Behavioral Authentication of Server Flows

James P. Early[1] Carla E. Brodley[2] Catherine Rosenberg[2]

[1]CERIAS
Purdue University
West Lafayette IN 47907-2086
earlyjp@cerias.purdue.edu

[2]School of Electrical and Computer Engineering
Purdue University
West Lafayette, Indiana 47907-2035
{brodley, cath}@ecn.purdue.edu

Abstract

Understanding the nature of the information flowing into and out of a system or network is fundamental to determining if there is adherence to a usage policy. Traditional methods of determining traffic type rely on the port label carried in the packet header. This method can fail, however, in the presence of proxy servers that re-map port numbers or host services that have been compromised to act as backdoors or covert channels.

We present an approach to classify server traffic based on decision trees learned during a training phase. The trees are constructed from traffic described using a set of features we designed to capture stream behavior. Because our classification of the traffic type is independent of port label, it provides a more accurate classification in the presence of malicious activity. An empirical evaluation illustrates that models of both aggregate protocol behavior and host-specific protocol behavior obtain classification accuracies ranging from 82-100%.

1. The Need for Authentication of Server Flows

Understanding the nature of the information flowing into and out of a system or network is fundamental to determining if there is adherence to a usage policy. Without this understanding, administrators of systems or networks can not know if information is being compromised, if their system resources are being used appropriately, or if an attacker is using a service for unauthorized access to a host. In this paper we address the problem of server flow authentication – the on-going identification of server type for a stream of network packets. Specifically, we address the question of whether we can correctly identify the TCP application protocol of a flow based on features that measure the behavior of the flow.

The traditional method of determining the client-server protocol is by inspecting the source and destination port numbers in the TCP header. The mappings between port numbers and their corresponding service are well known [2]. For example, HTTP server traffic uses port 80, and SMTP server traffic uses port 25. In essence, we rely on a correct *labeling* of the traffic to accurately determine its type. The binding between the port label and the type of traffic is a *binding by convention*. This label is also used as a basis for firewall filtering and intrusion detection [26, 29].

The problem lies in that there are several different attack scenarios for which the port number may not be indicative of the true nature of the server traffic.

Proxies: These servers are used to consolidate access to a particular service for a group of users. For example, web proxies are used to handle all HTTP client requests to external servers. However, there are also proxies that exist for the specific purpose of evading firewall filtering rules for set of applications [13]. In this case, the proxy takes traffic that would normally be dropped by the firewall and remaps the port numbers to make the traffic appear to be HTTP traffic. Because HTTP traffic is routinely allowed to pass through firewalls, the user of this proxy is able to circumvent the network policy.

Server Backdoors: When a server has been compromised, the attacker often places a "backdoor" in one or more of the running services [7]. The purpose is provide the attacker with a portal that he/she can use to regain access at a later time. Traffic from this portal will have the same port number label as legitimate traffic for the compromised service. The attacker may replace the binary of an authorized service X with a binary that can function as both X and Telnet. When a packet is received from a particular source IP, the rogue server knows to execute Telnet, otherwise it executes service X.

User-Installed Servers: This category includes the installation of unauthorized Telnet, HTTP, or other servers for some illicit purpose. It also represents the increasing numbers of peer-to-peer file sharing networks [20]. These servers are initiated by the user and can be configured to use almost any port. This category also includes the recent appearance of "super worms" - worms that propagate via e-mail and carry their own mail server [12]. Once installed, these worms utilize their rogue mail server to forward unsolicited e-mail messages i.e., Spam. Without prior knowledge of a port to service mapping, the true nature of the traffic cannot be determined.

Each of these scenarios represents an instance where the port number label fails to accurately indicate the type of traffic. Worse yet, it is precisely these scenarios where an accurate identification of the traffic would reveal a compromised service or policy violation. Thus, there exists a need to classify traffic associated with a particular service, what we will henceforth refer to as a *server flow*, using a method other than a mere label that is easily modified, ambiguous, or conceals unauthorized activity.

Significant effort has been invested in the design of tools for detecting the presence of unauthorized services on a host. These range from file system integrity tools that detect modification to server application files (e.g., Tripwire [15]) to tools that look for artifacts of successful intrusions (e.g., ChkRootKit [22]). Successful use of these tools requires proper configuration and, in some cases, a suspicion that an attack has occurred. But, *the fact that a machine running these tools was compromised casts doubt on the information these tools report.* For example, if a Linux system has been root-kitted by a Loadable Kernel Module (LKM) [21], it may be impossible to detect this from *inside* the compromised host [32]. This raises the distinct possibility that an unauthorized service can go undetected indefinitely.

