilized by those with malicious intent.

Trojans and backdoors provide attackers a foothold on the network. Even with anti-virus software, firewalls, and intrusion detection, new threats emerge that have been tailored to disable

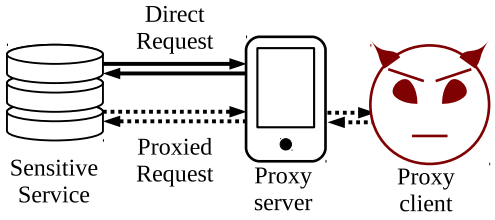


Fig. 1: Direct Traffic vs Socket Proxy Backdoor

the protection and/or evade detection. Once a machine is compromised, it can be used for detrimental purposes. For example, the machine could be used as an inside proxy, a stepping stone, that allows an attacker access to sensitive internal network resources. The trusted machine may even connect to the sensitive network remotely through VPN. It then operates in the role of the proxy server in Figure 1 and may still issue legitimate direct requests along with proxy requests. The attackers can then create a covert channel from which they can probe the network, steal private information, compromise other systems, and cause service outages. Preventing the systems from being compromised is preferable, but expecting attacks never to succeed is unreasonable. Thus, we focus on identifying and stopping unauthorized access to services.

Open proxy servers are available on the Internet, and blacklists of their IP addresses are also available [2]. Information from logs of VPN servers and proxy servers can be useful in tracing abusers back to their original IP address. However, when the proxy server is hosted by the attackers themselves on a compromised machine, the attackers are free to implement the proxy servers as they choose. Proxy logs will likely not exist in this case, so there is a need for additional measures to identify attackers in these cases. All this is done after the fact. Would it not be better to prevent leaking information first?

Whenever information systems are managed by a third party such as a cloud provider, the cloud user may have limited ability to enhance the security of their services beyond what is already provided by the third party. Attacks can be detected without collaboration with the service provider by developing methods that operate at the service rather than the network infrastructure. The level of difficulty required for an attacker to compromise a service can be increased by writing security-aware services that have built in detection strategies requiring only a small amount of data. These detection schemes would aim to detect or deter attackers with minimal impact on performance and with minimal increase to the attack surface of the service.

Although there are methods to detect the channels by which an attacker can funnel information outside the network, most rely on having access to more than one flow of that traffic [3] and [4]. In cases where a virtual private network (VPN) is used, the attacker could funnel the data through a side channel without passing it back through the VPN. Examples of channels which an attacker might use to avoid monitoring include

Bluetooth, cellular data, air-gap techniques, and third party infrastructure. In other words the attacker could create a proxy to services using a compromised host's VPN and connect to that proxy through a side channel such as cellular data. This scenario is of interest because it bypasses typical intrusion detection systems, which typically need to have access to the proxy-to-client traffic. Our work specifically investigates techniques for identifying covert proxy traffic at the sensitive server or network resource. Although the proposed detection methods do not provide the originating IP address of the attacker, they do point out that a particular machine is being used as a proxy. Once a compromised machine is detected, attention can be focused on that machine's traffic to identify the attacker and clean off any malicious software or additional security checks can be required.

Our approach employs timing information, specifically delays at the transport layer and application layer measured at the sensitive server to distinguish direct traffic from proxy traffic. The paper makes the following significant contributions: (1) Proposes a new class of stepping stone detection methods based on anomaly detection. Two examples are provided, one at the TCP layer and another at the application layer. These anomaly detection based methods have the following properties: (1a) Only the traffic at the server is required. (1b) The traffic may or may not be interactive (1c) Detection can be performed near real-time allowing for intervention. (1d) Training is per endpoint and detection is per flow thus reducing the computational complexity of network traffic analysis. (2) Proposes a technique for hardening the training process against manipulation by attackers. (3) Shows that the proposed techniques are lightweight and can be effective with a very small number of samples enabling them to detect attacks in real-time.

II. Detection Methods at the Service

Previous work, [5] and [6], has made progress in detecting abuse of privileges. We consider the scenario in which the attacker is stealing data from a service. Our methods can identify tunnels and proxies at the sensitive service providing a way to log, limit, or deny access after detection.

Our approach relies on network delay measurements. Rather than depending on the contents of the traffic, which may be controlled by the attacker, this method relies on properties of the network path and timing of the traffic. Our hypothesis is that the RTT measurements of traffic when the service is accessed by the attacker through the proxy will be different from the RTT measurements when the service is accessed by the proxy machine directly.

Let d_{sp} denote the variable RTTs observed between the server and the proxy, d_{spa} the variable RTTs observed when the service is accessed by the attacker through the proxy, $\{d_u\}$ a set of unlabeled measurements (one that may contain either d_{sp} or d_{spa}), and $\{d_{tr}\}$ a collection of training measurements of d_{sp}. Our hypothesis is that d_{sp} and d_{spa} will have different distributions and will be distinguishable.

Several techniques and measurements can be employed to distinguish d_{sp} and d_{spa}. Our goal is that given training data, $\{d_{tr}\}$, we can identify and assign an accurate label to $\{d_u\}$. In addition, the delays themselves can be measured at different levels to provide different levels of protection.

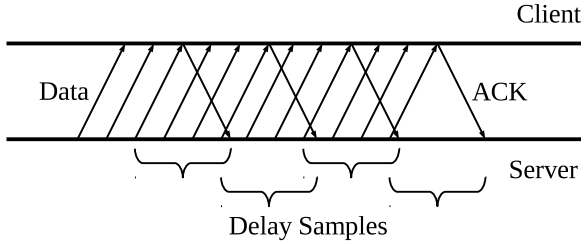


Fig. 2: TCP Delay Distribution Measurements

In this paper, we consider two types of delay measurements, one at the transport layer and the second at the application layer. The transport layer approach has the advantage of not having to modify or enhance the applications whereas the application layer approach can leverage application semantics to enable the design of a better tool.

