Anomaly Detection Amidst Constant Anomalies: Training IDS On Constantly Attacked Data

M. Patrick Collins and Michael K. Reiter

April 9, 2008
CMU-CyLab-08-006

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Anomaly Detection Amidst Constant Anomalies: Training IDS On Constantly Attacked Data

M. Patrick Collins and Michael K. Reiter

Abstract

Automated attack tools and the presence of a large number of untrained script kiddies has led to popular protocols such as SSH being constantly attacked by clumsy high-failure scans and bot harvesting attempts. These constant attacks result in a dearth of clean, attack-free network traffic logs, making training anomaly detectors for these protocols prohibitively difficult. We introduce a new filtering technique that we term *attack reduction*; attack reduction reduces the impact of these high-failure attacks on the traffic logs and can be used to extract a statistical model of normal activity without relying on prior assumptions about the volume of normal traffic. We demonstrate that a simple anomaly detection system (counting the number of hosts using SSH) trained on unfiltered data from our monitored network would fail to detect an attack involving 91,000 hosts; in contrast, it can be calibrated to detect attacks involving as few as 370 hosts using our attack reduction methodology. In addition, by using the same statistical model we use for filtering attacks, we estimate the required training time for an IDS and demonstrate that the system will be viable in as little as five hours.

1 Introduction

While scanners and bot harvesters can evade detection by limiting their behavior [17], sometimes they don't have to bother with subtlety. Automated tools used to exploit vulnerabilities on well-known applications are widely disseminated, resulting in a constant stream of crude scanning and infiltration attempts that are largely aimed at nonexistent targets. The presence of these constant clumsy attacks means that anomaly detection systems face a bootstrap problem: before they can build an accurate model of normalcy[1], they need an effective means to filter out hostility.

In this paper, we demonstrate a method for recovering the normal activity of a network in the presence of constant attacks. We use this method of *attack reduction* to eliminate the noise caused by common high-failure attacks, and are able to produce a model of normal system activity that is far more precise than one derived from raw data. To illustrate the utility of our approach, we show that a graph-based anomaly detection method that counts the number of hosts observed over time [4], when trained on raw SSH data observed on a network we monitor, resulted in an anomaly detection threshold of 91,000 hosts in a system with 15,000 known servers. As such, without applying our techniques, the anomaly detection system would fail to detect, say, a distributed denial-of-service from an entire /16. Using our attack reduction approach, the model of normalcy can be tuned far more precisely, to reduce the detection threshold to only 370 hosts.

[1]Throughout this paper, by "normal" behavior we mean behavior that is both normal and legitimate. That is, we do not intend to include these constant, clumsy attacks in what we describe as the "normal" traffic, and in fact the point of this paper is to eliminate these simplistic attacks to create a model of normal and legitimate traffic.

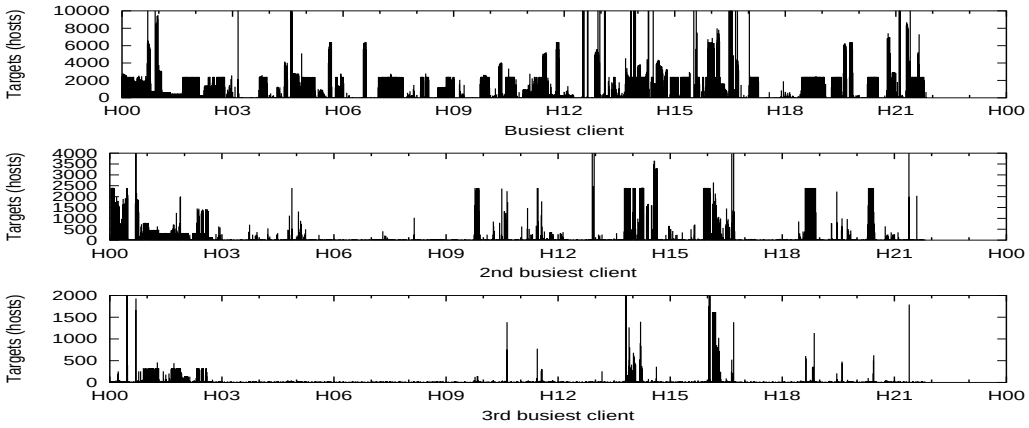


Figure 1: Communication of the three highest-degree SSH clients per thirty second period on the monitored network. As the figure shows, multiple clients appear to be communicating with hundreds or thousands of servers at a time, a likely indicator of scanning or failed harvesting.

Our method combines statistical and heuristic filtering to recover periods of normal activity. This approach leads to two distinct advantages outside of support for the statistical learning system we have developed. First, since the attack reduction methodology identifies periods of normal activity, traffic records during those normal periods can be used to train other forms of IDS, such as Jung *et al.*'s TRW [11]. In addition, using our statistical approach, we are able to establish the training time required by our anomaly detection system and other approaches that rely on parametrized distributions. In our case, we demonstrate that we expect to achieve a 95% accuracy within 5 hours of wall clock time, including attacks.

To illustrate the noisy and clumsy attacks that we seek to filter out, Figure 1 plots the degree (number of servers contacted) for the first, second and third-highest degree SSH clients observed on our monitored network in 30s periods beginning on September 10th, 2007. As this figure shows, clients regularly contact over 2,000 targets per period during this time; in the most extreme cases, an individual client will contact over 250,000 distinct targets. The feasibility of a single client opening thousands of simultaneous SSH connections in half a minute notwithstanding, the observed network has less than 15,000 SSH servers.[2]

These results are consistent over the data examined for this paper: a two week log of SSH traffic in which we are reasonably confident that some form of attack occurs in over 80% of the 30s periods observed. The majority of these attacks are scans and automated botnet harvesting attempts, however since the failure rate for these attacks is so inordinately high, we cannot actually determine what an attacker was doing in the majority of cases. We will refer to these attacks as *High Connection Failure Attacks* (HCFAs) to emphasize this.

[2]Servers were identified by looking for hosts that had payload-bearing responses from port 22 during the observation period.

The key properties of HCFAs, as seen in the SSH data, is that they are persistent, continuous, massive and almost entirely failures. SSH traffic has been persistently scanned for several years by script kiddies and bots, generally by using brute-force password attacks[3]. The level of hostile traffic is so pervasive that various security publications recommend moving the SSH listener port in order to evade scanning[4].

Anomaly detection systems generally classify attacks as deviations from a model of normal behavior generated from historical data [5, 4, 24]. While anomaly detectors have the potential to detect subtle attacks quickly and effectively, these systems must train on clean data in order to produce an accurate model [8]. As shown above, however, for protocols such as SSH, clean data is rare – HCFAs are, on a flow by flow basis, the most common form of traffic. If an anomaly detection system trains on such data, it will correctly and uselessly model normal activity with the HCFAs. A statistical model based on this traffic will have an inordinately wide variance that will allow subtle attacks to hide in the noise. Raising alerts on HCFAs is also counterproductive – they are so common and so rarely successful, that validating the traffic will distract operators from far more serious threats, if those threats are even noticed.

Our attack reduction process assumes that the IDS models normal network activity using a parametrized distribution. In this paper we use the Gaussian distribution $\mathcal{N}(\mu_g, \sigma_g)$ as the model, which we will refer to as the *state model* of the system. To extract this state model from the traffic, attack reduction utilizes three steps. The first, *log-level* filtering, examines records for evidence of *significance*. A significant record is one which may have been received and processed by the targeted application. The majority of HCFA flows will be insignificant since they will communicate with nonexistent targets. This process ensures that the filtered data set is manageably small without eliminating important normal interactions, but the simple heuristics used by the log-level filter mean that it cannot filter all HCFAs.