For situations where we cannot trust the results from a compromised system, or the operator is unaware of a successful attack, it would be beneficial to have an external auditor for the purpose of ensuring proper server operation and/or detecting unauthorized services. The identification method used by this auditor should eschew port number labels. Rather, the identification should be indicative of the proper *behavior* of a given server flow.

In this paper, we investigate how server flows can be classified based on their behavior. The result is a system that monitors network traffic to check conformity with expected network services and to detect service anomalies. The remainder of this paper is organized as follows. Section 2 investigates whether behavioral characteristics of server flows can be measured. Section 3 discusses how features measuring these characteristics can be used for server identification. Section 4 presents an empirical evaluation that illustrates that we can discriminate among servers based on characteristics of their flow behavior. Section 5 discusses how our classification method can be integrated with network intrusion detection systems. Methods an attacker might use to subvert our classification system are presented in Section 6. Related work is presented in Section 7. Finally, conclusions and future work are discussed in Section 8.

2. Understanding the Nature of Server Flows

The key issue in the behavioral authentication of server flows is what characteristics or *features* of the traffic should be monitored. In environments where there are concerns about user privacy, or where encryption is used to hide the data carried in network packets, we cannot rely on the contents of the payload as a source of features. Rather, we examine the packet header and the operational characteristics of the traffic itself to define our feature set.

For the purposes of our analysis and experiments, we focused on the HTTP, FTP, Telnet, SMTP, and SSH application protocols. These protocols are well understood, stable, widely implemented, and represent the vast majority of user traffic [14].

Based on our initial observations, we concluded that features based on the TCP state flags (URG - Urgent, ACK - Acknowledgment, PSH - Push, RST - Reset, SYN - Synchronize, and FIN - Finish) [3] can operationally differentiate server flow behavior. For example, HTTP traffic generally contains far fewer packets with the PSH flag than does Telnet traffic. Specifically, for each of the flags, we calculate the percentage of packets in a window of size n packets with that flag set. In addition to these six features we calculate the mean inter-arrival time and the mean packet length for the window of n packets. During monitoring, these features are used by the classification method to determine whether the previous n packets match the learned behavior of the server flows. In the next section we describe how we form a classifier for server flows.

3. Classification of Server Flows

In this section we describe how we can view behavioral authentication of server flows as a supervised machine learning problem. In supervised learning, the learner is given a set of observations each labeled as one of k classes. The learner's task is to form a classifier from the *training set* that can be used to classify previously unseen (and unlabeled) observations as one of the k classes. A criticism of many anomaly detection systems based on data mining/machine learning is that they assume that they are dealing with a supervised learning problem. That is, the learner will be given examples of both normal and attack

data [5]. It is unrealistic to think that one will receive labeled attack data for a particular host because the act of generating labeled data is human intensive. In such cases, one applies unsupervised learning to form a model of expected behaviors. During monitoring one looks for anomalies with respect to the learned model.

However, server authentication can be naturally cast as either a supervised learning task or an anomaly detection task. To cast the problem as a supervised learning problem we must choose k possible server applications, collect training data for each, and then apply a supervised learning algorithm to form a classifier. Given a new server flow we can then classify it as one of these k types of servers. To cast the problem as an anomaly detection problem we look at each service individually. For each of the k server applications of interest we form a model of normal behavior. Given a new server flow, we compare the new flow to each of the models to determine whether it conforms to any of these models. Casting the problem as an anomaly detection problem uses the same framework as user behavioral authentication [16, 17]. In user authentication the goal is to identify whether the user is behaving normally with respect to a learned profile of behavior.

In this paper we have chosen to investigate server flow authentication based on the supervised learning framework, because we assume a policy exists specifying the services that are to be run on a given host. A drawback of this assumption is that if an attacker replaces or alters an existing service it may not behave like any of the permitted services, and this may not be readily detectable. However, it is unlikely that it will behave *identically* to any of the permitted services, but we plan to examine this conjecture in future work.

4. An Empirical Evaluation

Our experiments are designed to investigate whether we can classify server flows based on features of behavior. We first describe the data used in the experiments and the supervised learning algorithm we chose. We then present experimental results with learning aggregate flows and by-host flows using both synthetic and real network traffic.

4.1. Data Sources

The first data set chosen for our experiments is the 1999 MIT Lincoln Labs Intrusion Detection Evaluation Datasets [1]. Although created for a specific evaluation exercise, these datasets have subsequently been widely used for research into other later intrusion detection systems not part of the original evaluation [18, 19, 36].