We present two detection methods: the TCP Delay Distribution (TCP-DD) method, which relies on timing information from TCP packets; and the Application Layer Response Time (AL-RT) method, which leverages data dependencies in the application layer. Many machine learning algorithms require training data for each label. Our detection methods fall into the category of anomaly detection because we only have access to training data for one label, direct traffic. Direct traffic from trusted machines can be observed before an attack. Prior measurements for tunneled traffic can only be obtained after an attack has been carried out and identified. Our goal is to identify an attack as it is being carried out to prevent it from being successful. Requests classified as proxy traffic can be required to go through additional security checks or can be denied access to sensitive network services. This method has the flexibility of being deployed as part of an intrusion detection system or as part of a service.

A. TCP Delay Distribution Method

We measure TCP delays at the transport layer. A set of training samples are employed to indicate the characteristics of direct traffic. TCP delays are constantly measured and compared against the training set to see if proxy traffic is originating from the machine. This method does not require modifications to the server or client software. Because it does not require access to the client machine, the detection does not require changes to the application layer software, and it supports the commonly used protocol, TCP. If the intrusion detection system controls a dynamic firewall, unwanted TCP flows could be interrupted or blocked. If this is integrated into a service, the service could limit, or block access to TCP flows or SSL sessions that are not classified as direct traffic. Note that once a session is classified initially, it still may be monitored because sessions can be hijacked [7].

Here is an examination of how this method works: Consider the instantaneous delay between the server resource and the proxy server for direct traffic, d_{sp}, and the instantaneous delay between the proxy and the client, d_{pa}. Although the round

trip time for acknowledgments will follow d_{sp} for the most part, the timing of acknowledgments is typically also affected by techniques to reduce the number of acknowledgments sent such as Nagle's algorithm and TCP large segment offload. As long as these techniques are dependent on whether there is new data to send, there will be information leakage from d_{pa} to the server resource.

Although TCP time-stamps could be used instead of measuring the delay, the TCP time-stamp fields could be forged by the attacker; the attacker could send a future time-stamp. It is conceivable that the attacker could use a custom implementation of TCP to have more control over the distribution of delays between packets and their corresponding acknowledgments. For the attacker to successfully launch an attack, they would need to know the characteristics of direct traffic observed from the service and have a way to replicate them.

The mean values of d_{sp} and d_{spa} could appear to be the same. However, functions other than the mean could reveal differences between d_{sp} and d_{spa}. TCP-DD has the following procedure: Record the delays between a TCP ACK and the most recent TCP segment it acknowledges to obtain an unlabeled set of measurements, d_u, for the RTT. Compute a function on the timings, d_u, such as a binned distribution estimate, a density estimate, the variance, etc. Compare the result with training data for d_{sp} and classify the connection as normal or anomalous.

One category of statistical measures we used provided a comparison of empirical distributions or densities. A simple way to obtain an empirical distribution is binning, the technique used to generate histograms. After binning, the probability mass function is obtained by dividing the value of each bin by the sum of the values of all the bins. Also, a smoothing function such as the Gaussian function (see Equation 2) can be applied to the histogram to obtain a density estimate from sparse data. An expected distribution can be constructed by aggregating all the training data into one distribution.

Raw JSD: The similarity of distributions can be measured using the Jensen–Shannon divergence (JSD) [8]. The JSD is shown in Equation 1 where p_1 and p_2 are the distributions, and X is the set of bins. This allows for the measurements of one trial to be compared to the expected distribution. To identify when a measurement is anomalous it is necessary to have some notion of what range of values the data should have. This can be obtained by using the JSD to calculate a distance between the distribution estimates for each trial and the distribution estimates for the aggregate of the training data. The JSD becomes undefined whenever there is at least one bin with a zero value, the solution we used was to initialize the bins to a very small positive number before performing binning.

$$JS_\pi(p_1, p_2) = \sum_{x \in X} \left[\frac{p_1(x)}{2} \log \frac{p_1(x)}{p_2(x)} + \frac{p_2(x)}{2} \log \frac{p_2(x)}{p_1(x)} \right] \tag{1}$$

For simple binning and JSD there is an order $O(n + m)$ computation cost that scales with the number of samples, n, and the number of bins, m.

Smooth JSD: If a Gaussian (see Equation 2) smoothing filter is applied to the binned results to obtain a different density estimate before using the JSD, it has an order $O(n+m\times l)$ computation cost that scales with the number of samples, n, and the number of bins, m, times the number of filter taps, l.

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{2}$$

Variance: Another statistic measure that we found to distinguish d_{sp} from d_{spa} is the variance. Although the mean between d_{sp} and d_{spa} could be too similar to distinguish them, the variance may not be. Training data isn't needed to show the separability of the variances calculated for the direct traffic trials from the variances calculated for the tunneled traffic, but it is necessary for anomaly detection. This is an order $O(n)$ computation cost that scales with the number of samples, n.

MPGS: The mean pairwise Gaussian similarity (MPGS) also can be used. The Gaussian similarity [9] can be described as a measure of closeness and could be compared with the inverted squared Euclidean distance. The mathematical formulation for this measure is shown in Equation 3. This is calculated for each pair of a training sample with an unlabeled data sample. The results are averaged to obtain the MPGS that is compared against the MPGS values for the trials in the training data. The computational cost for calculating the MPGS is $O(n \times k)$ where n is the number of samples in d_u, and k is the number of training samples.

$$\text{MPGS}(d_u) = \frac{1}{|d_u||d_{tr}|} \sum_{x \in d_u, y \in d_{tr}} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-y)^2}{2\sigma^2}} \tag{3}$$

Mean distance measures are another way to distinguish the direct and tunneled traffic. The mean minimum distance (MMD) from the inter-arrival times, d_u, in an unlabeled set to the times in the training data can also be used. The L1 norm was used for the distance. It is the absolute value of the difference between the values being compared, $|a - b|$. We took the mean of the lower quartile of the distance values. The computation cost depends on the implementation but costs of $O(n \log k)$, and $O(n \log n + k)$ can be achieved where n is the number of samples in d_u, and k is the number of training samples. This was chosen because it uses the k-distance measure, which is part of the formulation for the Local Outlier Factor [10].

