Beehive: Large-Scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks

Ting-Fang Yen
RSA Laboratories
Cambridge, MA, USA
tingfang.yen@rsa.com

Alina Oprea
RSA Laboratories
Cambridge, MA, USA
aoprea@rsa.com

Kaan Onarlioglu
Northeastern University
Boston, MA, USA
onarliog@ccs.neu.edu

Todd Leetham
EMC Corp
Hopkinton, MA, USA
todd.leetham@emc.com

William Robertson
Northeastern University
Boston, MA, USA
wkr@ccs.neu.edu

Ari Juels
RSA Laboratories
Cambridge, MA, USA
ajuels@rsa.com

Engin Kirda
Northeastern University
Boston, MA, USA
ek@ccs.neu.edu

ABSTRACT

As more and more Internet-based attacks arise, organizations are responding by deploying an assortment of security products that generate situational intelligence in the form of logs. These logs often contain high volumes of interesting and useful information about activities in the network, and are among the first data sources that information security specialists consult when they suspect that an attack has taken place. However, security products often come from a patchwork of vendors, and are inconsistently installed and administered. They generate logs whose formats differ widely and that are often incomplete, mutually contradictory, and very large in volume. Hence, although this collected information is useful, it is often *dirty*.

We present a novel system, Beehive, that attacks the problem of automatically mining and extracting knowledge from the dirty log data produced by a wide variety of security products in a large enterprise. We improve on signature-based approaches to detecting security incidents and instead identify suspicious host behaviors that Beehive reports as potential security incidents. These incidents can then be further analyzed by incident response teams to determine whether a policy violation or attack has occurred. We have evaluated Beehive on the log data collected in a large enterprise, EMC, over a period of two weeks. We compare the incidents identified by Beehive against enterprise Security Operations Center reports, antivirus software alerts, and feedback from enterprise security specialists. We show that Beehive is able to identify malicious events and policy violations which would otherwise go undetected.

ACSAC '13 Dec. 9-13, 2013, New Orleans, Louisiana, USA
Copyright 2013 ACM 978-1-4503-2015-3/13/12 ...$15.00.

1. INTRODUCTION

Protection of computing infrastructure in large organizations is a mounting challenge. Both generic malware and targeted attacks (i.e., Advanced Persistent Threats (APTs)) are growing in sophistication and outstripping the capabilities of traditional defenses such as antivirus software. At the same time, administrators' visibility into the posture and behavior of hosts is eroding as employees increasingly use personal devices for business applications. The traditional perimeters of organizations are breaking down, moreover, due to new complexities in business intelligence sharing, contractor relationships, and geographical distribution.

Many organizations are responding to this challenging landscape by deploying an assortment of security products that enforce policies and generate situational intelligence in the form of security logs. These products proxy web traffic, detect malware infections, check for policy violations, implement effective authentication mechanisms, and perform many other useful security functions.

These products yield logs that contain high volumes of interesting and useful information about activities in the network, and are among the first data sources consulted by security specialists when an attack is in evidence. For example, authentication logs might suggest that an employee's account has been compromised because a login has occurred from a region that the targeted employee has never visited. As another example, web proxy logs might record which site a victim visited before being compromised by a drive-by download attack.

While it is standard practice to examine logs to discover or conduct forensic analysis of suspicious activity in a network, such investigation remains a largely manual process and often relies on signatures of known threats. A major challenge is that security products often come from a patchwork of vendors and are inconsistently installed and administered. They produce log data with widely differing formats, and logs that are often incomplete, mutually contradictory, very large in volume (e.g., security-relevant logs in a large enterprise grow by terabytes per day), and filled with non-specific or spurious event records. Hence, although this collected information is useful, and is sometimes the only information at hand, at the same time it is often *dirty*.

In this paper, we attack the problem of automatically mining and extracting knowledge from the dirty log data produced by a wide variety of security products in a large enterprise. Our aim is to improve on signature-based approaches to detecting security incidents and instead achieve behavioral detection of suspicious host activities that signal potential security incidents. These incidents can then be further analyzed by incident response teams to determine whether a policy violation or attack has occurred. A key insight of our approach is that behaviors observed in enterprise settings are often significantly more constrained by policy and typical employee behavior than those on the open Internet. Thus we can effectively identify suspicious activities by analyzing behaviors in enterprise-specific ways.

Our automated approach, called Beehive, has three layers: (1) Functionality to *parse, filter, and normalize log data* using network-specific configuration information; (2) *Feature-generators* that process normalized log data to produce a set of fifteen distinct features for each host per day; and (3) A *detector that performs clustering over features* to identify outlying, suspicious periods of host activity, which the system reports as *incidents*.

We have evaluated Beehive on the log data collected in a large enterprise, EMC, over a period of two weeks. We compare the incidents automatically identified by Beehive against enterprise Security Operations Center reports, antivirus software alerts, and feedback received from information security specialists in the company. We show that Beehive is able to identify a number of malicious events within the enterprise network, including malware infections and policy violations, that otherwise go unnoticed by existing, state-of-the-art security tools and personnel.

Our work makes the following contributions:

- We propose a novel automated approach called Beehive that analyzes very large volumes of disparate log data collected in large organizations to detect malicious activity such as malware infections and policy violations. Beehive leverages unique features of enterprise networks to produce accurate, actionable information.

- We have evaluated Beehive on more than 6 terabytes of log data and have correctly identified suspicious activities that went unnoticed by existing state-of-the-art security tools and personnel.

- To the best of our knowledge, ours is the first exploration of the challenges of "big data" security analytics for large volumes of disparate, real-world log data.