The data represent five weeks of simulated network traffic from a fictional Air Force base. Weeks one through three constitute the *training* data used by anomaly-based intrusion detection systems to model behavior. The data in week one and week three are attack-free. There are five network trace files for each week – one for each business day representing network usage from approximately 8:00 AM to 5:00 PM. Each file is in libpcap format (readable with tcpdump), then compressed using gzip. On average, each week consists of roughly 1 GB of compressed data representing 22 million network packets. We used data from week one in our training sets and data from week three in our test sets. Note that we do not use the attack data, since our purpose is to evaluate whether we can classify server behavior – not whether we can detect intrusions. Our assumption is that the intrusion has already occurred and that an attacker has implemented one of the scenarios described in Section 1.

In addition to the Lincoln Labs data, we include experiments using data obtained from our own network. The purpose here is to test the applicability of our method on "real world" network traffic. In particular, we are interested in classifying traffic from some of the newer peer-to-peer file sharing protocols – something that the Lincoln Labs data sets do not contain. Some concerns have been raised about the artificial nature of the Lincoln Labs data [4], and thus an additional objective was to identify any marked differences between experiments with these two data sets.

4.2. Decision Tree Classifier

We chose to use decision trees because they provide a comprehensible representation of their classification decisions. Although techniques such as boosting [11, 30] or support vector machines [6] might obtain slightly higher classification accuracy, they require more computation during classification and further they obscure the decision making process.

A decision tree is a tree structure where each internal node denotes a test on a feature, each branch indicates an outcome of the test, and the leaf nodes represent class labels. An example decision tree is shown in Figure 1. To classify an observation, the *root* node tests the the value of feature *A*. If the outcome is greater than some value *x*, the observation is given a label of *Class 1*. If not, we descend the right subtree and test the value for feature *B*. Tests continue until a leaf node is reached. The label at the leaf node provides the class label for that observation.

We chose to use the C5.0 decision tree algorithm [27] – a widely used and tested implementation. For details regarding the specifics of C5.0 the reader is referred to [27, 28]. Here we provide only the key aspects of the algorithm related to decision tree estimation, particularly as it pertains to feature selection. The most important element of the decision tree estimation algorithm is the method used to estimate splits at each internal node of the tree. To do this,

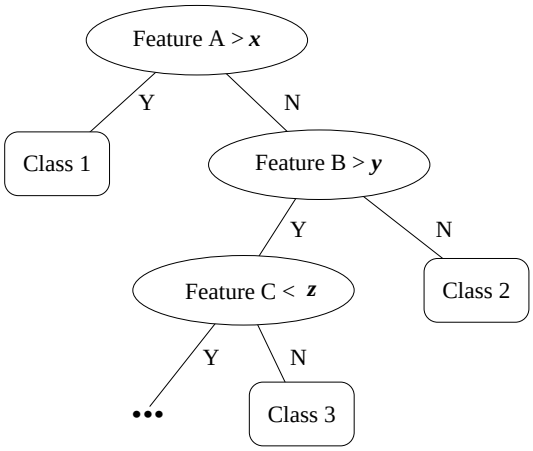

Figure 1. Decision tree abstraction showing how the values associated with certain features determine the class label. In this example, observations whose value for feature *A* is greater than *x* are assigned a class label of *Class 1*. Other classifications are based on the values of features *B* and *C*

```
tcpPerFIN > 0.01:
:...tcpPerPSH <= 0.4: www (45)
:   tcpPerPSH > 0.4:
:   :...tcpPerPSH <= 0.797619: smtp (13)
:       tcpPerPSH > 0.797619: ftp (38)
tcpPerFIN <= 0.01:
:...meanIAT > 546773.2:
    :...tcpPerSYN <= 0.03225806: telnet (6090)
    :   tcpPerSYN > 0.03225806:
    :   :...meanipTLen > 73.33: ftp (21)
    :       meanipTLen <= 73.33:
    :       :...tcpPerPSH > 0.7945206: smtp (8)
```

Figure 2. Portion of a decision tree generated by C5.0.

C5.0 uses a metric called the *information gain ratio* that measures the reduction in entropy in the data produced by a split. In this framework, the test at each node within a tree is selected based on splits of the training data that maximize the reduction in entropy of the descendant nodes. Using this criteria, the training data is recursively split such that the gain ratio is maximized at each node of the tree. This procedure continues until each leaf node contains only examples of a single class or no gain in information is given by further testing. The result is often a very large, complex tree that overfits the training data. If the training data contains errors, then overfitting the tree to the data in this manner can lead to poor performance on unseen data. Therefore, the tree must be pruned back to reduce classification errors when data outside of the training set are to be classified. To address this problem C5.0 uses confidence-based pruning, and details can be found in [27].