B. Application Layer Response Time Method

This method estimates the round-trip-time by measuring the end-to-end delay at the application layer. Socket proxies may hide the RTT at the transport layer, but they forward the unmodified application layer data. One potential way of identifying tunneled traffic is to measure the RTT at the application layer. Ideally, this would be done without requiring changes to the application layer protocol. Many applications have data dependencies between served content and requests. The time delay between a message being sent and the receipt of the first dependent message can be used to bound the round trip time between the server and client applications. For this to be an effective way of detecting a tunnel, the attacker should not

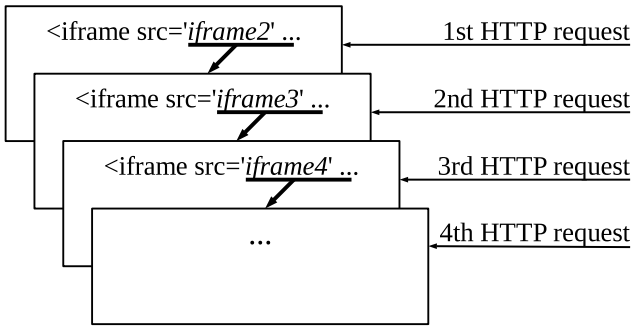


Fig. 3: HTTP Response Time Method

be able to send the dependent message early. This challenge can be solved by requiring a sufficiently hard to guess random token in the dependent message.

In this paper we use HTTP to validate the concept of the Application Layer Response Time Method. HTTP serves pages typically containing references to additional content such as images, scripts, styles, or frames with other pages. The HTTP example for determining if a visitor to a website is using a proxy or not is as follows: Create a chain of iframes such that a browser will not cache the iframes (they will be loaded every time), and the delay between HTTP requests is logged. Nonces are used to ensure that each iframe in the chain must be loaded one after the other. This establishes a data dependency which is used to measure the delay from the browser to the webserver. We can then compare a function of the delays between requests for the iframes to training data to determine if they are anomalous. Although we focus on HTTP in this paper here, it is possible to extract delays at the application layer similarly with other protocols.

Iframes were chosen because they allow for chaining multiple measurements one after the other even when JavaScript is disabled. Figure 3 shows how embedded iframes can be used to create data dependencies for measuring the end-to-end RTT. If only one measurement is needed, an image or style file could be used instead.

An attacker may employ an application-layer proxy server. This would potentially defeat TCP-DD; however, without prefetching the proxy server would still leak the end-to-end time to HTTP-RT. Prefecting proxy servers neglect images by default because of the cost of prefetching all the images on all related pages, so images could be incorporated into the HTTP-RT method to prevent prefetching of certain resources. The access times of these non-prefetched resources would then be used in the HTTP-RT measurements.

For HTTP-RT the absolute minimum round trip time between the service resource and the proxy server is a lower bound; attackers cannot cross except in highly improbable situations: Assume a set of nonces is chosen randomly from a set such that the chance of guessing each nonce is much smaller than the number of times the valid user connects. A nonce is sent as part of a URL for the destination of a resource such as an iframe. Each resource loads another resource at a URL including the next nonce. The first nonce isn't known until after the first page loads, the second until after the first resource loads, the third until after the second resource loads,

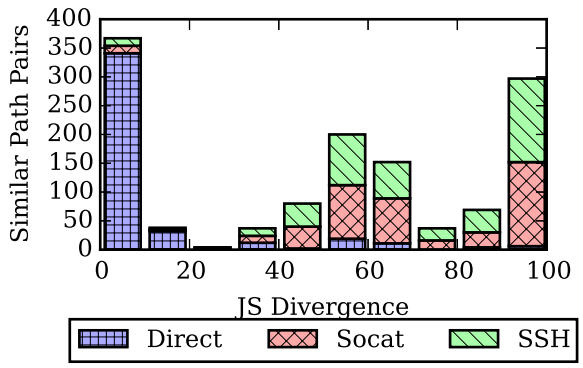


Fig. 4: Jensen Shannon Divergence Between Training Data from Similar Paths. Note that the proxy data, SSH and Socat, are included to show where an attacker's distribution would fall, but these would not be available as training data.

and so on. For the attacker to request the next resource they must know (or guess) the nonce. An incorrect guess identifies an attack. The probability of the attacker correctly guessing all the nonces in a chain of dependent resources quickly drops off to zero. The chain provides more data points to reduce the effects of noise. If the attacker receives the nonce through the proxy server, the delay will be greater than the delay between the proxy server and the requested service.