2. PROBLEM AND CHALLENGES

Here we describe the threat model we address, the nature of the log data on which the system operates, and the challenges that arise in designing and evaluating Beehive.

2.1 Threat Model

Beehive aims to detect threats arising in two different scenarios. The first scenario involves a network host that is infected or at imminent risk of infection. A host of this kind may exhibit malicious or dangerous behaviors such as contacting an attack website, communicating with a C&C server, or exfiltrating data to a previously unseen external destination. These behaviors may result from the automated activities of malware (e.g, zombie machines in a botnet), direct control of hosts by an attacker (e.g., through a remote access Trojan), or benign users being tricked into dangerous behavior (e.g., via phishing attacks).

The second scenario concerns host-based user activities that violate enterprise policies. These activities are not necessarily malicious in intent, but are potentially damaging to the business, and thus undesirable. Examples include peer-to-peer file sharing, instant messaging, multimedia streaming, viewing of adult content, online gaming, and using proxies and tunneling services in an attempt to bypass enterprise firewalls and other network-perimeter security mechanisms.

2.2 Data

Beehive uses a wide range of logs generated by various network devices, including web proxies that log every outbound connection, DHCP servers that log dynamic IP address assignments, VPN servers that log remote connections to the enterprise network, Windows domain controllers that log authentication attempts within the corporate domain, and antivirus software that logs the results of malware scans on end hosts. These logs are stored in a commercial security information and event management (SIEM) system.

All of these devices monitor and log the activities of enterprise hosts transparently, with the notable exception of the web proxy. For every external destination, the web proxy consults a list of domain reputation scores and site categories maintained by the product vendor, and automatically blocks the connection if the destination has a low reputation or is in a prohibited category (e.g., adult-content sites). However, if the destination is not on this list, the proxy displays to the user a warning page describing the potential dangers of connecting to an unknown destination. The user must then explicitly click on a link to acknowledge agreement to the company's network policies, or not visit the site.

Beehive makes heavy use of the web proxy logs, since the majority of network activity in the enterprise is over HTTP or HTTPS. In addition, these logs include all fields in HTTP headers, making them a particularly useful resource for understanding users' browsing behavior, detecting suspicious connections, and forensic analysis. Table 1 lists a subset of the fields included in web proxy logs.

IP Header	Source/destination IP and port, protocol
Transport Header	Destination domain, URL, HTTP status code, web referer, domain reputation, domain category, user-agent string
Connection Attribute	Sent and received bytes, network policy

Table 1: Fields in web proxy logs.

At EMC, 1.4 billion log messages are generated daily on average, at a rate of around one terabyte a day. These "raw" logs contain huge amounts of information about user and host behaviors. They are also noisy — with non-standardized timestamps, different identifiers (i.e., some store a host's IP address, some the hostname, others the username) — and can be truncated or arrive out of order. We discuss the challenges of analyzing such "big data" in the next section.

2.3 Challenges

The massive number of events observed on a real-life enterprise network and logged inside the SIEM system poses a major challenge (a "big data" problem) in log analysis and detection of security incidents. As noted above, we observed collection by the SIEM an average of 1.4 billion logs per day in our case study. Timely detection of critical threats by Beehive thus requires efficient data-reduction algorithms and strategies to focus on security-relevant information in the logs.

A second challenge for Beehive is that of identifying meaningful security incidents in the face of a significant semantic gap between the logs collected by the SIEM system and the information that security analysts require to identify suspicious host behavior. In our case, Beehive needs to produce a daily report of suspicious *hosts*; however, most network devices only log the *IP addresses* of network endpoints. Associating IP addresses with specific hosts requires correlation across different logs, which can be especially challenging when IP addresses are dynamically assigned in the network. The situation is complicated still more by the fact that the logging devices may be located in different time zones (as is typical for the infrastructure of a global organization), and thus the timestamps they produce must be normalized for accurate temporal correlation.

Finally, a major challenge in designing Beehive is the difficulty of evaluating its effectiveness accurately, due to an inherent lack of ground truth. Specifically, an enterprise network typically has state-of-the-art security technologies deployed both on the network perimeter and on hosts, and security analysts actively work toward resolving known incidents. As a result, real-life network data obtained from such a network often lacks traces of known security threats. For example, malware on hosts is often cleaned or quarantined before becoming active, and accesses to botnet C&C servers are often blocked by the firewall. Such threat resolution prevents automatic testing of Beehive over a ground truth of *known* malicious network traffic, as none is in evidence. Many of the incidents identified by Beehive are in fact *unknown* to security systems in use in the enterprise. We therefore must rely on manual evaluation of the security incidents and suspicious hosts identified by Beehive.

In designing Beehive, we explore and tackle these challenges of big data security analytics. Briefly, in a pre-processing phase, Beehive normalizes the timestamps of all log entries to UTC, determines statically and dynamically assigned IP addresses of hosts, and constructs bindings between hosts and IP addresses, thus attributing every logged event to a specific host. Beehive then extracts enterprise-specific features for individual hosts, and clusters hosts with similar suspicious behaviors. The resulting outlying clusters are presented as incidents to be examined by security analysts.

To assess the efficacy of Beehive in this study, we manually investigate its reported incidents with the help of our enterprise's Security Operations Center (SOC). Given existing enterprise resolution of attacks with known signatures, of particular interest in our study is Beehive's identification of network policy violations by users and of malware *n*ot detected by current signature-based tools.

3. THE BEEHIVE SYSTEM

In this section we describe in detail the operation of the Beehive system. We start by describing our approach to remove noise and inconsistencies in the data collected by the SIEM. We then discuss the selection of features distinctive to an enterprise setting for detecting misbehavior. Finally, we describe the unsupervised learning approach we employed for generating incidents of anomalous host behavior.