When using the decision tree to classify unseen examples, C5.0 supplies both a class label and a confidence value for its prediction. The confidence value is a decimal number ranging from zero to one – one meaning the highest confidence – and it is given for each instance.

4.3. Aggregate Server Flow Model

Our first experiment was designed to determine the extent to which FTP, SSH, Telnet, SMTP, and HTTP traffic can be differentiated using a decision tree classifier. We used the data from week one of the Lincoln Labs data to build our training dataset. The set was created by first randomly selecting fifty server flows for each of the five protocols. Each server flow consists of the packets from a server to a particular client host/port. The largest flow contained roughly 37,000 packets, and the smallest flow contained 5 packets. The 250 flows represented a total of approximately 290,000 packets. We refer to this as an *aggregate model* because the collection of flows came from many different servers.

The fact that this data is certified as attack-free meant that we could have confidence in the port numbers as indicative of the type of traffic. We used the server port to label each of flows in the training set. Each server flow was then used to generate data observations based on our feature set. The result is a data set consisting of approximately 290,000 thousand labeled observations. We repeated this process for each of seven packet window sizes. The window size is an upper bound on the number of packets used to compute the means and percentages. If an individual flow contains fewer packets than the packet window size, the number of available packets is used to calculate each observation.

Each of the seven training sets was then used to build a decision tree using C5.0. We constructed test sets in the same manner – fifty server flows from each protocol were randomly selected from week three of the Lincoln Labs data. These were then passed to our feature extraction algorithm using the same seven window sizes.

Before describing how a tree is used to classify a flow, we give an example of a portion of a decision tree generated by C5.0 in Figure 2. In this example, the root node tests the percentage of packets in the packet window with the FIN flag set (tcpPerFIN). If this percentage exceeds 1%, a test is made on the percentage of packets with the PSH flag set (tcpPerPSH). If this value is less than or equal to 40%, the observation is classified as “www”, indicating HTTP traffic. The numbers in parenthesis indicate the number of train-

Window Size	FTP	SSH	Telnet	SMTP	WWW
1000	100%	88%	94%	82%	100%
500	100%	96%	94%	86%	100%
200	98%	96%	96%	84%	98%
100	100%	96%	96%	86%	100%
50	98%	96%	96%	82%	100%
20	100%	98%	98%	82%	98%
10	100%	100%	100%	82%	98%

Table 1. Classification accuracy of the aggregate model decision trees on unseen individual server flows. Each value represents the percentage of correctly classified flows out of the fifty flows for each protocol

ing observations classified with this leaf node. Other tests can be seen involving the mean inter-arrival time (meanIAT) and mean packet length (meanIPTLen).

During testing, the class label for a given flow was calculated by summing the confidence values for each observation in the flow. The class with the highest total confidence was assigned to that flow. The classification results are shown in Table 1. For each of seven window sizes, we report the percentage of correctly classified server flows out of the set of fifty flows for each protocol. As can be seen in the table, the classification accuracy ranges from 82% to 100%.

In general, the classification accuracy was lower for SMTP server flows than for other protocols. We examined the misclassified flows in more detail and discovered that these flows were generally 2-4 times longer than correctly classified flows. Longer SMTP server flows represented longer periods of interaction, and thus contain increasing numbers of observations classified as Telnet or FTP. In these few cases, our feature set is not adequate for discriminating between the behaviors of these flows.

It is more desirable to use a smaller window size because this decreases the time to detect that a service is behaving abnormally. Indeed for SSH we see that too large a packet window size (1000) hurts classification accuracy. For FTP, SSH and Telnet, a window size as small as ten packets achieves 100% classification accuracy.

Because the proposed method would be used to monitor traffic in real time, we did a rough calculation of classification time. The average length of time used by C5.0 to classify an entire flow was 70mS.[1] Training is done offline so computation time is of lesser importance, but note that the average length of time used by C5.0 to create each decision tree was 22 seconds. Finally, we need to address the storage requirements for maintaining a window of n values to compute the value of each of the features. We can approximate the value created by storing all n values by retaining only the mean for each feature, μ_{F_i} and using the following update rule for each new packet:

$$\frac{(n-1)\mu_{F_i} + new_{F_i}}{n}$$

In future work we will investigate whether this technique significantly degrades performance.

We conclude from our experimental results that the behavior of server flows for the five protocols can be differentiated using a decision tree classifier built on aggregate flows. We will later discuss how this method can be used to compliment an intrusion detection system.