C. Identifying proxies during training

Adaptive methods or retraining open up the possibility of the attacker manipulating the results of the training data. Any classification method that uses training data is sensitive to the quality or truthfulness of that data. When these methods are put into a situation with adversaries, one might ask whether an adversary has the ability to taint the training data to thwart the classification method. We propose a solution to prevent attackers from being able to manipulate the training data. Training data from similar network paths can be cross checked to notify the administrator when retraining falls outside the acceptable limits. An example definition of a similar network path is one that is only different in at most the first and last links of the path. Using this definition we tested the similarity of the distributions of training data across similar paths as well as with the proxy data. We collected additional data from Planetlab for groups of nodes which are hosted by the same organization. The results for HTTP-RT are shown in Figure 4. We include the proxy traffic represented by socat and SSH on the plot to show how the attacker's distributions compare with the direct traffic. Ideally the direct traffic would be clustered together, and the proxy traffic would not overlap with the direct traffic. In most cases the direct traffic has a lower Jensen-Shannon Divergence with the similar paths than the proxy traffic. For TCP-DD the results are much less spread, but the direct and proxy traffic were completely separable for 77% of the groups of similar paths. This cross-check method for verifying the training data increases the difficulty the attacker faces. Either they will have to compromise more hosts to launch a successful attack, or they will be limited in how much their attack traffic can vary from hosts on similar paths.

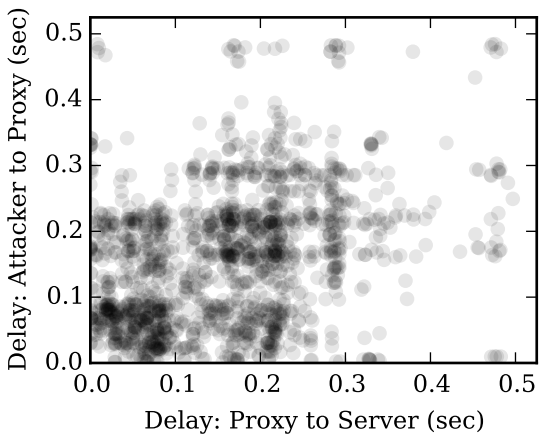


Fig. 5: Path pings of Each Set of Selected PlanetLab Nodes

III. Experimental Evaluation

We evaluated our approach in three settings. The first setting employed PlanetLab [11] with the sensitive server, proxy and the attacker location distributed across the globe. The second setting considered attacks within a local area network where all three parties, sensitive server, proxy and attacker, all reside on the same network. The two settings provide different vantage points with wide-delay measurements and tight-delay measurements. Third we tested the sensitivity of the results to various parameters such as CPU load.

As part of our evaluation we compared the accuracy of different statistical measures for the function computed on d_u. On one hand, this shows that using the RTT measurements has some flexibility. On the other hand, the comparison shows which measures might be best in deploying TCP-DD.

We considered different download intensive communication models that the attacker could employ. With *socat* the traffic is passed unencrypted to the proxy client, whereas with *ssh* the traffic is encrypted between the proxy and the proxy client. Transparent proxies that work by network address translation (NAT) will not hide the end-to-end RTT so they were not tested because the end-to-end RTT should be enough to distinguish direct and tunneled traffic.

A. Wide Area Network Measurements

PlanetLab offers a network testbed spanning diverse geographic locations. Our slice provided 80 active nodes covering different continents with which we conducted our measurements. Figure 1 shows the basic setup we used for each trial. One node functions as the service. It hosts the web server and records access times and packet traces. Another node functions as the proxy server. It hosts proxies both through *socat* [12] and *SSH* [13]. It also makes a direct request to the web server. The last node functions as the proxy client and makes requests to the web server both through the *socat* and *ssh* proxies.

Combinations of three nodes were randomly selected. For each combination all permutations were tested. 1727 working configurations were tested for the 80 PlanetLab nodes. Groups of nearby nodes were also selected to allow for measuring the accuracy when network distances were smaller. Figure 5 shows the delays between nodes in each working configuration tested. Notice there is a good spread; in some cases the attacker is closer to the proxy than the proxy is to the server and vice versa. The measurements were taken once every thirty minutes over a 24 hour period.

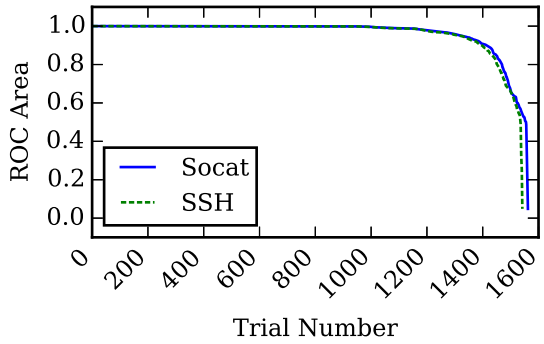
The separability between the proxy traffic and the direct traffic can be seen in the ROC (Receiver operating characteristic) area plots in Figure 6. ROC plots are typically used to show how well a communication channel operates. In our case we are plotting the percentage of true positives achievable by selecting a specific threshold for a percentage of false positives. An ROC area of 1.0 is ideal where all the true positives can be included by a threshold without including any false positives.

Figure 6 shows the results of our experiments using both TCP-DD and HTTP-RT methods and different measures in the PlanetLab setting. Figure 6a shows the TCP-DD measure with the highest accuracy, smoothed JSD. A maximum of forty-eight training points were used when training data was needed in the TCP-DD measures. After analyzing how many training points are needed for smoothed JSD, we found that with five trials most of the separation between the direct traffic and proxy traffic was already achieved. Different training points (d_{sp}) are needed for each pair of a service host with a proxy host because they represent different possible paths. The true positives of the ROC plot are $min(\{d_{sp}\})$ for each trial and the false positives are $min(\{d_{spa}\})$ for each trial. Figure 6b shows the HTTP-RT measure with the highest accuracy, the minimum RTT.