3.1 Data Normalization

In order to address the challenges of dirty and inconsistent data explained in the previous sections, Beehive preprocesses the log data before starting its behavioral-analysis. We explain different stages of this process below.

Timestamp Normalization. In an enterprise running a global network, devices that produce critical logs are located in different geographies. Moreover, these devices may timestamp logs differently, using their local time zones, UTC, or other time representations. This results in log entries with inconsistent timestamps that must be normalized to a standard representation before any temporal analysis becomes accurate. While delegating the coordination of time settings to network administrators may be feasible for small networks, these approaches are unworkable at the scale of a global enterprise.

Beehive addresses this problem by leveraging the common use by enterprises of a central SIEM system for log management that tags each log entry with its own timestamp t_{siem}, recording the time at which the log was received by the SIEM. For each device that sends logs to the SIEM system, we first compute a set of time difference values $\Delta_i = t_{siem,i} - t_{device,i}$ (rounded off to the nearest 30 minutes) from each log entry i generated over a large time period (e.g., one month). Next, we determine the timestamp correction value $\Delta_{correction}$ for the device by setting it to the value Δ_i that accounts for the majority of differences. Applying this correction value to each device timestamp gives us a normalized timestamp value, $t_{normalized,i} = t_{device,i} + \Delta_{correction}$. In our case, as EMC's SIEM system is configured to use UTC timestamps, $t_{normalized,i}$ corresponds to UTC time. We apply this normalization technique to each device on the network that produces log data, normalizing all log timestamps to UTC.

IP address-to-Host Mapping. In an enterprise setting, end hosts are usually assigned IP addresses dynamically for a short time period when they connect to the network, through the Dynamic Host Configuration Protocol (DHCP). This makes it difficult to associate the IP addresses reported in logs with unique host machines during analysis, because that same IP address could be assigned to different hosts after the event has been logged.

To address this issue, Beehive analyzes the DHCP server logs collected in the SIEM system and constructs a database of IP-to-host mappings (i.e., *bindings*) over time. Each binding is represented as a tuple {IP address, hostname, MAC address, start-time, end-time} mapping an IP address to a host in a specific time interval. This algorithm is run daily to update existing bindings as new DHCP logs become available. Given the resulting database of bindings, Beehive can identify the host corresponding to a given IP address with reference to a normalized timestamp.

Static IP Address Detection. An IP-to-host lookup on the bindings database we construct may fail because an IP address is assigned statically to a specific host, not dynamically leased. Maintaining a list of static IP address assignments, however, is difficult in a large enterprise network, and administrator-created lists are often non-existant or outdated. Thus Beehive also adopts the following method for automatically identifying hosts with static IP addresses.

In the bootstrap step, we first retrieve all IP addresses found in all collected logs to create a pool of IP addresses active in the enterprise, denoted by set A. Next, we retrieve IP addresses from logs that we know *must only* contain hosts with dynamic IP addresses, such as DHCP and VPN logs, to create a pool of known dynamic IP addresses, denoted by set D. We then compute the set difference $S = A - D$ which contains *potentially* static IP addresses, perform a reverse DNS lookup for every address in S, and save the results to complete the bootstrap phase. Periodically (e.g., once a day), as new logs become available, we repeat this procedure to harvest new IP addresses and update the sets A, D and S accordingly. However, unlike before, with each iteration we resolve IP addresses in S to their host names and compare the names to the previously stored values. If the host names changed between two iterations, we conclude that the given IP address is not statically assigned, and we remove it from the set S. In this way, we refine the pool of potentially static IP addresses with each iteration. If Beehive fails to find a corresponding binding for an IP-to-host lookup, but instead finds the given address in S, we treat that IP address as a static host and use the host name found in S.

Dedicated Hosts. Beehive focuses on monitoring the behavior of *dedicated hosts*, which are machines utilized by a single user. Since a list of dedicated hosts is difficult to maintain in large enterprises due to constantly changing network configurations, we infer the list of dedicated hosts from the data available in the SIEM system.

We make use of authentication logs generated by Microsoft Windows domain controllers for this purpose. For each user in the enterprise, a history is kept of hosts onto which the user had authenticated (i.e., logged on), the number of authentications, and the authentication timestamps. This information is collected over the course of three months to build an accurate history of user activity. At the end of the collection period, we consider a host as "dedicated" if a single user is responsible for the large majority (e.g., 95%) of the authentication events on that host. Through this process, we have identified over 78,000 dedicated hosts at EMC.

3.2 Feature Extraction

We extracted features from the logs to characterize outbound communications from the enterprise. Our feature selection is guided by observation of known malware behaviors and policy violations within EMC, as well as properties of the environment in which Beehive operates, i.e., the presence of perimeter defenses in the form of strict firewall policies, the business orientation of (most) users' activities, and the (relatively) homogeneous software configurations of enterprise-managed hosts.

For each dedicated host in the enterprise, we generate daily a feature vector that includes the 15 features listed in Table 2. The features can be grouped into four categories: features based on new and unpopular destinations contacted by the host, features related to the host's software configuration, features related to the company policy, and features based on traffic volume and timing. We describe each of them in detail below.

Feature Type	#	Description
Destination-Based	1	New destinations
	2	New dests. w/o whitelisted referer
	3	Unpopular raw IP destinations
	4	Fraction of unpopular raw IP dests.
Host-Based	5	New user-agent strings
Policy-Based	6	Blocked domains
	7	Blocked connections
	8	Challenged domains
	9	Challenged connections
	10	Consented domains
	11	Consented connections
Traffic-Based	12	Connection spikes
	13	Domain spikes
	14	Connections bursts
	15	Domain bursts

Table 2: Beehive features.