We therefore complement log-level filtering with a second, *state-level* filtering method. In this work, we use a filter based on the Shapiro-Wilk normality test to identify and eliminate outlier states [22]. The Shapiro-Wilk test statistic is an estimate of how closely a set of observations matches a Gaussian distribution. State-level filtering assumes that if an HCFA's traffic records pass log-level filtering, the HCFA's aggregate behavior will still be an observable outlier. More specifically, an HCFA will only be able to increase the observed state and it will do so in an obvious manner, and the Shapiro-Wilk based filter will identify this deviation.

The third and final step of attack reduction is *impact evaluation*. The previous filtering steps identify outliers in their datasets and pass sources of those outliers, *anomalous addresses*, to an impact evaluator to determine if the addresses have done anything to warrant further investigation. Since the majority of HCFAs are failures, the evaluation step differentiates between *actionable* and *inactionable* activities. An actionable address is one that has engaged in action that merits operator response, such as successfully communicating with a target; inactionable addresses have done nothing to merit further investigation yet.

This approach will therefore be applicable for the constant HCFAs seen in our SSH data. While the majority of HCFAs are scans and failed harvesting, any attack where the attacker is willing to generate a large number of failed TCP connections will be filtered by this model.

[3] "Protecting Linux against automated attackers", Ryan Twomey, `http://www.linux.com/articles/48138`, last fetched January 18th, 2008.

[4] "Analyzing Malicious SSH Login Attempts", Christian Seifert, `http://www.securityfocus.com/infocus/1876`, last fetched January 18th, 2008.

The remainder of this paper is structured as follows: §2 is a survey of previous work in anomaly detection and the problem of error reduction in IDS. §3 describes the architecture of our system, the source data and the problems associated with log-level filtering. §4 describes our methods for state-level filtering. §5 examines the impact of our methods on SSH data. §6 discusses impact evaluation and notification, while §7 discusses methods that an attacker can use to evade detection or control the system. Finally, §8 concludes our work.

2 Related Work

While our work is within the domain of anomaly detection, we emphasize that the focus of this paper is on IDS training and use in the presence of persistent interference from HCFAs. Since HCFAs are likely to be scans or failed harvesting attempts, our work is strongly influenced by existing anomaly and scan detection methods, with the notable difference that we assume attacks are the norm within our data.

The pollution from HCFAs has been studied by multiple researchers. Moore *et al.* [16] developed the standard method for examining aggregate behavior by studying *dark space*. Pang *et al.* [18] have discussed the characteristics of the random traffic on the Internet as "background radiation". More directly relevant to our work is the approach of Garg *et al.* [6] for managing the data from this background radiation in NIDS. Our approach differs from that of Garg *et al.* both in application (our method is anomaly-based, whereas theirs is payload-dependent) and implementation, as their work is a hardware-driven solution and ours is more general.

The problem addressed by eliminating HCFAs is best understood by addressing the types of learning done by anomaly detection systems. For the purposes of this paper, we break down these systems into three broad categories: *threshold* systems, which include GrIDS [24] and our graph-based method [4]; *data mining* systems, such as MINDS [5] and LERAD [15]; and *map-based* systems, most notably TRW [11]. Each of these systems relies on past history and is consequently vulnerable to HCFAs.

Threshold systems detect anomalous behavior by looking for a measurable change in traffic levels. Examples of these systems include the original IDES [14] and EMERALD [20], and systems for identifying and quarantining worms [27, 10, 29]. These systems all identify anomalies by looking for a change in behavior that exceeds a threshold estimate. This threshold may be generated by an automatic process, as is the case with Soule *et al.* [23], or provided by a human expert, as is the case with GRiDS [24] and MISSILE [7]. When these thresholds are set automatically (a process we seek to enable here), such systems have assumed that attacks are rare (often implicitly). Here we focus on situations in which this is not the case.

Data mining systems apply undirected learning and clustering techniques to traffic data [1, 5, 15, 12, 9, 26]. These systems group together traffic by log features in order to differentiate attacks from normal activity. As before, they expect that traffic will generally be attack-free (in particular, LERAD [15] is specifically designed to detect "rare time series events"). An exception to this is Barbara *et al.* [2] who address the problem of bootstrapping a data-mining IDS. However, the data set they use, the 1999 Lincoln Labs data [13] is synthetic and does not contain attacks with remotely the frequency of our SSH data.

Mapping systems rely on an inventory of network hosts to identify aberrations. The most well-known of these systems is Jung *et al.*'s TRW method of scan detection [11] and Shankar and Paxson's Active Mapping tool for Bro [21]. These systems have a varying degree of vulnerability

to HCFAs based primarily on the mapping method used. Active approaches will not be affected by HCFAs but on larger networks, passive mapping approaches become increasingly xattractive [28]. Since passive mapping systems must also deal with HCFAs communicating with nonexistent hosts, HCFAs can become a significant challenge.

Our system is not intended to supplant these IDS but rather supports them by providing a method to identify normal activity. Consequently, the results generated by our attack reduction methodology can also be used as a source of labeled normal data for data mining systems or to build the maps used by mapping systems.

3 System Architecture and Log-level Filtering

The majority of connections in HCFAs fail to reach real targets and consequently result in a large number of flow records that are easily distinguished from normal traffic. Log-level filtering is the process by which these *insignificant* flows are distinguished from *significant* flows which may have actually communicated with a server process. Log-level filtering is stateless and inexpensive in order to manage the expected volume of flow records.

This section describes the system architecture and the process of log-level filtering. It is divided into three sections: §3.1 describes our source data and the notation we use to describe it. §3.2 provides an information flow diagram for the system and describes the expected format of log data and state data. §3.3 describes the log filtering process and shows the impact of log-level filtering on representative SSH data.

3.1 Source Data

The source data used in this paper consists of NetFlow V5 records[5] generated by internal routers in a large (in excess of 16 million distinct IP addresses) network. We use a two-week NetFlow trace of SSH traffic collected over this network during the period of September 10th-24th, 2007. The routers record both internal and cross-border traffic.

NetFlow records approximate TCP sessions by grouping packets into *flows*, sequences of identically addressed packets that occur within a timeout of each other [3]. NetFlow records contain size and timing information, but no payload. We treat NetFlow records as tuples of the form (clntip, srvip, direction, stime, bytes, packets, syn, ack). These elements are a modified subset of the fields available in CISCO NetFlow. When referring to a single flow record λ, we will reference constituent elements of a single flow record via "dot" notation (e.g., λ.srvip or λ.packets).

We classify traffic by port number; any flow which is sent to or from TCP port 22 is labeled an SSH flow. Because we examine only SSH traffic, port numbers are not germane to our analyses. Instead, we use the port number to define the srvip, clntip and direction fields. srvip is the address that uses port 22, clntip is the address which used an ephemeral port[6], and direction indicates which direction the recorded traffic flowed (*i.e.*, traffic in the flow was from srvip to clntip or vice versa).

syn and ack refer to the TCP SYN and ACK flags respectively. CISCO's NetFlow implementation contains a field providing a logical OR of all TCP flags observed in a TCP flow. However, this field is not available under all IOS implementations. We therefore limit the use of flag data

[5]CISCO Systems, "CISCO IOS Netflow Datasheet", `http://www.cisco.com/en/US/products/ps6601/products_data_sheet0900aecd80173f71.html`, last fetched October 8th, 2007.