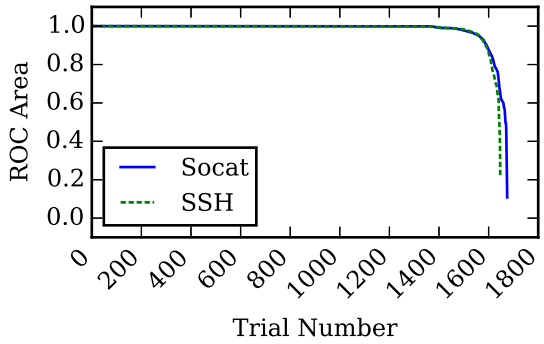
The results between the unencrypted and encrypted socket tunnel implementations, *socat* and *ssh*, are almost the same with *ssh* having a few more measurement failures resulting in fewer trials. The majority of trials had a close to ideal separation. In each of these cases a threshold could be chosen for TCP-DD using smoothed JSD in those trials to distinguish the direct traffic from the tunneled traffic.

In Figure 6c the separability of different functions are used over the samples collected for each trial for TCP-DD. These functions are discussed in more detail earlier in subsection II-A. "MMD." represents mean minimum distance for the highest quartile. "Raw JSD" represents binning the inter-arrival times to obtain a distribution estimate and using the Jenson–Shannon Divergence (JSD). "MPGS" represents using the Mean Pairwise Gaussian Similarity (MPGS). Bins of 25 milliseconds were used, but further optimization could be done for this parameter. "Variance" represents taking the variance of the measurements within a trial. The variance values were then compared across trials to test for separability. "Smooth JSD" represents first binning to obtain a density estimate. The result is smoothed using a Gaussian smoothing filter that is truncated to 4 standard deviations of 1250 μ-seconds. Lastly, then the JSD is used to compare the smoothed density of the training data with the one for the unlabeled data.

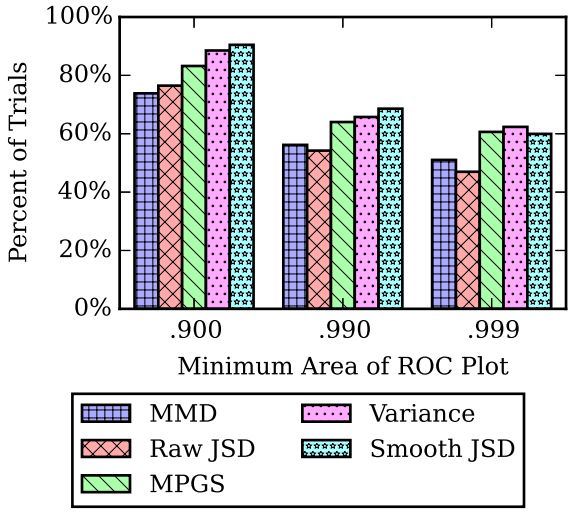
The results show that both TCP-DD and HTTP-RT can separate the direct traffic from proxy traffic. Figure 6c and Figure 6d show the percentage of trials that had ROC areas above 0.9, 0.99, and 0.999. From these figures it is clear that HTTP-RT outperforms TCP-DD for the configurations tested.



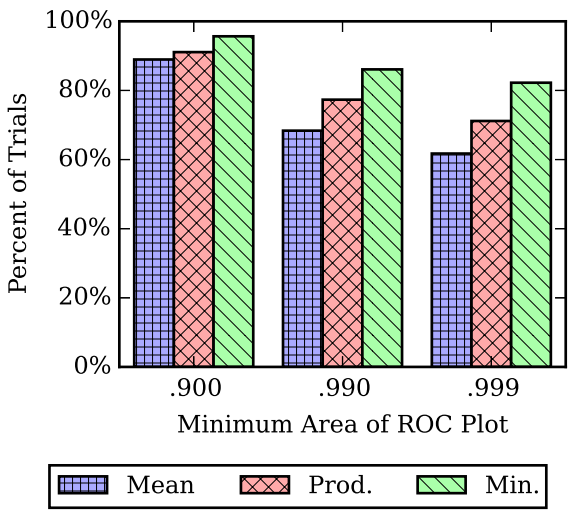
(a) ROC Area of Each TCP-DD Trial Using Smoothed JSD Listed in Descending Order



(b) ROC Area of Each HTTP-RT Trial For Three Samples Using Minimum Listed in Descending Order



(c) TCP-DD Processing Method Comparison with *socat* and *ssh* Combined



(d) HTTP-RT Processing Method Comparison with *socat* and *ssh* Combined

Fig. 6: Performance Comparison of Implementation Options

An ROC area of 0.90 can be likened to a 90% true positive detection rate for 0% false positives. It does not correspond directly because an ROC area of 0.90 may have different shapes. For the rightmost bar of HTTP-RT using the minimum of the measurements, an ROC area of 0.999 and above was achieved for more than 80% of the trials.

For Figure 6d the measured round trip times differ between the direct and proxy traffic, but there is noise present. Three samples of round trip times were used for the HTTP-RT classification. "Mean" represents taking the average of the samples in a trial, "Prod" represents multiplying the samples, and "Min." represents taking the minimum of the samples. All these functions have $O(n)$ complexity over the n samples per trial. Out of all the methods, the minimum function resulted in the greatest separability.

From Figure 6c and Figure 6d it is clear that HTTP-RT outperforms TCP-DD in the tested configurations. Tests were also run for more than one proxy node to confirm that adding more nodes increases the detection accuracy for both methods.

B. Local Area Network Measurements

In a local setting the distances are much shorter, the links are more reliable, and the bandwidth limitations are less. We expected to have a more difficult time detecting the tunnels when the proxy server and client were on the same local network. A single host was used to provide the service. Two different proxy servers were used in different locations. Four different proxy client locations were used with both a Wi-Fi and wired host each. The measurements were recorded every five minutes for at least two and a half hours yielding 30 or more data points for each trial. In this experiment the packets were processed in real time to obtain the TCP ACK delays and iframe delays. In this dataset about twelve TCP ACK delays were recorded per measurement

A good separability was achieved in all the trials. The direct traffic was completely separable from the tunneled traffic in more than half of the trials. For the local measurements TCP-DD outperformed HTTP-RT.