3.2.1 Destination-Based Features

We are interested in identifying hosts that communicate with new, or obscure, external destinations that are never (or rarely) contacted from within EMC. Assuming popular websites are better administered and less likely to be compromised, connections to uncommon destinations may be indicative of suspicious behavior (e.g., communication with command-and-control servers).

New Destinations. Our first destination-based feature is the number of *new* external destinations contacted by each host per day. We build a history of external destinations contacted by internal hosts over time. After an initial bootstrapping period of one month, we consider a destination to be *new* on a particular day if it has never been contacted by hosts in the enterprise within the observation period (and as such is not part of the history). On a daily basis we update the history to include new destinations contacted in the previous day.

We encountered a number of challenges when analyzing web proxy logs to build the history of external destinations. In our initial, naïve, implementation, we process every proxy log, resolve all destinations that are IP addresses, and include all observed new destinations in the history. It turns out that this naïve approach is not scalable both in terms of running time and history size.

First, the naïve implementation takes 15 hours to process a day's worth of web proxy logs (the equivalent of 300 million logs, or 600 GB). Second, to our surprise, the number of new destinations does not decrease over time, even as the size of the history grows to 4.3 million unique destinations over the course of a month. As shown in Figure 1, between 30% to 40% (about 145,000) of all unique destinations each day are new. In the naïve approach, all of these new destinations are added to the history daily, and as such the size of the history grows indefinitely over time.

Further inspection of the web proxy logs showed that the majority of the new destinations are content delivery networks (CDNs), cloud services (which frequently use random strings as subdomains), or IP addresses belonging to popular services (Google, Facebook, Twitter). To make Beehive scalable, we employ a number of data reduction techniques, including filtering, custom whitelisting and domain "folding."

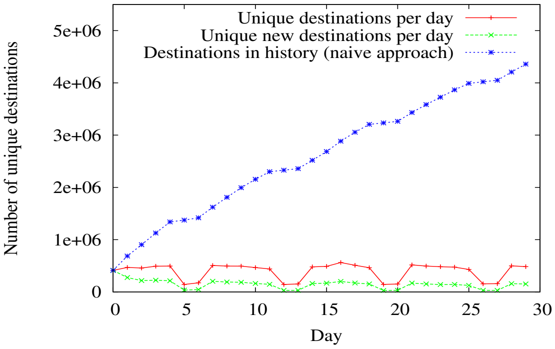

Figure 1: Destinations contacted by internal hosts, naïve approach.

We first filter "popular" destinations by creating a custom whitelist, where "popularity" is defined over hosts in the enterprise. The whitelist includes external destinations (both domains and IP subnets) whose number of interacting internal hosts over time (i.e., a training period of one week) exceeds a threshold. Figure 2 shows the amount of web traffic filtered using a whitelist constructed from the first week's worth of data from April 2013. A threshold of 100 hosts achieves a reduction from 300 million logs a day to 80 million—a 74% reduction in the number of logs Beehive needs to process.

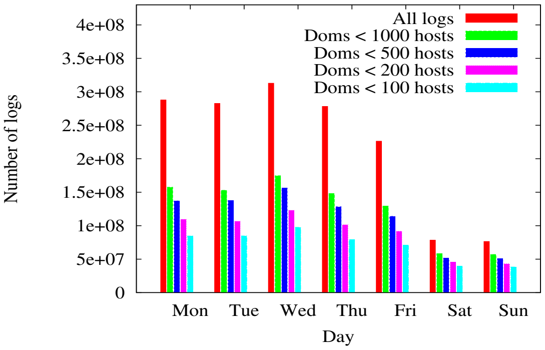

Figure 2: Data reduction through custom whitelisting.

In addition to custom whitelisting, we "fold" destinations to the second-level domain so as to filter services employing random strings as subdomains. We also ignore connections retrieving "favicon" — likely due to bookmarks. Finally, we choose not to resolve raw IPs (since most legitimate sites are referred to by their domain name), and always consider raw IPs that are not on the whitelist as "new".

These optimizations reduce daily processing time from 15 hours to about five hours. The number of new (folded) destinations added to the history is reduced to an average of 28,000 per day. After a period of four months (from January 13th to May 13th, 2013), our history of external destinations has 2.7 million folded domains, whereas the history built naïvely already had 4.3 million domains after one month.

New destinations without whitelisted referer. Our second destination-based feature is an extension of the first one, but counts the number of "new" destinations contacted by a host *without* a whitelisted HTTP referer. Users are most commonly directed to new sites by search engines, news sites, or advertisements. In these cases, we expect the HTTP referer to be one listed in our custom whitelist. Host that visit new sites without being referred by a reputable source (or with no referer at all) are considered more suspicious.

Unpopular IP destinations. We are also interested in unpopular external destinations that are raw IP addresses. We first count the number of destinations contacted by a host that are both unpopular (not on the custom whitelist described above) and are IP addresses. Connections to unpopular IPs can indicate suspicious activity, as legitimate services can usually be reached by their domain names. Second, we include the *fraction* of unpopular destinations contacted by a host that day that are IP addresses. While occasional communication with IP destinations is normal (e.g., to IP ranges owned by CDNs), such behavior is suspicious when frequent.

3.2.2 Host-Based Features

Hosts in an enterprise are significantly more homogeneous in their software configurations than, for example, those in academic networks. Hence we are interested in cases where hosts install new (and potentially unauthorized) software.