[6]We constrain the flows used in this paper to flows which used an ephemeral port between 1024 and 5000.

to exploratory analysis and scan validation; the *implementations* of log- and state-level filtering in this paper do not use these fields.

The source data for state filters are *observations* constructed from flow records that have already gone through the log filtering process. We group flow records into distinct *logs* $\Lambda = \{\lambda_1 \dots \lambda_n\}$. The contents of any log are determined by the stime field. In this paper, each unique log contains all records whose stime is within a distinct 30s period, and the records within the log are ordered by stime.

The model of normal activity used in this paper is a statistical one based on our previous work on *protocol graphs* [4]. In that paper, we demonstrated that the graph size and largest component size of graph representations of several major protocols could be satisfactorily modeled using a Gaussian distribution. For this paper we rely exclusively on graph size for brevity and generality – graph size is equivalent to the the total host count for a protocol, and consequently the state model does not rely on any special features of our previous work.

An *observation* of the graph size, $g(\Lambda)$, is the graph size for a particular log Λ. Recall that these logs cover unique 30s periods within the source data, so the graph size is the size of the observed protocol graph for SSH within a 30s observation period. $g(\Lambda)$ is defined in Equation 1 as the cardinality of the set of all IP addresses (client and server) in Λ.

$$g(\Lambda) = \left| \left(\bigcup_{\lambda \in \Lambda} \lambda.\mathsf{srvip} \right) \cup \left(\bigcup_{\lambda \in \Lambda} \lambda.\mathsf{clntip} \right) \right| \tag{1}$$

State-level data is a vector of graph sizes $G = (g_1 \dots g_k)$ where each g_i is a $g(\Lambda)$ value. G's observations are ordered by increasing magnitude rather than chronological order, so g_{10} may have been observed some time after g_{11}. The source Λ^{sig} data is comprised of 39,600 distinct logs, each covering a 30s period beginning on September 10th 2007 and continuing until September 24th. 720 logs were discarded from the data set due to sensory or collector failures; if a single log in an hour had an observed failure, we discarded the entire hour. We assume the graph sizes are the result of a random process modeled as $\mathcal{N}(\mu_g, \sigma_g)$ where μ_g is the mean and σ_g is the standard deviation. We will use $\bar{g}$ to refer to the sample mean of G, *i.e.*, $\sum_{g \in G} \frac{g}{|G|}$.

3.2 System Architecture

Figure 2 is an information flow diagram that describes how log and state data are handled by the attack reduction system. This diagram describes four processes: log-level filtering, state-level filtering, normal operation, and impact evaluation/notification. Recall that a single log contains all the flow records observed during a particular 30s period – the information flow in Figure 2 takes place within this time constraint (*i.e.*, all processing decisions take place within the 30s after Λ is reported). Numbers in the following text refer to the corresponding arrows in Figure 2.

The first phase of the process is log-level filtering, during which data are collected from multiple *sensors* located throughout the monitored network. These sensors generate NetFlow records which are then passed (1) to a *log filter* that partitions the Λ into two logs: Λ^{sig}, a log of significant flow records, and Λ^{insig}, a log of insignificant flow records. We approximate significance using the number of packets in a flow, a process described in §3.3. Λ^{insig} is passed (3) to the *log processor*, which produces a list of the busiest hosts for further examination by the evaluator.

After log-level filtering, Λ^{sig} is passed (2) to a *state generator* which realizes the observation $g(\Lambda^{\text{sig}})$. This is then processed either by the state-level filter (4) or normal operations (5). Where

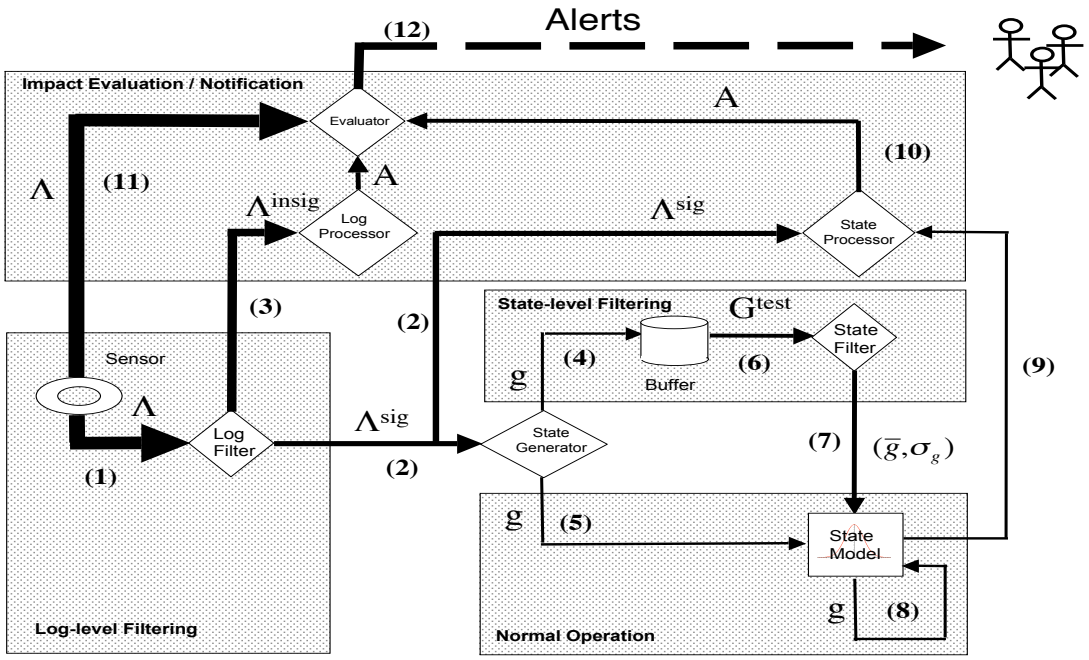


Figure 2: Information flow diagram for the attack reduction system.

observations are processed is a function of whether the system is in bootstrap phase (during which time, the data is processed by the state-level filter) or operating normally.

During the bootstrap phase, the system has insufficient information to judge whether or not any particular observation is an outlier. Consequently, during this phase, the state data is passed to a buffer to create a *test vector*, G^{test}. When $|G^{\text{test}}|$ exceeds a *sample limit*, s, it is run through the state-level filter (6) and outliers are removed. If the filtered sample still exceeds the threshold, then the filter generates parameters for a model of normal traffic: $(\bar{g}, \sigma_g)$. The filter then passes those parameters to the *state model* (7) and the system shifts into normal operations. If the filtered test vector is smaller than the sample limit, additional data is collected (4) and the process repeated (6) until the filtered vector reaches sufficient size.

During normal operation, g observations are passed directly to the state model (5). Recall that the state model used in this paper is a simple statistically derived threshold: $g(\Lambda^{\text{sig}})$ is calculated and compared to the model. If $g(\Lambda^{\text{sig}})$ exceeds the anomaly threshold, then an alert is passed to the *state processor* (9). During normal operation, the state model can update by feeding normal observations back into the model (8).