4.4. Host-Specific Models

Our second experiment addresses whether creating models for specific hosts provides better performance than the aggregate model. There are three advantages to using host-specific models:

1. By creating models for individual server flows, we can monitor these flows for changes in behavior.
2. A host-specific model can capture the implementation subtleties of a particular service running on a host. This resolution is missing in the aggregate model consisting of many server flows.
3. The training examples in an aggregate model will be dominated by the server generating the most traffic. This may dilute examples from other servers. The host-specific model solves this problem.

We first identified a set of hosts in the Lincoln Labs data that each ran three or more server protocols. Training data for each host was collected by randomly selecting server flows from week one for each of the protocols running on

1 The hardware platform used for building the decision trees and classifying observations was a 500Mhz Dual Pentium III PC with 772MB of RAM running Red Hat Linux (kernel version 2.4.18).

Host	FTP	SSH	Telnet	SMTP	WWW
172.16.112.100	95%	–	100%	90%	100%
172.16.112.50	92%	100%	84%	100%	–
172.16.113.50	100%	–	100%	100%	–
172.16.114.50	100%	95%	100%	95%	95%
197.218.177.69	100%	–	100%	100%	–

Table 3. Classification accuracy of host model decision trees on unseen server flows. Each row reports the host address and the percentage of correctly classified flows for each protocol. Fields with a "–" indicate there was no traffic of this protocol type for this host.

Host	Training Flows	Test Flows
172.16.112.100	20	20
172.16.112.50	30	25
172.16.113.50	35	23
172.16.114.50	10	20
197.218.177.69	25	35

Table 2. Number of flows used for each protocol in training and test sets for each host model

these hosts. The number of flows used in each model was chosen such that each protocol was represented by the same number of flows. Table 2 lists the number of training and test flows per host.

Based on our results using the aggregate models, we chose a packet window size of 100 for generating observations. The selection was driven by the fact that SMTP accuracy was greatest using this window size with the aggregate models, and other protocol classifications accuracies were between 96% and 100%. We then trained a decision tree for each host that could be used to differentiate the server flows coming from that host. Test data was collected from week three in the same manner as the training data.

The results in Table 3 indicate that, in general, the host specific models achieve approximately the same classification accuracy as the aggregate models. One difference observed is that classification accuracy varies by protocol. For example, the classification accuracy of Telnet flows for host *172.16.112.50* is 84% whereas the classification of Telnet flows in the aggregate models averaged 96.2%. Examination of the packets in the misclassified Telnet flows revealed an interesting phenomenon. We often observed large time gaps between packets. The time gaps indicate lapses in user activity where the Telnet server is not echoing characters or supplying responses to commands. In our framework, a single large gap can radically alter the values for the mean inter-arrival time of packets, thus resulting in misclassification of the subsequent observations. We refer to this as the *Water Cooler Effect* – the user temporarily leaves the interactive session, then resumes a short while later. We are investigating the sensitivity of our classifiers to this effect. One possible solution would be to subdivide flows based on some time gap threshold and use the interactive sub-flows to build our classifiers.

4.5. Models from Real Network Traffic

In this section we present experiments with real network traffic. We collected a number of server flows using the protocols described. We augmented this set to include flows from hosts acting as Kazaa servers. Kazaa [20, 31] is a peer-to-peer file sharing system that is growing in popularity [8, 33]. Peer-to-peer network traffic was not part of the Lincoln Labs dataset.

Our goal was to determine if there was a significant difference in classification accuracy when using synthetic versus real traffic. We observed classification accuracies by protocol ranging from 85% to 100% for both the aggregate and host models. The peer-to-peer traffic was classified correctly for 100% of the unseen flows. This is an especially interesting result because Kazaa flows carry a port label that is user-defined. Thus, we are able to correctly classify peer-to-peer flows behaviorally – without the use of the port number. These results indicate that our classification method is effective for real network traffic. The range of accuracies match those observed with the synthetic data. Thus, we can identify no appreciable difference in the per-flow behavior in the synthetic Lincoln Labs data versus those in real network traffic.

5. Classification for Intrusion and Misuse Detection

The two types of classification models presented here give rise to new functionality in the context of intrusion and misuse detection. Aggregate models try to classify a flow based on the general behavior of many flows of a given type.