Lacking visibility onto the host machine, and having access only to logs collected from network devices, we infer the software configurations on a host from the user-agent (UA) strings included in HTTP request headers. A user-agent string includes the name of the application making the request, its version, capabilities, and the operating environment. Our host-based feature is hence the number of "new" UA strings from the host.

We build a history of UA strings per host over a month-long period, during which every UA string observed from the host is stored. Afterwards, a UA string is considered "new" if it is sufficiently distinct from all UA strings in that host's history. Edit distance (also called Levenshtein distance) is used to compare UA strings, measuring the number of character insertions, deletions, and substitutions required to change one string into another. This allows us to accommodate "new" UA strings that result from software updates, where only a small substring (e.g., the version number) changes.

3.2.3 Policy-Based Features

Also unique to enterprise environments is the enforcement of network policies on outbound connections. As described in Section 2.2, a connection to an external destination can be blocked if it has a dubious reputation or is categorized as prohibited for employees. Blocked domains (and connections) are thus a coarse indicator of host misbehavior.

Upon visiting an *unknown* destination, i.e., one that has not yet been categorized or rated, the user must explicitly agree to adhere to the company's policies before being allowed to proceed. We refer to the domains (and connections) that require this acknowledgment as *challenged*, and those to which the user has agreed as *consented*.

Our policy-based features include all three types of communications described above. For a host, we count the number of domains (and connections) contacted by the host that are blocked, challenged, or consented.

3.2.4 Traffic-Based Features

Sudden spikes in a host's traffic volume can be caused by malware (e.g., scanning, or bot activities in response to botmaster commands) or automated processes. Our traffic-based features attempt to capture these interesting activities by the amount of time when a host is generating abnormally high volumes of traffic.

Specifically, we define a connection (or domain) spike as a one-minute window when the host generates more connections (or contacts more domains) than a threshold. A connection (or domain) burst is a time interval in which every minute is a connection (or domain) spike.

To identify an appropriate threshold for "high" traffic volume, we examine all dedicated hosts over a one-week interval, and count the number of connections (and domains) each host generates (or contacts) per minute. Figure 3 shows the cumulative distribution across all one-minute windows for all dedicated hosts. 90% of the hosts generated less than 101 connections, and contacted less than 17 distinct domains, per minute. We hence set the threshold for connection spikes and domain spikes to these values.

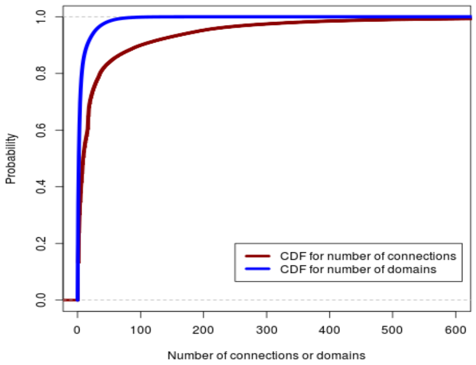

Figure 3: CDF for number of web connections and number of domains contacted by a host per one-minute interval.

For bursts, we relax its definition slightly, so that the threshold for spikes within a burst is the 75% percentile of all one-minute windows across all hosts (see Figure 3). This value is 26 for connection spikes, and 6 for domain spikes.

For each dedicated host, its traffic-based features include: 1) the number of connections spikes, 2) the number of domain spikes, 3) the duration of the longest connection burst, and 4) the duration of the longest domain burst.

3.3 Clustering

Lacking ground truth as to which hosts are infected or behaving anomalously (a challenge we describe in Section 2.3), we tackle the problem of detection through an unsupervised learning approach – clustering. Since employees in the enterprise perform specific job functions on the corporate network, and there are multiple employees in most departments, we should be able to observe groups of hosts (belonging to users with similar roles) exhibiting similar behaviors, while misbehaving hosts with unique behavioral patterns appear as *outliers*.

Given the 15 features described in Section 3.2, each internal host is represented as a 15-dimensional vector, $v = (v[1], v[2], \cdots, v[15])$. However, the features may be related or dependent upon one another; e.g., a domain spike also triggers a connection spike. To remove such dependencies between the features and reduce the dimensionality of the vectors, we apply Principal Component Analysis (PCA) [23].

Briefly, PCA enables data reduction by projecting the original vectors onto a new set of axes (i.e., the principal components). Each principal component is chosen to capture as much of the variance (and thus the original information) in the data as possible. Depending on how much variance we want to capture from the original data, the top m principal components are selected, permitting projection of the original vectors down to dimensionality m. In Beehive, we select the top m components that capture at least 95% of the data variance.

We apply a clustering algorithm to the projected vectors after PCA. Our algorithm is an adaptation of the K-means clustering algorithm, but does not require the number of clusters to be specified in advance [25].

1. Randomly select a vector as the first cluster hub. Assign all vectors to this cluster.
2. Select the vector furthest away from its hub as a new hub. Reassign every vector to the cluster with the closest hub.
3. Repeat step 2 until no vector is further away from its hub than half of the average hub-to-hub distance.

We compare vectors via L1 distance, i.e., for vectors v_1 and v_2, their distance is $\mathsf{L1Dist}(v_1, v_2) = \sum_{i=1}^{m} |v_1[i] - v_2[i]|$.

The clustering algorithm in Beehive is applied daily on the feature vectors for all active, dedicated hosts in the enterprise. (We observe between 27,000 and 35,000 hosts active during weekdays, and between 9,000 and 10,100 hosts active on weekends.) Interestingly, after one iteration of the algorithm, the vast majority of hosts fall into one large cluster, while the remaining clusters consist of few hosts whose behaviors deviate significantly from the norm (i.e., outliers).