Anomaly processors examine anomalous addresses to determine whether they are worth deeper investigation by the *evaluator* for validation and notification. The method used by each processor is dependent on the source data (Λ^{sig} or Λ^{insig}). The log processor receives Λ^{insig} (3) data and consequently must process much higher volumes than the state processor. (In our present implementation the log processor simply returns the top-5 highest degree addresses; see Section 6.) The

state processor receives the much smaller Λ^{sig} (2) and will use whatever method is appropriate for its state model. Our implementation uses a graph-based method described in our previous work [4]: it returns a list of those nodes whose removal substantially alters the shape of the graph (see Section 6).

Once each processor generates a list of anomalous addresses, this list is passed to the evaluator. The evaluator examines the anomalous addresses for evidence of *actionability*. We expect the majority of scans to be, if not harmless, then pointless – they include residue from ancient worms (a phenomenon noted by Yegneswaran *et al.* [30]) and blind bot scanning. As such, an anomaly detection system which raises an alarm for every scan will be counterproductive; unless a HCFA is actionable, it is noted in other collection systems but otherwise ignored.

3.3 Log-level Filtering

During log-level filtering, we apply simple heuristics to individual flow records in order to identify the majority of HCFA traffic and remove it from further analysis. The criterion for keeping or removing a flow record is a property we term *significance*. A flow record is significant if the traffic it describes constitutes communication with the targeted host's process rather than just the TCP stack; *e.g.*, an SSH flow record is significant if the target's SSH process receives payload. Otherwise, the flow record is insignificant.

We approximate significance by using properties of the TCP stack. Since TCP implements a state machine on top of packet switching protocols, a correct TCP session requires 3 packets of synchronization in addition to any packets for payload [25]. Flows which contain three packets or fewer are likely to be insignificant. While it is possible to have a significant flow below this limit (such as keep-alives), we expect that they will represent a small proportion of the total number of short flows. Of more concern is that, due to the timeout-based aggregation mechanism used by NetFlow, multiple SYN packets to the same host will be grouped together as a single flow.

Potential alternative significance criteria include TCP flag combinations and payload. We have elected not to use flags because scanning applications such as `nmap`[7] specifically include packet-crafting options that use eccentric flag combinations. We elect not to use payload because the prevalence of SYN packets also makes estimating the actual payload of a flow problematic: SYN packets usually contain nontrivial stack-dependent TCP options [19], meaning that a "zero payload" TCP flow may contain 12 bytes of payload per SYN packet. Alternatively, a flow record describing the ACK packets from a host receiving a file may have no recorded payload, but is still significant. Payload estimates are used for impact evaluation in §6, but at that time we are specifically looking for activity initiated by a single client.

We therefore label a flow λ as significant if $\lambda.\mathsf{packets} > 3$ and insignificant otherwise. Table 1 is a contingency table showing the breakdown of our source data traffic using this rule. As this table shows, the total data describes approximately 2.4 TB of traffic contained in approximately 230 million records, and that approximately 87% of the observed flows are insignificant but make up less than 8% of the observed traffic by volume.

Examining those sensors that provide flag data indicates that approximately 70% of insignificant flows have no ACK flag – an indication that the flow was part of a HCFA. Of the remaining insignificant flows, 21% have SYN, ACK and FIN flags raised. While this type of traffic is potentially

[7] `http://www.insecure.org`

		λ.ack		
		0	1	Total
λ.packets	1-3	1.592e+08 (1.372e+10)	3.750e+08 (4.840e+09)	1.97e+08 (1.856e+11)
	4-∞	1.153e+07 (8.358e+11)	1.863e+07 (1.530e+12)	3.016e+07 (2.370e+12)
Total		1.707e+08 (8.495e+11)	5.614e+07 (1.536e+12)	2.269e+08 (2.388e+12)

Table 1: Breakdown of activity in recorded data set by flows and bytes (bytes in parentheses); source data is all SSH traffic from September 10th to 24th, 2007.

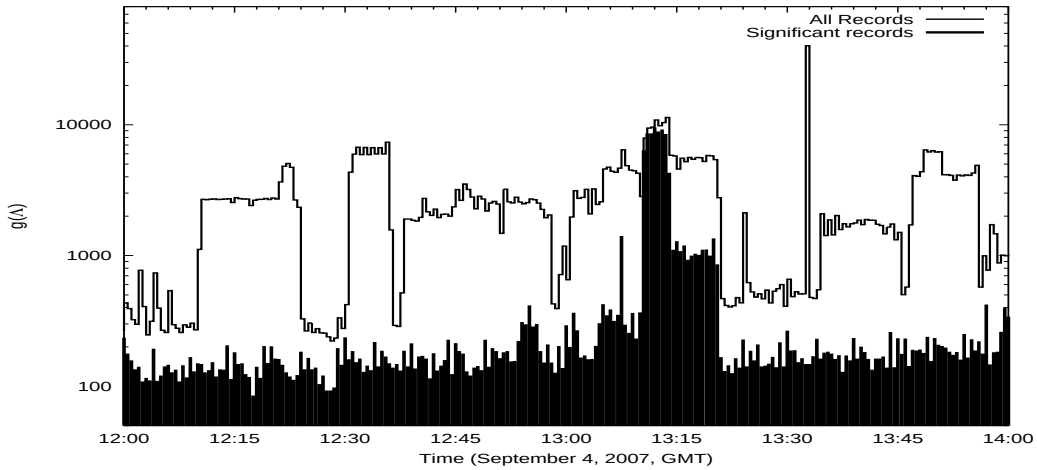


Figure 3: Contribution of significant and insignificant flows to traffic.

normal, it is also characteristic of certain forms of SYN scans[8].

From these results, we conclude that insignificant flows are dominated by HCFAs. A more stringent criterion for significance, such as a higher number of packets, risks removing more normal traffic from the data.

Figure 3 shows the impact of removing insignificant flows from the data set. This figure plots total graph size for the periods between 12:00 and 14:00 GMT for September 4th, 2007. As this figure shows, even with the removal of insignificant flows, it is likely that scanning still appears in the data.

4 Implementing a State-level Filter

In §3 we demonstrated that log-level filtering was a necessary but insufficient method for recovering a system's normal behavior. While log-level filtering can reduce the amount of data to a manageable level and remove the most obvious effects of HCFAs, aggressive log-level filtering raises a serious risk of eliminating normal traffic.

[8]This particular scan can be implemented using the '-sS' option in nmap.

Recall from §1 that our hypothetical anomaly detection system tracks a single *state variable* (in the case of this paper, observed graph size). In this section we introduce a *state-level filter* which is used during the bootstrap phase of the IDS to identify and eliminate anomalies in training data.

State-level filtering applies the same clumsiness assumption that was used in log-level filtering. Specifically, we assume that the attacker's activities will increase the observed state variable and regularly produce outliers due to the attacker's general lack of knowledge about the observed state variable.

This section describes the construction of a state-level filter that uses the g value discussed before. We emphasize that this approach is not dependent on protocol graphs, only that the state variable have a parametrizable distribution. This section is divided as follows: §4.1 describes the source data, §4.2 process of state filtering, and §4.3 describes how much data is required to build a satisfactory model.