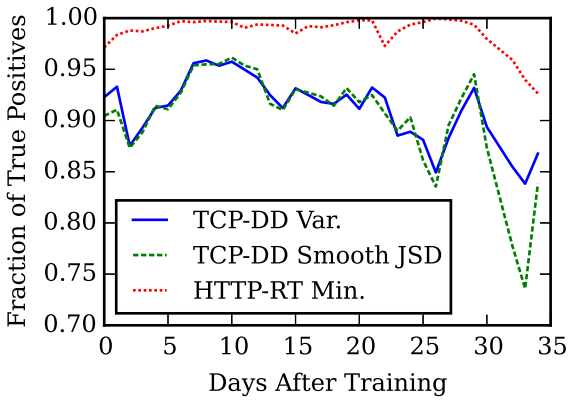


Fig. 7: True positives of Direct Traffic Over Time After Initial Training Averaged Across Trials

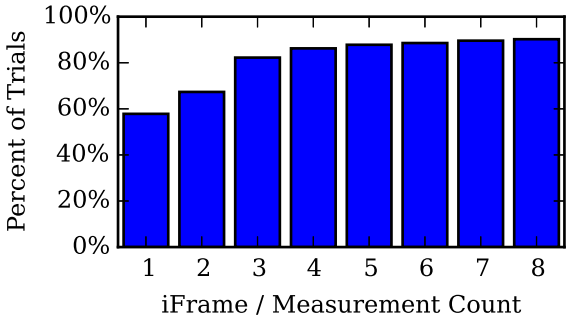


Fig. 8: Percent of HTTP-RT Trials with ROC area above 0.999

C. Sensitivity

Planetlab measurements were taken over a 34 day period to determine how often retraining is needed. The measurements were taken every 30 minutes, and only the first 24 were used as training data. Figure 7 shows the fraction of true positives out of direct measurements averaged across trials plotted over time. HTTP-RT Min. had over a 97% true positive rate after a month. The true positive accuracy of TCP-DD degraded after 20 days. The decision to retrain can be made on a per host basis once the false positive rate is high enough.

How many measurements are warranted for HTTP-RT? Figure 8 shows the relationship between separability and the number of measurements used. A higher percentage corresponds to better separation between the direct traffic and the tunneled traffic. At three measurements three iframes are loaded after the initial page load. It is observed that most of the separability is achieved within 3 measurements. Since the cost in terms of delay to the user increases linearly with the number of measurements, three is probably the best number of measurements to take based on our results.

Figure 9 shows the classification accuracy using separate training data and direct measurements from the PlanetLab data. In these experiments, the measured data contains both direct traffic and proxy traffic. These experiments are designed to see if presence of proxy traffic can be detected when the compromised machine is being used both for direct and proxy

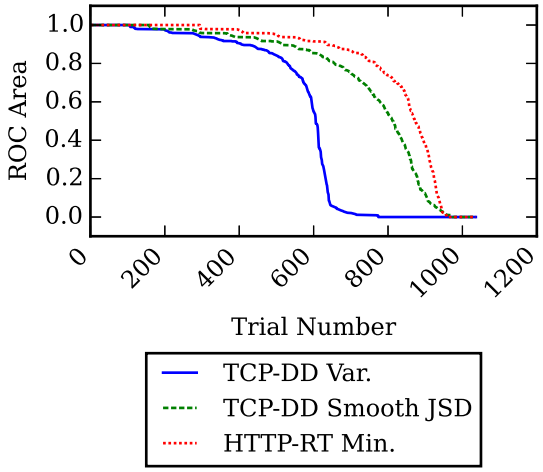


Fig. 9: Classifiers Using $\mu \pm 2\sigma$

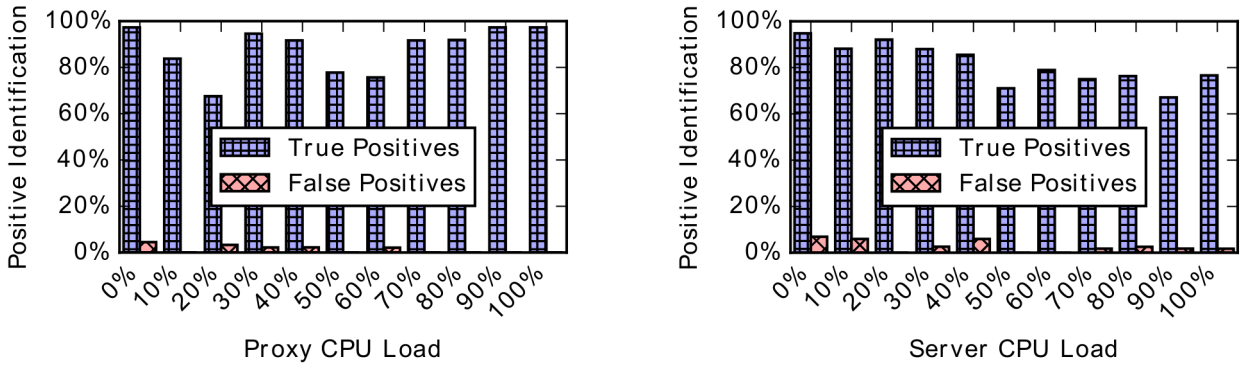
communications. Up to 48 training points were used to find the mean and standard deviation of each measure. The accepted range was defined as $\mu \pm 2\sigma$ for the training data. Separate direct measurements and proxy measurements for both *socat* and *ssh* were tested to determine the experimental true positive and true negative rates. Note that the performance of TCP-DD using the variance is poor in this case even though it has a high separability in Figure 6c. We then used the true positive percentage times the true negative percentage as the ROC area in the plot. Again HTTP-RT outperforms TCP-DD. It is observed that the proposed techniques provide very high detection rates with very low false positive rates across the many experiments.