Beehive generates *incidents* for the top outlying hosts, and reports them to the security analyst. Note that the algorithm forms clusters by iteratively identifying the node that is furthest away from its cluster hub, and so the clusters have an inherent ordering to them. In cases where the algorithm is biased by extreme outliers (e.g., forming only two clusters, one with a single node, and the other with all other nodes), we apply PCA and the clustering algorithm again to the largest cluster. This process is iterated until at least 50 outlying hosts are identified in a day.

4. EVALUATION

We evaluated Beehive in a large enterprise, EMC, over a period of two weeks, from April 22 to May 5, 2013. During this period, Beehive generated 784 incidents, with an average of 56 incidents per day and a standard deviation of 6.88. Among these, only eight incidents overlap with the alerts generated by the enterprise's state-of-the-art security tools.

To evaluate Beehive in the absence of ground truth – in the sense of accurate, existing classification of identified incidents – we resort to a two-step process of manual investigation. We first examine and label all incidents generated by Beehive (see Section 4.2). Incidents whose cause we could not identify are labeled as "suspicious" and presented to the enterprise's Security Operations Center (SOC) for further analysis (see Section 4.3).

4.1 Clustering Context

In addition to the host and its feature vector, each Beehive incident also includes contextual information about the cluster each host belongs to, such as the number of hosts in the cluster and the average value of the feature vectors. This information facilitates the manual investigation process described below, as it allows the analyst to quickly identify the distinctive features in each cluster.

To give intuition about the types of activities captured by the clustering, Figure 4 shows, for all 15 outlying clusters generated on April 24th, the normalized average feature vectors. That is, we first compute the mean feature vector of each cluster averaged over all vectors in that cluster, and then normalize each vector component — i.e., feature value – relative to the maximum across all mean feature vectors. As shown in Figure 4, the clusters are distinct in that they are characterized by different (subsets of) features.

While clusters are distinct from one another, the hosts in the same cluster exhibit very similar behaviors. Table 3 shows the actual feature vectors of hosts in two example clusters (3 and 6) on April 24th. For instance, cluster 3 is unique in its high numbers of blocked connections, connections spikes, and domain spikes — suggesting an automated process is responsible for the activity. (Indeed, during our investigation described in the next section, hosts in this cluster are found to violate the company policy by contacting blocked domains excessively.) As another example, hosts in cluster 6 contacted high numbers of new destinations, as well as generating connection spikes to blocked and challenged domains. Our investigation revealed that this cluster is comprised solely of infected hosts that contact dynamically generated domains (DGAs).

In the following, we detail our manual investigation process and the suspicious activities detected by Beehive.

4.2 Manual Labeling

Our manual investigation process for a host began by identifying the feature(s) that were distinctive for the host's cluster. We then attempted to identify the root cause of this distinction using contextual information reported by Beehive and examining raw logs collected by the SIEM system about the host. The latter involved looking up the UA strings, HTTP status codes, web referers, and timing patterns in connections from the host, as well as the reputation of contacted domains (using McAfee SiteAdvisor and URLvoid [1]) and when they were registered (using DomainTools [2]).

Briefly, 25.25% of the Beehive incidents were either confirmed malware or "suspicious" (we give more details on "suspicious" incidents in Section 4.3), 39.41% were violations of enterprise security policies, and 35.33% were associated with unrecognized (but noisy) software or services. Table 4 lists the number and percentage of incidents in each category.

Category	# of incidents	
Malware	117	14.92%
Suspicious	81	10.33%
Policy Violation - Tunneling	1	0.12%
Policy Violation - File sharing	2	0.25%
Policy Violation - Streaming	86	10.96%
Policy Violation - IM	56	7.14%
Policy Violation - Porn	6	0.76%
Policy Violation - Gaming	13	1.65%
Policy Violation - Proxy	4	0.51%
Policy Violation - Remote access	8	1.02%
Policy Violation - Blocked sites	133	16.96%
Other - Uncategorized sites	57	7.27%
Other - Browsing	63	8.03%
Other - Automated	157	20.02%

Table 4: Breakdown of the categories of Beehive incidents over two weeks from April 22 to May 5, 2013.

Many of the malware detected by Beehive contacted (or attempted to contact) domains created by domain-generation algorithms (DGAs). The destination-based features (see Section 3.2) proved to be most useful in identifying such malware, as most of the DGA hosts sit in clusters with high numbers of new destinations. Among clusters from April 24th presented in Figure 4, all of the hosts in clusters 1, 6 and 8 were confirmed to be infected with DGA malware. *None* of these infections were previously detected by antivirus software or by the enterprise SOC.

Beehive also detected violations of the enterprise's network security policies. As with many companies, these forbid substantive personal use of company computing resources, installation of software not explicitly approved by the company, access to offensive materials (e.g. pornographic or otherwise disparaging content), and injudiciously high consumption of bandwidth for business purposes. Our manual investigation found 16.96% of the incidents associated with heavy traffic to external destinations blocked due to business concerns or labeled as potentially unsafe (e.g., sites associated with freeware). 10.96% of incidents were caused by high-volume video streaming applications, often characterized by connection or domain spikes or numerous challenged connections. Others included file sharing, third-party instant messaging, online gaming, viewing of adult content, or using tunneling or proxy services to bypass network security mechanisms (e.g., to access blocked websites).