4.1 State Data

Figure 4 plots the frequency of $g(\Lambda^{\text{sig}})$ values for all Λ^{sig} in the source data. As this figure shows, the distribution of $g(\Lambda^{\text{sig}})$ observations has a very long tail, but the majority of the observations (approximately 75%) can be modeled by a Gaussian distribution. This Gaussian fraction of the data is shown in the closeup in Figure 4. We hypothesize that the data set consists of two classes of periods: *normal* and *attack*. During a normal period, $g(\Lambda^{\text{sig}})$ is a function of the true population of the network and can be modeled by a Gaussian distribution. During an attack period, the $g(\Lambda^{\text{sig}})$ observation increases to the impact of HCFAs. Since HCFAs are caused by clumsy attackers, we further expect that the attacker cannot *decrease* $g(\Lambda^{\text{sig}})$, and consequently all observations above a certain threshold are likely to be attacks.

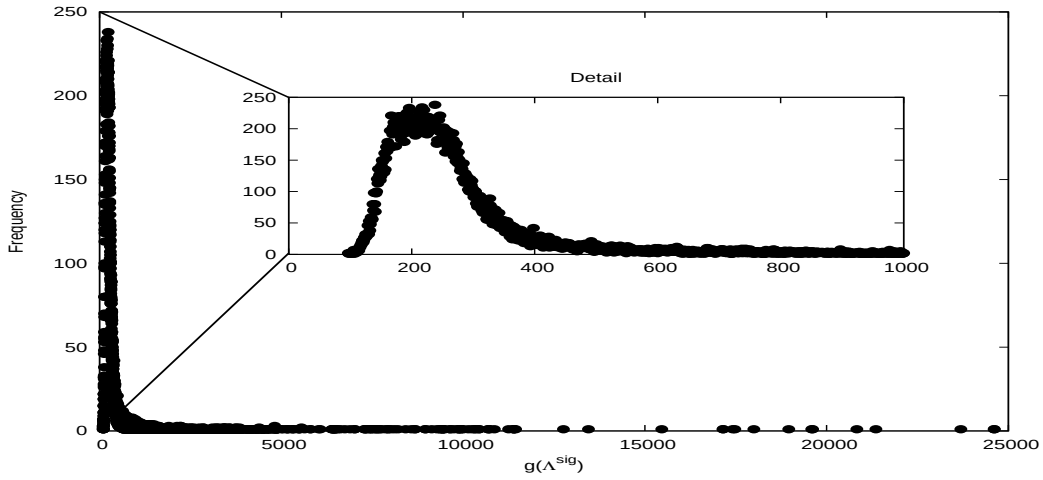


Figure 4: Histogram of total graph size with closeup on normally distributed minimal area; this figure supports our hypothesis that the majority of the high-level traffic are simply HCFAs.

In order to test this, we sampled logs from the upper 40% of the Λ^{sig} set for scanning activity.

Our sample consisted of 60 Λ^{sig} logs chosen randomly, 20 from each of three strata: $300 < g(\Lambda^{\text{sig}}) < 600$, $600 < g(\Lambda^{\text{sig}}) < 5000$ and $g(\Lambda^{\text{sig}}) > 5000$.

Within each log, we examined the traffic for signs of HCFAs such as an unusually high-degree client, or a client which appeared only during the associated phenomenon. Of the 60 logs we examined, a single client dominated each log, in each case communicating with at least 200 distinct targets. While different clients dominated different logs, the set of clients was small: 31 distinct clients out of 60 logs. This result is due to the brazenness of the attackers; in certain cases, an attacker scanned continuously for several hours in a row. Using DNS, we checked client identity and verified that all of the clients were from outside the monitored network. By extracting full logs of the clients' activity over the observation period, we found no examples of payload-bearing interactions from these clients. From these results, we conclude that the clients were engaged in either scanning or the use of automated harvesting tools.

Given these results, we partition G, the vector of graph size observations, into two subsidiary vectors, G^{atk} and G^{normal}. The partitioning criterion is $g(\Lambda^{\text{sig}})$: G^{atk} consists of all $g(\Lambda^{\text{sig}})$ observations where $g(\Lambda^{\text{sig}}) > 300$, G^{normal} consists of everything else. Out of the 39,600 observations in G, 10,101 or approximately 25.5% ended up in G^{atk}. As with G, G^{atk} and G^{normal} are ordered by increasing magnitude. We emphasize that this classification is intended for experimental validation and does not necessarily indicate that these observations actually contain HCFAs. During our experiments, we use the top 3000 values of G^{atk} in order to gain confidence that all of those observations are due to attacks.

4.2 Process

Assume G^{test} is a vector of graph size observations wherein the majority of observations are normal. A state-level filter is a process which examines G^{test} and returns one of two values: either a Gaussian model $\mathcal{N}(\mu_g, \sigma_g)$, or a requirement for further data.

For this paper, we eliminate outliers by using the Shapiro-Wilk test [22]. The Shapiro-Wilk test specifically evaluates the similarity of an ordered vector of observations to a Gaussian distribution. In comparison to other statistical tests, the Shapiro-Wilk test can generate results with a wide variety of sample sizes (between 40 and 2,000 observations) and without advance knowledge of the modeled distribution (*i.e.*, the μ_g and σ_g do not have to be supplied). The statistic generated by the Shapiro-Wilk test (W) is a value in the range of $[0, 1]$, with 1 representing a perfect fit to a Gaussian model.

We calculate the W-statistic, using a vector of observations $G^{\text{test}} = (g_1 \dots g_l)$ ordered by increasing magnitude and an equal size vector M that consists of order statistics for a Gaussian distribution of the form $(m_1 \dots m_l)$. W is then formulated as:

$$W = \frac{\left(\sum_{i=1}^{l} a_i g_i\right)^2}{\sum_{i=1}^{l}(g_i - \overline{g})^2} \tag{2}$$

where

$$(a_1 \dots a_l) = \frac{M^T V^{-1}}{\sqrt{M^T V^{-1} V^{-1} M}} \tag{3}$$

and where V is the covariance matrix of M and $\overline{g}$ is the sample mean of G.

To filter out HCFAs we use a state-level filter shapfilt($G^{\text{test}}, \theta_W$). shapfilt takes two parameters: G^{test} and θ_W, which is the minimal W value we use to determine if a vector contains a Gaussian distribution. Recall that G^{test} is ordered by magnitude and that attackers cannot decrease $g(\Lambda^{\text{sig}})$. We therefore expect that there will be a pivot value below which $g(\Lambda^{\text{sig}})$ observations are normal and above which $g(\Lambda^{\text{sig}})$ observations are attack observations. shapfilt will return the index of that pivot.

shapfilt starts by calculating the W value for the complete G^{test}. If the resulting W exceeds θ_W, then shapfilt terminates and returns $|G^{\text{test}}|$ as its result. Otherwise it removes the largest value and repeats the process with the reduced vector. shapfilt will iteratively test every vector, progressively eliminating the larger observations until $W > \theta_W$ or all the observations are eliminated. The resulting index, s, is then checked against a minimum sample size. If s is too small (*i.e.*, there were too many attacks in the log data), then more observations will be requested and added to G^{test}. If s is sufficiently large, then $\bar{g}$ and σ_g are calculated from the reduced vector and passed to the state model.

4.3 State Filtering: Sample Size

Recall that shapfilt is intended for the initial training and configuration of an IDS. The longer the system remains in training, the more accurate the resulting model. However, while the system is training, it cannot react to attacks.