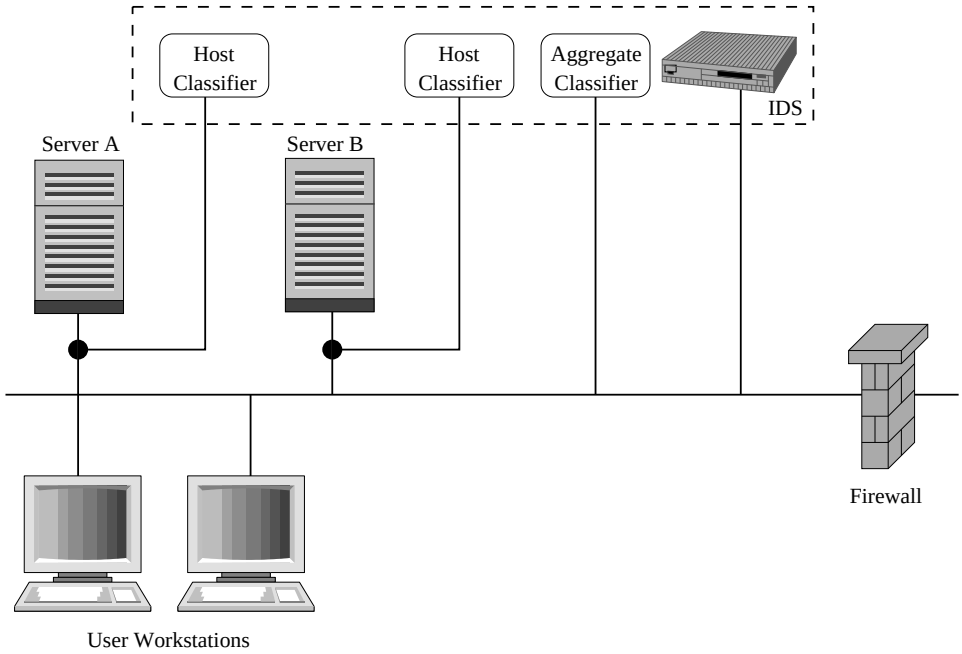

Figure 3. Network placement of the host and aggregate classifiers. Host classifiers monitor specific server flows for deviations from expected behavior. Aggregate classifiers monitor user traffic to determine if flow behavior matches generalized behavior of other flows of the same type.

The question the aggregate model tries to answer is: "What other flows does this flow resemble?" In contrast, host models are based on the previously observed behavior of flows for a specific host. Given an unseen flow, the host models try to answer the question: "Does this flow resemble previous server flows from this host?"

Intrusion/misuse detection systems and firewalls try to identify actions a priori as being harmful to the system or network. An IDS may passively monitor traffic and alert in the presence of some attack condition. Firewalls actively drop network packets that violate some network policy. Our classification method attempts to identify activities indicative of intrusion or misuse *after* such an event has occurred. Working in concert with a priori mechanisms, we can attempt to determine at any moment in time whether there is an impending attack or artifacts of a successful attack.

Figure 3 shows how our classification methods can be integrated into a network with an existing IDS. The organization uses servers to provide network services (internally, externally, or both) to some community of users. Our host-flow classification system monitors the output of these servers directly. The purpose is to determine if currently observed flows continue to behave as expected. If an attacker manages to take control of a particular service, he/she will need to interact with the server in such a way as to exactly match the previous behavior. A trojaned web server that behaves like a Telnet sever when communicating with a select group of host addresses would not match the expected host model, and thus be detected.

The network carries additional user traffic to servers that are external to the organization. This traffic is monitored with the aggregate model. Here, we classify the flow generally and compare this to the port label. Observation of traffic that resembles Telnet to some non-standard server port may be an indication of an installed backdoor. Traffic labeled as web traffic (with a server port of 80) that behaves more like Telnet traffic may indicate the presence of a proxy used to evade firewall rules. A peer-to-peer client operating at some user-defined port may be a violation of the network policy. In each of these cases, the aggregate classifier can indicate if a given flow behaves in a manner consistent with its port label. It may not be necessary to monitor every flow – the system could be configured to randomly select a flow and attempt to classify it. If this flow generally matches a flow that is unusual or undesirable for a port range, it can be identified and investigated.

Our method can operate on its own physical system, or it may be part of the IDS or firewall. This decision will be driven by the number of flows the system will be expected to monitor. Following the construction of the models, the system makes simple and rapid classifications.

6. Subverting Classification

Given the presence of a monitoring system described above, we examined ways in which an attacker could manipulate a session in order to affect classification of a server flow. We've previously seen one such example, albeit an innocuous one, in our discussion of the Water Cooler Effect. Here, the user suspends interaction thus causing variation in the arrival time of packets and hence a potentially large fluctuation in the mean inter-arrival time measured across the packet window for the server flow. An attacker can do the same thing. However, it is not clear that he/she would be able to cause a *particular* classification to be chosen. It is more likely that he/she will alter the observations as to cause some indeterminant class to be chosen. If a host classifier is being used, the deviation from the expected behavior would trigger an alarm.

Another method might involve the use of extraneous TCP flags in packets sent to the server. An example might be the use of the URG flag in HTTP packets. The distribution of TCP flags in the corresponding server flow may or may not be affected, based on the implementation of the server on that host. As with the effects of timing, we are investigating the sensitivity of classifiers to this manipulation in future work.