In addition to malware and policy violations, Beehive also detected unusual activities originating from possibly benign, but unrecognized and likely automated applications. For example, 20.02% of the incidents involved hosts that made tens of thousands of connections a day to the same website. These were typically news, sports, or finance sites, but included ad servers, online forums, and sites hosted on Google

[1] http://www.urlvoid.com

[2] http://www.domaintools.com

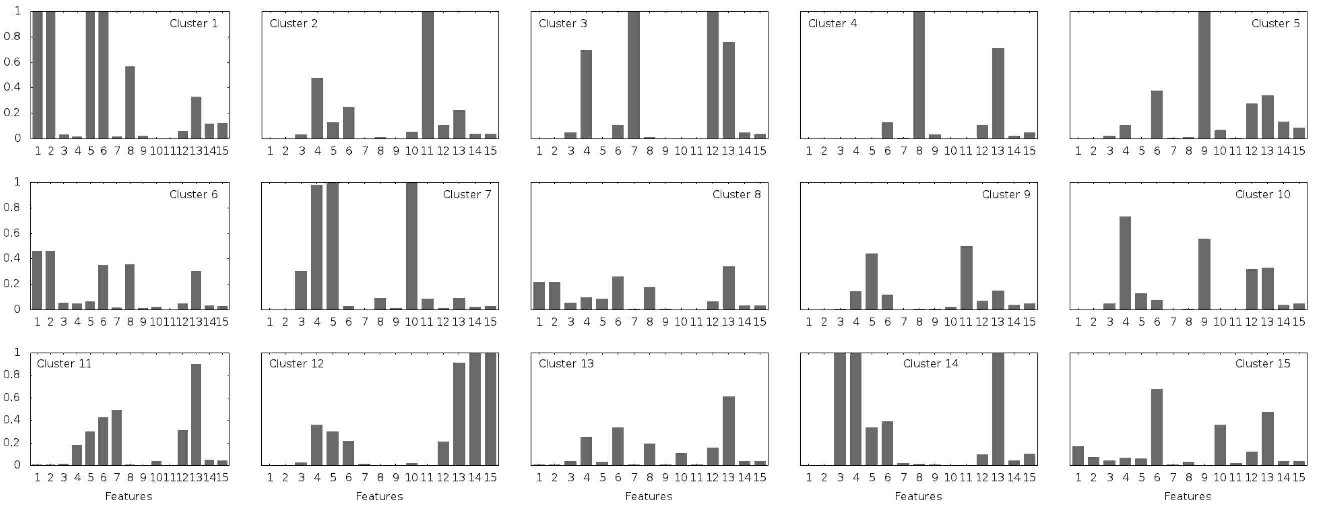

Figure 4: Normalized average feature vectors for clusters generated on April 24, 2013.

Features	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cluster 3	0	0	19	0.95	0	4	47833	7	52	0	0	386	96	2	2
	0	0	2	0.33	0	11	25479	1	1	0	0	309	6	2	1
Cluster 6	247	247	11	0.041	1	8	61	156	163	0	0	11	19	1	1
	200	200	14	0.053	0	36	435	142	170	6	10	27	34	2	2
	239	239	22	0.080	0	26	976	153	177	0	0	26	29	2	1
	214	214	0	0	0	31	739	142	147	0	0	0	0	0	0
Maximum value	490	489	286	0.984	4	114	47833	418	16373	64	3832	386	213	48	48

Table 3: Feature vectors for hosts in two example clusters. Each row corresponds to the feature vector for a host in that cluster. The last row shows the maximum value for each feature across all clusters on April 24, 2013.

App Engine. Several hosts were found to be mining bitcoins. We labeled all of these hosts as "Other - Automated."

Finally, 8.03% of the incidents either had high numbers (thousands per day) of *consented* connections, indicating that the user had explicitly acknowledged the company's network policies before proceeding to the site, or long periods (i.e., lasting several hours) of continuous web traffic to various domains. Without further evidence of malicious behavior, we classified these incidents in the "Other - Browsing" category, though they may not be entirely user-driven.

4.3 Investigation by the SOC

The first-round manual labeling process described in Section 4.2 yielded 81 incidents (10.33%) for which we could not identify the applications responsible for the observed behavior in the incident. The hosts in those incidents communicated with low-reputation websites (e.g., flagged by antivirus vendors, registered within the past six months, with DGA-like subdomains), downloaded executables or zipped files, used malformed HTTP user-agent strings, or repeatedly contacted the same URL at semi-regular intervals.

We labeled these incidents as "suspicious," and presented them to the enterprise SOC for a second round of analysis. Table 5 shows the labels the SOC assigned to these incidents.

SOC label	# of suspicious incidents	
Adware or Spyware	35	43.21%
Further investigation	26	32.09%
Other malware	9	11.11%
Policy Violation - Gaming	1	1.23%
Policy Violation - IM	1	1.23%
Policy Violation - Streaming	2	2.47%
Other - Uncategorized sites	7	8.64%

Table 5: "Suspicious" incidents categorized by the SOC at EMC.

While 54.32% of the "suspicious" incidents were confirmed by the SOC as adware, spyware, or known malware, a significant fraction (32.09%) were considered serious incidents and merit further investigation. These questionable activities are potentially previously unknown malware or other threats opaque to state-of-the-art security tools, and in-depth host inspection is necessary to determine the exact root cause.

A small number of the investigated "suspicious" incidents were identified as policy violations (4.93%) or uncategorized sites (8.64%). Their association with obscure software or low-reputation sites prompted their identification as "suspicious" in the first-round investigation.