When shapfilt completes operation on a G^{test} vector, it returns its estimate of how many observations within that vector can be used for training. We can calculate the lower limit for this value by relying on our previous assumption that, in the absence of HCFAs, G^{test} would consist of observations following a Gaussian distribution and calculate the *margin of error*. For our model, the error is the difference between the true mean and the sample mean (*i.e.*, $|\bar{g} - \mu_g|$); the margin of error is the upper bound on that value.

Assume a random process modeled as $\mathcal{N}(\mu_g, \sigma_g)$. We estimate μ_g via the sample mean $\bar{g}$. The margin of error, E, is then written as:

$$E = Z_{\alpha/2}\frac{\sigma_g}{\sqrt{s}} \tag{4}$$

$Z_{\alpha/2}$, the *critical value*, is a factor derived from α, the probability that the error is *greater* than the margin of error. Since the process is modeled with a Gaussian distribution, the critical value is expressed as the number of standard deviations to produce a $(1 - \alpha)$ probability of the error being less than the margin. For example, 68% of all observations within 1 standard deviation of the mean of a Gaussian distribution. In this case, $\alpha = 0.32$ (that is, the probability of the error falling outside this range is 32%), and $Z_{\alpha/2} = 1$.

Given a known standard deviation, σ_g, we can express s as follows:

$$s = \left(\frac{\sigma_g Z_{\alpha/2}}{E}\right)^2 \tag{5}$$

We will limit the margin of error to $0.05\mu_g$ (5% of the true mean). To achieve a 95% degree of confidence in the margin of error, $\alpha = 0.05$ and the corresponding critical value is 1.96. We can then rewrite Equation 5 to estimate that s should be at least:

$$s = 1536 \left(\frac{\sigma_g}{\mu_g} \right)^2 \tag{6}$$

This formula provides us with a method to estimate a lower limit on s, number of samples required. In our previous work [4] we encountered standard deviations which were anywhere from approximately one quarter to one twelfth of the mean. Given this, we will assume that our worst case scenario is where the standard deviation is half the mean. If so, then $s = 1536/4 = 384$. For convenience, we will round this value up to 400 observations, which equates to 3 hours and 20 minutes of 30s samples if there are no HCFAs.

This value assumes that normal traffic can be modeled via a Gaussian distribution. This may result in a longer clock time to gather those 400 observations based on seasonal phenomena, such as the business cycle. For example, in our original work, we divided the traffic into active and fallow periods of 12 hours each. Under this model, each period would require a distinct 400 observations, changing the minimum time required to just under 7 hours. If a different model was calculated for every hour of the day, than the minimum time required would be slightly longer than 3 days.

HCFAs within the data set will also impact the training time. The system will not switch from training mode until it receives a sufficient amount of uncorrupted data, and will reject outlying scans.

5 Evaluation

We now evaluate the ability of state-level filtering to recover an accurate model of normal activity. Our evaluation method uses synthetic G^{test} vectors created by randomly selecting observations from $G^{\text{atk}}$and G^{normal}. Since HCFAs cannot *decrease* graph size, we expect that every G^{test} will have a pivot point: all observations before the pivot point will be normal and all observations after the pivot point will be HCFAs.

We measure the efficacy of the state filtering mechanism by comparing the true pivot point of G^{test} against an estimate generated using the Shapiro-Wilk test. This metric, the *pivot error*, is the difference between the pivot point as estimated by shapfilt and the true pivot point generated when the test vector is constructed. Intuitively, the preferred value for the pivot error is zero, which indicates that shapfilt was able to perfectly separate attack and normal events. However, given the choice of a negative or a positive pivot error, a negative pivot error is tolerable. We expect normal observations will be clustered around the mean, and consequently removing some of them will have minimal effect on the resulting model. However, since HCFAs will be extreme outliers, including one will severely damage the model.

We use pivot error to calculate the efficacy of the W statistic. To do so, we calculate W as a function of pivot error and see how W changes as HCFAs are removed from G^{test}. We ran 500 simulations using synthetic G^{test} vectors generated by randomly selecting 300 observations from G^{normal} and 100 observations from G^{atk}. This vector agrees with our calculated s (400 observations), and matches the observed ratio of attack observations to non-attack observations. Recall that in §4.1 we determined that after log-level filtering, approximately 25.5% of $g(\Lambda^{\text{sig}})$ observations were attacks.

Figure 5 plots the result of our simulation using a boxplot that shows the minimum, first quartile, median, third quartile and maximum observed value for each pivot error. As this figure shows, W jumps significantly as the pivot error approaches zero. Once the scans are removed, the

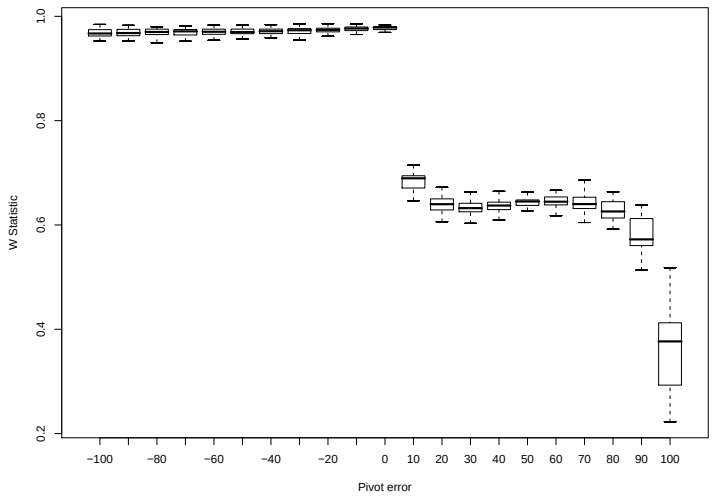


Figure 5: Impact of removing observations using shapfilt; note the decided increase in the W-statistic after removing all HCFAs.

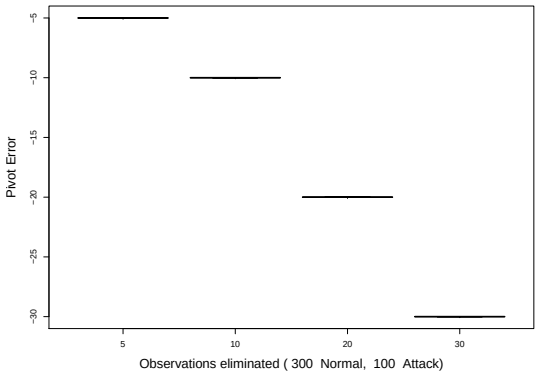
W value remains stable even as we remove more observations from the data set. This result justifies using a relatively stringent θ_W, and so we will set θ_W to 0.95.

Given the stability of W with negative pivot errors, we now address two problems: removing observations after W is above θ_W, and using different G^{test} sizes. Our interest in the former problem is a redundancy and reliability issue. Recall that a positive pivot error is unacceptable while a negative error is tolerable. By removing additional points we guarantee that the resulting vector has a negative pivot error. The second problem is a test of the validity of the estimate we generated in §4.3: if the efficacy of the estimate improves with larger vectors, then the investment in additional learning time may be worthwhile.