7. Related Work

Previous work in flow identification has employed a variety of techniques and feature sets. Dunigan and Ostrouchov used Principle Component Analysis (PCA) on two features (packet inter-arrival time and length) to create signatures for a variety of flow types [10]. Their reported classification accuracies are comparable to our method. However, their method requires off-line analysis. In contrast, our method performs classification in real time.

Tan and Collie used a modified neural network built on a single feature (total number of bytes transmitted) [35]. They confined their analysis to the classification of Telnet and FTP protocols. Their reported classification accuracies were generally lower than our method for these protocols.

Daniels [9] reports using a decision tree built with a single feature (first one hundred bytes of a packet) to classify flows. However, classification accuracy was found to be inadequate for practical use.

There are a number of commercial products that attempt to identify flow type [23–25, 34]. These are primarily used in the context of bandwidth allocation. For example, a network administrator creates a policy stating that web traffic must not exceed a certain percentage of total bandwidth and uses one of these products to selectively drop traffic when that policy is violated. Many details of the classification methods used by these products are not publicly available because they are proprietary. Thus, we are unable to compare our method to those used in these products. It is unclear whether any of these products can correctly classify flows in the presence of malicious activity (as described in Section 1). One company, Packeteer [25], reports that their product uses information "from all seven layers of the protocol stack" to create an application signature that is used to classify flows. As stated previously, such a system may or may not be appropriate in an environment where payload encryption is used or where there are concerns about user privacy.

We have also identified a component of the Snort IDS [29] that is used to classify server flows. However, this system relies on the port number and detection of the TCP 3-way handshake. As stated previously, in the presence of a proxy or compromised service, this system is unlikely to classify a flow correctly.

With respect to our feature set, the NATE (Network Analysis of Anomalous Traffic Events) [36] system is also based on TCP flags. NATE uses principle component analysis to identify that the TCP state flags can detect certain types of attacks. Our method differs in two respects. First, the NATE system attempts to model differences between normal traffic and attack traffic. They do not attempt to model differences in behavior between protocols. The second difference involves NATE's use of clustering to identify anomalies. This method must be done off-line, thus limiting the usefulness of the system in environments requiring near real-time detection. In contrast, once a decision tree has been created, our system can monitor packets in real-time.

8. Conclusions

We have presented a novel approach for defining a set of features to model operational behavior of server flow traffic. We demonstrated through the use of the C5.0 decision tree algorithm that our features can differentiate the behavior of server protocols with an accuracy of 82% to 100%. We illustrate empirically that aggregate models can classify an unseen server flow as belonging to a family of previously seen flows, and that host models can determine whether flows from a given server match the behavior of previously seen flows from that server. These classifiers can augment traditional intrusion detection systems to detect artifacts of successful attacks. Our techniques of classification are independent of packet labellings and are thus immune to techniques that modify port numbers to conceal activity.

The decision tree classifiers can be sensitive to fluctuations in the inter-arrival time of packets. This was exemplified in what we call the Water Cooler Effect. We plan to investigate how this sensitivity can be mitigated to increase the classification accuracy for certain protocols.

We intend to further examine the use and augmentation of our feature set to model additional types of server flows. Application of our technique will be expanded to other transport protocols, particularly those used by peer-to-peer file sharing systems. Other feature sets are in development to model UDP traffic, ICMP traffic, and selected routing protocols.

9. Acknowledgments

This research is supported by AFRL grant number F30602-02-2-0217.

References

[1] 1999 darpa intrusion detection evaluation data set. URL `http://www.ll.mit.edu/IST/ideval/data/1999/1999\_data\_index.html`.

[2] RFC 1700: Assigned Port Numbers. URL `ftp://ftp.internic.net/rfc/rfc1700.txt`.

[3] RFC 793 - Transmission Control protocol. URL `ftp://ftp.internic.net/rfc/rfc0793.txt`.

[4] Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000. ISSN 1094-9224.

[5] S. Barbara and S. Jajodia. *Applications of Data Mining in Computer Security*. Kluwer Academic Publishers, 2002.

[6] K. Bennett and C. Campbell. Support Vector Machines: Hype or Hallelujah? *SIGKDD Explorations*, 2:1–13, 2000.

[7] J. Boxmeyer. ONCTec - List of Possible Trojan/Backdoor Port Activity. ONCTek, llc. URL `http://www.onctek.com/trojanports.html`.

[8] Cornell University Student Assembly Committee on Information and Technologies and ResNet. Cornell Internet Usage Statistics. URL `http://www.cit.cornell.edu/computer/students/bandwidth/charts.html`.