To investigate the effects of CPU load on TCP-DD we conducted an experiment using *iperf3* [14] under different load conditions. We used the tool in download mode because the traffic from the server to the proxy is what generates the TCP acknowledgments used in TCP-DD. These tests were done with three machines connected through a single switch to eliminate uncontrolled network factors. This allowed us to observe the difference in the steady state delay distribution for different amounts of CPU load. We tested the effects of CPU load at both the server and the proxy. The results are shown in Figure 10.

Varying the load on the proxy case does introduce variance into the detection rates as shown in Figure 10a. There is a noticeable degradation of performance in the server case Figure 10b. The loss is about 2% less true positives for every 10% point increase of Server CPU usage. Since, the CPU load on the server can be measured training could be done across different CPU load regions to increase detection performance if needed. It is noted that even when tested against different loads from the training loads, the proposed methods provide reasonable detection rates.

IV. Related Work

A body of work exists for detecting stepping stones. Recently proxy detection at the service has started to get attention. In [15] motivated by Nagle's algorithm the size of



(a) Effect of Proxy Loading

(b) Effect of Server Loading

Fig. 10: Identification of Direct Traffic Across Load Using TCP-DD Smooth JSD. For each plot a $\mu \pm 2\sigma$ classifier was trained using the data for 0% load. Measurements were conducted using 10 second *iperf3* traces at 100Mbps for each load level. The traces were partitioned into sets of 100 measurements for the classification. Each trace had about 50 partitions.

segments and inter arrival times were used to identify proxies. They rely on a steady stream of small segments to detect proxies, while TCP-DD works for download intensive streams which may have segments at the MTU. They achieved a 94.0% true positive rate with a false positive rate ranging from 0.8% to 85.4% averaging 10.2% across all their trials. This corresponds to an ROC area of 0.844. Machine learning is used for proxy detection in [16]. Their method works well for the proxy server configurations they tested (squid) with a 94.1% true positive rate and a 7.9% false positive rate for distinguishing proxy traffic from direct traffic. This corresponds to an ROC area of 0.867. Using a $\mu \pm 2\sigma$ classifier, HTTP-RT using the minimum has a greater ROC area in 67.4% of our trials, and TCP-DD using smooth JSD in 56.8% of our trials. While they employ timing features for classification, they do not contrast delay distributions as employed in our approach. We chose to use anomaly detection to avoid requiring labeled proxy data which targets the classification on particular proxy implementations and behaviors.

Prior stepping stone detection techniques fall into one of three categories: (1) content based methods, which require clear text or the same content across the observed flows [17] and [18]; (2) passive time based methods, which rely on the assumption of a maximum tolerable delay [18], [19], [20], and [21]; and (3) active time based methods often called watermarking [22] and [23]. The typical threat model used by the above stepping stone detection papers assume all the traffic on the network is being monitored and that the attackers do not have access to an alternative unmonitored Internet connection. Our work does not make these assumptions. Instead of tracing the attack back to an IP address at the edge of the network, we identify connections that have stepping stones. These methods rely on correlating two or more connections to identify stepping stones and thus cannot detect a stepping stone that has only one visible connection. For this reason we argue stepping stone detection at the service can be applied in addition to the previous methods to detect stepping stones that would not be detected otherwise.

Timing of packets has already been shown to betray information in the past such as keystrokes over SSH in [24]. TCP jitter has been used as a means of identifying rouge Wi-Fi access points on a network [25]. It has also be used to defeat location spoofing VPNs in [26].

Work has been done to identify the destination of encrypted HTTP traffic (over WPA and WEP) by using statistical analysis, lengths of the packets, and the distribution of inter-arrival times [27]. Statistical measures of packet lengths [28] and information theoretic measures [29] have been employed to classify TCP flows and to detect anomalous traffic.

Detection of middleboxes such as NATs has been explored through observation of instant messaging traffic [30]. Tracetcp, [31], uses TCP SYN packets and TTL values to perform a traceroute over TCP ports. It can identify transparent TCP tunnels and proxies, but is designed for the client side. It is also dependent on the proxy forwarding the TTL values, which is up to the proxy implementation. Our methods handle proxies at the server side and are not dependent on TTL values.

Work has been done to identify tunnels through application layer protocols such as HTTP [3] and [4]. These are aimed at detecting information leakage from a network, whereas we are addressing the problem of whether trusted clients are compromised and are serving as a proxy.

In [32] HTTP proxy traffic is detected to find attempts to circumvent firewalls, and spyware back channels. This work focuses on detecting anomalies using a variety of different techniques such as the HTTP header fields, request sizes, request rates, bandwidth usage, and request inter-arrival times. Our goal is to find proxies on trusted nodes rather than find trusted nodes that are accessing proxies. A key difference in our work is that instead of using inter-arrival times to look for periodic traffic, we are estimating the end-to-end RTT or comparing inter-arrival time distributions.

There are existing methods for HTTP client fingerprinting that can serve to distinguish proxy clients. Client side Javascript is used for detection in [33]. SSL handshakes are used to fingerprint clients in [34]. A survey of these methods is provided in [35]. If a machine is compromised, metrics used in fingerprinting such as screen resolution, browser plugins, etc. may be determined and spoofed by the attacker. Even the

IP address of the machine can be used by the attacker through proxies. Our work seeks to address this issue.

A. Future Work

Other features could be investigated to determine if the classification accuracy may be improved. Along with new features, methods that perform anomaly detection across a combination of features can be evaluated. Investigation could be done to find a method for preventing the case when an attacker does not use a tunnel but instead funnels the information first to the compromised machine and later extracts it. IPsec [36] could be investigated to see if similar results could be achieved at the network layer. One other possible are to investigate would be to perform training for each group of similar paths rather than each end point. This would reduce the amount of training needed.