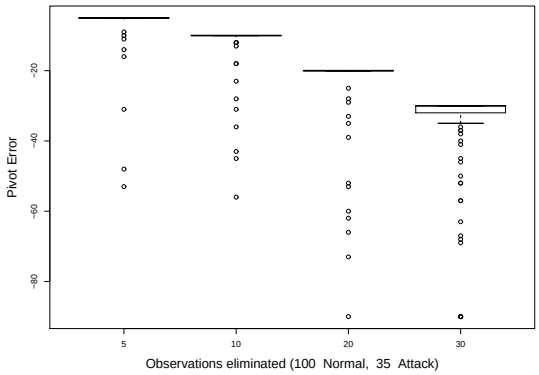
Figure 6 plots the pivot error as a function of the number of observations removed after $W > \theta_W$: 5, 10, 20 and 30 observations respectively. Each plot uses a boxplot with outlier runs represented as single points. Furthermore each figure uses a different total G^{test} size: 400 observations (our standard case) in Figure 6(i), 135 in Figure 6(ii) and 4000 in Figure 6(iii).

Our standard case, Figure 6(i), shows that the pivot error is identical to the number of observations removed after reaching θ_W. For that sample size, there is no reason to remove additional observations. Furthermore, we note that this boxplot has no variation. However, for different vector sizes, we see greater variation, and of different types depending on the sizes.

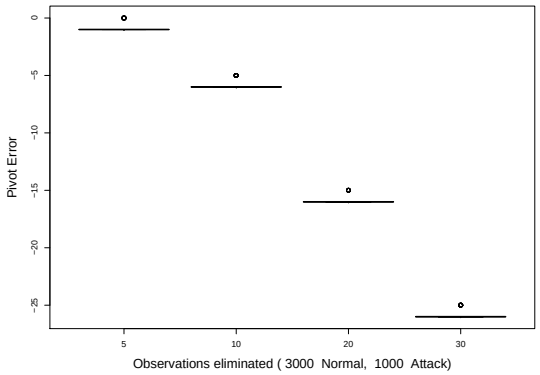
Figure 6(ii) plots the results with 135 observations, a considerably smaller vector size. This figure shows that the value of the pivot error is relatively stable, but that there are a significant number of outliers (approximately 15% of all the runs). More significant, however is that the pivot error was consistently negative even before removing observations, sometimes by up to 20 observations. In these cases, shapfilt is overestimating the pivot. This is likely a result of the small size of G^{test} in these simulation runs. We can compensate for these results by using a less stringent θ_W, but see no compelling reason to do so. We have access to data as long as we are willing to



(i) 300 normal, 100 attack



(ii) 100 normal, 35 attack



(iii) 3000 normal, 1000 attack

Figure 6: Impact of removing nodes after W exceeds θ_W for different sample sizes of equivalent composition.

wait a sufficiently long time, and the savings on training time is minimal compared to the loss in accuracy.

Figure 6(iii) plots the pivot error using a much larger vector: 4000 total observations. In this case, the pivot error is actually slightly higher than the number observations removed (*e.g.*, when five observations are removed, the actual pivot error is -2). As before, the estimates are very stable, with a limited number of outliers that are even higher. This result indicates that, at least for these vector sizes, the number of normal observations reduces the loss in accuracy caused by missing attack observations. However, we also have to note that this is a very large amount of normal observations – 3,000 normal observations are offsetting the effect of around 2 attack observations. Consequently, we conclude that at larger sample sizes, removing additional observations is useful, but not critical for acquiring a more accurate estimate. Furthermore, we note that we have no compelling reason for training the system for forty hours when we have achieved comparable results after three.

Given these results, we are confident that we can train the system with 500 observations in

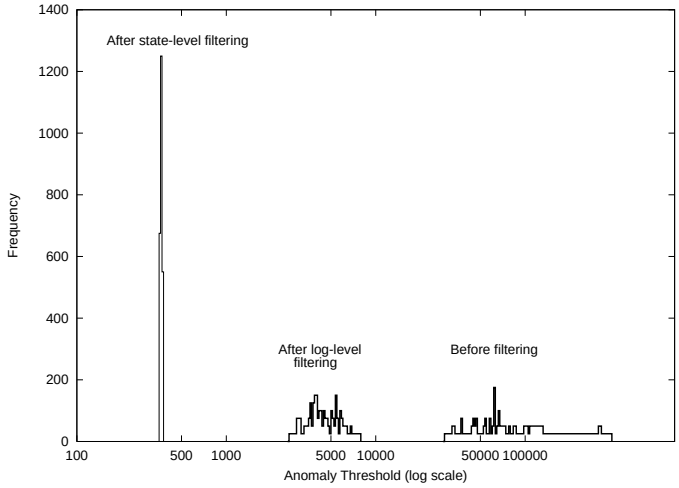


Figure 7: Impact of state-level and log-level filtering on data set.

G^{test}, setting θ_W to 0.95 and without removing any additional observations. We must now evaluate the impact of this filtering method. To do so, we calculate the *anomaly threshold*: $\bar{g} + 3.5\sigma_g$. This value is the expected point at which our IDS, trained on a filtered or unfiltered dataset, would raise an alarm. Figure 7 compares the anomaly thresholds generated using $\bar{g}$ and σ_g values for three different scenarios: raw data (*i.e.*, no filtering at either the log or the state level), after log-level filtering and after state-level filtering. The x-axis on this plot is logarithmically scaled, and plots the distribution of observed thresholds over 5000 test runs for each scenario. As this plot shows, the impact of the filtering is significant - the expected threshold without filtering is 91,000 nodes, with expected thresholds of 4,600 and 370 nodes for the other two categories.

These numbers show that unless aggressive traffic filtering such as ours is implemented, the attacker needn't bother with subtlety. In the worst case scenario an attacker can use the resources of a full /16 without the IDS noticing an attack. Even with just log-level filtering, the attacker can DDoS a target and hide in the noise.

6 Impact Evaluation and Notification

Since HCFAs are common and rarely successful, simple alerts mean that an accurate warning system is still counterproductive: we have developed a system that can now accurately detect attacks and, based on the observed attack frequency, will raise an alarm every two minutes.

We address this problem by dividing alerts on a criterion of *actionability*. The majority of HCFAs are inactionable in the sense that there is relatively little do that an operator can do about them. They are likely to be automated and conducted from compromised hosts, meaning that the actual culprit is elsewhere and the actual owner of the host is unaware of the attack. An attack is actionable if it merits activity by an operator, inactionable otherwise.

Determining actionability is a two-stage process. In the first stage, each of the processors (log-level and state-level) generates a list of anomalous addresses, A, from the current period's log data. These addresses are then passed to an *evaluator*, which determines if there is any activity from the addresses in A that is actionable. As with significance, actionability is a quality that we approximate from concrete behaviors in the logs. Currently, an anomalous address is actionable if there exists a flow from the anomalous address which includes nontrivial payload. For example, if an anomalous address does find an actual target and communicates with it, then that address is actionable. If an anomalous address is actionable, it is sent to operators for further examination, while inactionable addresses are placed on a watchlist which can be examined by operators at their discretion.

Log-level and state-level processors use different criteria for determining if an address is anomalous. Recall that Λ^{insig} will generally be orders of magnitude larger than Λ^{sig} for any time, and consequently the log processor cannot conduct particularly subtle analyses. In the current implementation, the log processor uses a simple top-5 list, identifying the 5 highest degree clients and passing them to the evaluator. The state-level processor will use different methods based on the state model, since our implementation of state-level filtering is graph based, we use component counting method from our previous work [4].