[9] Thomas E. Daniels. personal communication, September 2003.

[10] Tom Dunigan and George Ostrouchov. Flow Characterization for Intrusion Detection. Technical report, Oak Ridge National Laboratory, November 2000.

[11] Y. Freund. Boosting a Weak Learning Algorithm by Majority. *Information and Computation*, 121(2):256–285, 1995.

[12] Saul Hansell. E-mail's Backdoor Open to Spammers. *New York Times*, May 20 2003.

[13] iNetPrivacy Software Inc. Antifirewall. URL `http://www.antifirewall.com/intro.htm`.

[14] Internet Technical Resources. Traffic Statistics. URL `http://www.cs.columbia.edu/~hgs/internet/traffic.html`.

[15] G. H. Kim and G. Spafford. The Design and Implementation of Tripwire: A File System Integrity Checker. In *ACM Conference on Computer and Communications Security*, pages 18–29, 1994. URL `citeseer.nj.nec.com/article/kim94design.html`.

[16] T. Lane and C. E. Brodley. Temporal Sequence Learning and Data Reduction for Anomaly Detection. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 150–158. Association for Computing Machinery, Nov 1998.

[17] T. Lane and C. E. Brodley. Temporal Sequence Learning and Data Reduction for Anomaly Detection. *ACM Transactions on Computer Security*, 2(3):295–331, 1999.

[18] W. Lee and S. Stolfo. A Framework for Constructing Features and Models for Intrusion Detection Systems. *ACM Transactions on Information and System Security*, 3(4):227–261, November 2000.

[19] M. Mahoney and P. K Chan. PHAD: Packet Header Anomaly Detection for Indentifying Hostile Network Traffic. Technical Report CS-2001-4, Florida Tech, 2001.

[20] Evangelos P. Markatos. Tracing a Large-scale Peer to Peer System : an Hour in the Life of Gnutella. Technical Report 298, 2001. URL `citeseer.nj.nec.com/markatos01tracing.html`.

[21] T. Miller. Detecting Loadable Kernel Modules (LKM). URL `http://www.incident-response.org/LKM.htm`.

[22] N. Murilo and K. Steding-Jessen. chkrootkit: A Tool that Locally Checks for Signs of a Rootkit. URL `http://www.chkrootkit.org/`.

[23] NetScreen Technologies, Inc. NetScreen-5000 Series. URL `http://www.netscreen.com/products/datasheets/ds_ns_5000.jsp`.

[24] Captus Networks. Captus ips 4000 series. URL `http://www.captusnetworks.com/`.

[25] Packeteer, Inc. Packeteer PacketShaper. URL http://www.packeteer.com/resources/prod-sol/PSDS.pdf.

[26] P. A. Porras and P. G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proc. 20th NIST-NCSC National Information Systems Security Conference*, pages 353–365, 1997. URL http://citeseer.nj.nec.com/porras97emerald.html.

[27] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[28] Ross Quinlan. Data Mining Tools See5 and C5.0. URL http://www.rulequest.com/see5-info.html.

[29] M. Roesch and C. Green. Snort - The Open Source Network Intrusion Detection System. URL http://www.snort.org/.

[30] Robert E. Schapire. A Brief Introduction to Boosting. In *IJCAI*, pages 1401–1406, 1999. URL citeseer.nj.nec.com/schapire99brief.html.

[31] Sharman Networks Ltd. . Kazaa Media. URL http://www.kazaa.com/us/.

[32] E. Skoudis. *Counter Hack: A Step-by-Step Guide to Computer Attacks and Effective Defenses*. Prentice Hall, 2002.

[33] J. St. Sauver. Percentage of Total Internet2 Traffic Consisting of Kazaa/Morpheus/FastTrack - University of Oregon. In *Collaborative Computing in Higher Education: Peer-to-Peer and Beyond Workshop*. URL http://darkwing.uoregon.edu/~joe/kazaa.html.

[34] Stampede Technologies, Inc. Turbogold enterprise edition. URL http://www.stampede.com/productsclienttoserverfeaturesTraffic.html.

[35] K. M. C. Tan and B. S. Collie. Detection and Classification of TCP/IP Network Services. In *Proceedings of the Thirteenth Annual Computer Security Applications Conference*, pages 99–107, San Diego, California, December 1997.

[36] Carol Taylor and Jim Alves-Foss. NATE: Network Analysis of Anomalous Traffic Events, a low-cost approach. In *Proceedings of the 2001 Workshop on New Security Paradigms*, pages 89–96. ACM Press, 2001. ISBN 1-58113-457-6.