V. Conclusion

This paper proposed using anomaly detection as a new class of stepping stone detecting methods by monitoring traffic at the server. These methods handle the case when an attacker has access to unmonitored communication channels, which thwart previous detection techniques. Although compromised machines may be used as proxies, we have presented techniques that can be used to identify such proxies at the service. We have shown two working examples of these methods work on actual networks both on a global scale and a local scale. Our results show that the proposed techniques can provide very high detection rates of up to 99% with very low false positive rates. In addition, we show that comparing training data from similar paths can be used to harden the method against manipulation of the training data. The methods we present may also be combined with other features as part of a machine learning system for proxy detection, or used as an early detection system to supplement more computationally expensive post processing methods.

Acknowledgment

This publication was made possible by the NPRP award [NPRP 5-648-2-264] from the Qatar National Research Fund.

References

[1] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones, "SOCKS Protocol Version 5," RFC 1928, Mar. 1996.

[2] SORBS. (2015) The open SOCKS proxy server. [Online]. Available: http://www.dnsbl.sorbs.net/information/proxy.shtml

[3] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli, "Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting," *Comput. Netw.*, vol. 53, no. 1, Jan. 2009.

[4] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Detecting HTTP tunnels with statistical mechanisms," in *ICC*, June 2007.

[5] Y. Chen, "Protection of database security via collaborative inference detection," Ph.D. dissertation, UCLA, 2007.

[6] N. Baracaldo and J. Joshi, "A trust-and-risk aware RBAC framework: Tackling insider threat," in *SACMAT*. ACM, 2012.

[7] P. G. Sarkar and S. Fitzgerald, "Attacks on SSL a comprehensive study of BEAST, CRIME, TIME, BREACH, Lucky 13 & RC4 biases," ISEC Partners, Tech. Rep., Aug. 2013.

[8] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Trans. Inf. Theor.*, vol. 37, no. 1, Sep. 2006.

[9] J. Kunegis, A. Lommatzsch, and C. Bauckhage, "Alternative similarity functions for graph kernels," in *ICPR*, Dec 2008, pp. 1–4.

[10] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, May 2000.

[11] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: An overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, Jul. 2003.

[12] dest-unreach. (2015) socat - multipurpose relay. [Online]. Available: http://www.dest-unreach.org/socat/

[13] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," RFC 4251, Jan. 2006.

[14] ESnet. (2015) iperf3. [Online]. Available: http://software.es.net/iperf/

[15] R.-M. Lin, Y.-C. Chou, and K.-T. Chen, "Stepping stone detection at the server side," in *INFOCOMW*, April 2011.

[16] V. Aghaei-Foroushani and A. Zincir-Heywood, "A proxy identifier based on patterns in traffic flows," in *HASE*, Jan 2015.

[17] S. Staniford-Chen and L. T. Heberlein, "Holding intruders accountable on the internet," in *SP*, 1995.

[18] Y. Zhang and V. Paxson, "Detecting stepping stones," in *SSYM*. USENIX Association, 2000.

[19] A. Blum, D. X. Song, and S. Venkataraman, "Detection of interactive stepping stones: Algorithms and confidence bounds," in *RAID*, E. Jonsson, A. Valdes, and M. Almgren, Eds., vol. 3224. Springer, 2004.

[20] Y.-W. Kuo and S.-H. Huang, "An algorithm to detect stepping-stones in the presence of chaff packets," in *ICPADS*, Dec 2008.

[21] G. Di Crescenzo, A. Ghosh, A. Kampasi, R. Talpade, and Y. Zhang, "Detecting anomalies in active insider stepping stone attacks," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 2, no. 1, 2011.

[22] X. Wang, S. Chen, and S. Jajodia, "Tracking anonymous peer-to-peer voip calls on the internet," in *CCS*. New York, NY, USA: ACM, 2005.

[23] A. Houmansadr, N. Kiyavash, and N. Borisov, "Non-blind watermarking of network flows," *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, Aug. 2014.

[24] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on SSH," in *SSYM*. USENIX Association, 2001.

[25] G. XIE, T. He, and G. Zhang, "Rogue access point detection using segmental tcp jitter," in *WWW*. ACM, 2008.

[26] A. Abdou, A. Matrawy, and P. Van Oorschot, "Cpv: Delay-based location verification for the internet," *Dependable and Secure Computing, IEEE Transactions on*, vol. PP, no. 99, 2015.

[27] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine, "Privacy vulnerabilities in encrypted HTTP streams," in *PET*. Springer, 2005.

[28] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, Apr. 2006.

[29] W. Lee and D. Xiang, "Information-theoretic measures for anomaly detection," in *SP*. IEEE Computer Society, 2001.

[30] J. Bi, M. Zhang, and L. Zhao, "Security enhancement by detecting network address translation based on instant messaging," in *EUC*. Springer-Verlag, 2006.

[31] SimulatedSimian. (2014) tracetcp: Traceroute utility that uses tcp syn packets to trace network routes. [Online]. Available: https://simulatedsimian.github.io/tracetcp.html

[32] K. Borders and A. Prakash, "Web tap: Detecting covert web traffic," in *CCS*. ACM, 2004.

[33] MaxMind, Inc. (2014) Device tracking add-on for minfraud and proxy detection services. [Online]. Available: http://dev.maxmind.com/minfraud/device/

[34] Qualys, Inc. (2015) HTTP client fingerprinting using SSL handshake analysis. [Online]. Available: https://www.ssllabs.com/projects/client-fingerprinting/

[35] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *SP*. IEEE Computer Society, 2013.

[36] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," RFC 4301, Dec. 2005.