An important consequence of this processing approach is that the log-level processor will generate a steady stream of anomalies and will have a high false positive rate. Note that while the log-level processor has a high false positive rate, the system does not: the evaluator will not pass an alert about a high-degree host unless that host has done something actionable. In comparison to the log-level processor, the state-level processor will generate alerts infrequently (based on the provided data, approximately once every two minutes). The evaluator also expects some degree of redundancy between the anomaly lists - an attack with mixed success may appear in both Λ^{sig} and Λ^{insig}.

7 Attacking Attack Reduction

Attack reduction reduces distractions from HCFAs so that an IDS can focus on more successful, and therefore higher-risk, attacks. To do so, it assumes that the majority of attacks are HCFAs high failure, reaching targets that don't exist and having little impact on the network. We must now address intelligent attackers: how an attacker conversant in attack reduction can manipulate the system to their advantage.

We contend that the primary result of our statistical IDS is to effectively throttle attacker behavior. We assume that an intelligent attacker does not want to be noticed, and too much activity is going to raise alerts from the IDS. Using our graph-based IDS, he can effectively only engage in any activity that involves payload-bearing communications to less than 370 clients and servers combined in a 30s period. This is an upper limit, since it does not include normal activity. Evasion therefore consists of operating under this limit.

An attacker could manipulate the attack reduction system by inserting consistent chaff. Assume that the attacker is aware of when the IDS is in state-level filtering. Throughout this training period, the attacker consistently sends short sessions crafted to pass the log-level filter into the model to nonexistent targets. Because the other traffic is normally distributed, the constant attacks will increase μ_g and provide the attacker with a buffer. Then, during normal operations the attacker can disengage the sources sending chaff traffic and use that breathing room for his own activity.

To do this, the attacker must consistently commit hosts to an attack. Once the system is trained, however, he may return those hosts to his own uses. One way to identify for this type of anomaly would be to use symmetric thresholds. Since Gaussian distributions are symmetric, an unusually low value should be as rare as an unusually high value. If those thresholds are incorporated, than an attacker would have to maintain his hosts in place until the attack was actually conducted.

8 Conclusions

This paper has developed a method to implement an anomaly detection model in protocols that are under constant attack. These consistent HCFAs impact the training and notification processes of anomaly detection systems, and we have shown that without aggressive filtering, an anomaly detection system cannot detect massive attacks in popularly targeted protocols. Using attack reduction we are able to develop a more precise model of normalcy. The impact of having a more precise model is substantial. For example, we showed that in our monitored network, a simple form of anomaly detection improves by more than two orders of magnitude: it will detect 370 attacking hosts with our techniques, but would miss 91,000 attacking hosts (e.g., in a massive DDoS) in the absence of our techniques. By then filtering these detections into actionable and inactionable anomalies, we are then able to better focus the attention of human analysts on attacks that otherwise would have been lost in the noise of HCFAs and that are likely to be more important to examine.

References

[1] D. Barbará, J. Couto, S. Jajodia, and N. Wu. ADAM: a testbed for exploring the use of data mining in intrusion detection. *SIGMOD Rec.*, 30(4):15–24, 2001.

[2] D. Barbará, Y. Li, J. Couto, J. Lin, and S. Jajodia. Bootstrapping a data mining intrusion detection system. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, 2003.

[3] K. Claffy, H. Braun, and G. Polyzos. A parameterizable methodology for internet traffic flow profiling. *IEEE Journal of Selected Areas in Communications*, 13(8):1481–1494, 1995.

[4] M. Patrick Collins and Michael K. Reiter. Hit-list worm detection and bot identification in large networks using protocol graphs. In *Proceedings of the 2007 Symposium on Recent Advances in Intrusion Detection*, 2007.

[5] L. Ertoz, E. Eilertson, A. Lazarevic, P. Tan, J. Srivastava, V. Kumar, and P. Dokas. *Next Generation Data Mining*, chapter 3. MIT Press, 2004.

[6] V. Garg, V. Yegneswaran, and P. Barford. Improving NIDS performance through hardware-based connection filtering. In *Proceedings of the 2006 International Conference on Communications*, 2006.

[7] C. Gates, J. McNutt, J. Kadane, and M. Kellner. Detecting scans at the ISP level. Technical Report SEI-2006-TR-005, Software Engineering Institute, 2006.

[8] C. Gates and C. Taylor. Challenging the anomaly detection paradigm, a provocative discussion. In *Proceedings of the 2006 New Security Paradigms Workshop*, pages 22–29, 2006.

[9] G. Giacinto and F. Roli. Intrusion detection in computer networks by multiple classifier systems. In *Proceedings of the 2002 International Conference on Pattern Recognition*, 2002.

[10] G. Gu, M. Sharif, X. Qin, D. Dagon, W. Lee, and G. Riley. Worm detection, early warning and response based on local victim information. In *Proceedings of the 2004 Annual Computer Security Applications Conference*, 2004.

[11] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, 2004.

[12] W. Lee, S. Stolfo, P. Chan, E. Eskin, W. Fan, M. Miller, and J. Hershkop, S.and Zhang. Real time data mining-based intrusion detection. In *Proceedings of the 2001 DARPA Information Survivability Conference and Exposition*, 2001.

[13] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, 2000.

[14] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, and P. Neuman. A real-time intrusion-detection expert system (IDES). Technical Report Project 6784, CSL, SRI International, 1992.

[15] M. Mahoney and P. Chan. Learning rules for anomaly detection of hostile network trafic. In *Proceedings of the 2003 IEEE International Conference on Data Mining*, 2003.

[16] D. Moore, C. Shannon, G. Voelker, and S. Savage. Network telescopes. Technical report, CAIDA, 2003.

[17] S. Northcutt. *Network Intrusion Detection: An Analyst's Handbook*. New Riders, 1999.

[18] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of internet background radiation. In *Proceedings of the 2004 Internet Measurement Conference*, 2004.

[19] K. Pentikousis and H. Badr. Quantifying the deployment of TCP options, a comparative study. *IEEE Communications Letters*, 8(10):647–649, 2004.

[20] P. Porras and P. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information Systems Security Conference*. NIST, 1997.

[21] U. Shankar and V. Paxson. Active mapping: Resisting nids evasion without altering traffic. In *Proceedings of 2003 IEEE Symposium on Security and Privacy*, 2003.

[22] S. Shapiro and M. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3–4):591–611, 1965.

[23] A. Soule, K. Salamatian, and N. Taft. Combining filtering and statistical methods for anomaly detection. In *Proceedings of the 2005 Internet Measurement Conference*, 2005.

[24] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – A graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[25] R. Stevens. *TCP/IP Illustrated*. Addison Wesley Longman, Inc., 1994.

[26] S. Venkataraman, J. Caballero, D. Song, A. Blum, and J.Yates. Black box anomaly detection: is it utopian? In *Proceedings of the 2006 HotNets Workshop*, 2006.

[27] N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms. In *Proceedings of the 2004 USENIX security symposium*, 2004.

[28] S. Webster, R. Lippmann, and M. Zissman. Experience using active and passive mapping for network situational awareness. In *Proceedings of the 2006 IEEE International Symposium on Network Computing and Applications*, 2006.

[29] J. Wu, S. Vangala, L. Gao, and K. Kwiat. An efficient architecture and algorithm for detecting worms with various scan techniques. In *Proceedings of the 2004 Network and Distributed Systems Security Symposium*, 2004.

[30] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: Global characteristics and prevalence. In *Proceedings of the 2003 ACM SIGMETRICS Conference*, 2003.