4.4 Summary

With assistance from the enterprise SOC, we manually investigated 784 Beehive incidents generated over the course of two weeks. Overall, we find 25.25% of the incidents to be malware-related or that warrant further SOC investigation, 39.41% to be policy violations, and 35.33% associated with unrecognized, but automated, software or services. Only 8 of the 784 incidents (1.02%) were detected by existing state-of-the art security tools, demonstrating Beehive's ability to identify previously unknown anomalous behaviors.

5. RELATED WORK

Network and host-based intrusion detection systems that use statistical and machine learning techniques have seen two decades of extensive research [11, 15, 33]. In addition, both commercial and open source products that combine signature, traffic and anomaly inspection techniques are widely available today [1, 2, 3]. To the best of our knowledge, though, we present the first study of the challenges of sanitizing, correlating and analyzing large-scale log data collected from an enterprise of this scale, and of detecting both compromised hosts and business policy violations.

There are many existing systems that aim to help human analysts detect compromised hosts or stolen credentials, and also a number of case studies on enterprise networks. For example, Chapple et al. [12] present a case study on the detection of anomalous authentication attempts to a university virtual private network using a clustering technique focused on geographic distance. Zhang et al. [41] extend this approach with additional machine-learning features to automatically detect VPN account compromises in university networks. In an earlier study, Levine et al. [26] use honeypots to detect exploited systems on enterprise networks. Similarly, Beehive aims to automate detection tasks for security analysts. In addition to new detection methods, however, we present the largest-scale case study so far on a real-life production network, together with the unique challenges of analyzing big and disjoint log data.

While there exists work that analyzes a specific type of malicious activity on the network (e.g., worms [37] or spammers [32]), recent attempts at malware detection mainly focus on detection of botnets. For instance, several systems [10, 27, 35] use classification and correlation techniques to identify C&C traffic from IRC botnets. BotTrack [16] combines a NetFlow-based approach with the PageRank algorithm to detect P2P botnets. A number of studies monitor crowd communication behaviors of multiple hosts and analyze the spatial-temporal correlations between them to detect botnet infected hosts independent of the protocol used for C&C communication [18, 20, 24, 40], while others specifically focus on DNS traffic similarity [13, 36]. BotHunter [19] inspects the two-way communications at the network perimeter to identify specific stages of botnet infection. DISCLOSURE [8] presents a set of detection features to identify C&C traffic using NetFlow records. Other works focus on tracking and measuring botnets [4, 14, 17, 22, 34].

Other work aims to identify domains that exhibit malicious behavior instead of infected hosts. For example, some systems [30, 29, 31, 21] propose various detection mechanisms for malicious fast-flux services. In a more general approach, EXPOSURE [9], Notos [5], and Kopis [6] perform passive DNS analysis to identify malicious domains. Ma et al. [28] extract lexical and host-based URL features from spam emails and use active DNS probing on domain names to identify malicious websites. Another branch of research aims to detect dynamically generated malicious domains by modeling their lexical structures and utilizing the high number of failed DNS queries observed in botnets using such domains [7, 38, 39]. In comparison, Beehive uses standard log data to detect suspicious activity in an enterprise network.

These approaches use network data for detection. In contrast, Beehive uses dirty enterprise log data to detect potentially malicious host behavior as well as policy violations specific to an enterprise setting.

6. CONCLUSIONS

In this paper, we presented a novel system called Beehive that attacks the problem of automatically mining and extracting knowledge in a large enterprise from dirty logs generated by a variety of network devices. The major challenges are the "big data" problem (at EMC, an average of 1.4 billion log messages—about 1 terabyte—are generated per day), and the semantic gap between the information stored in the logs and that required by security analysts to detect suspicious behavior.

We developed efficient techniques to remove noise in the logs, including normalizing log timestamps to UTC and creating an IP-to-hostname mapping that standardizes host identifiers across log types. Using a custom whitelist built by observing communications patterns in the enterprise, we effectively reduced the data Beehive inspects from 300 million log messages per day to 80 million (a 74% reduction).

Beehive improves on signature-based approaches to detecting security incidents. Instead, it flags suspected security incidents in hosts based on behavioral analysis. In our evaluation, Beehive detected malware infections and policy violations that went otherwise unnoticed by existing, state-of-the-art security tools and personnel. Specifically, for log files collected in a large enterprise over two weeks, 25.25% of Beehive incidents were confirmed to be malware-related or to warrant further investigation by the enterprise SOC, 39.41% were policy violations, and 35.33% were associated with unrecognized software or services.

To the best of our knowledge, ours is the first exploration of the challenges of "big data" security analytics at the scale of real-world enterprise log data.

Acknowledgments

We are grateful to members of the EMC CIRT team for providing us access to the enterprise log data, and for their help in investigating Beehive alerts.

This research is partly funded by National Science Foundation (NSF) under grant CNS-1116777. Engin Kirda also thanks Sy and Laurie Sternberg for their generous support.

7. REFERENCES

[1] OSSEC – Open Source Security. http://www.ossec.net.

[2] Snort. http://www.snort.org.

[3] The Bro Network Security Monitor. http://www.bro.org/.

[4] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *IMC*, 2006.

[5] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In *USENIX Security*, 2010.

[6] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, II, and D. Dagon. Detecting Malware Domains at the Upper DNS Hierarchy. In *USENIX Security*, 2011.

[7] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From Throw-away Traffic to Bots: Detecting the Rise of DGA-based Malware. In *USENIX Security*, 2012.

[8] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel. Disclosure: Detecting Botnet Command and Control Servers Through Large-scale NetFlow Analysis. In *ACSAC*, 2012.

[9] L. Bilge, E. Kirda, K. Christopher, and M. Balduzzi. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. In *NDSS*, 2011.

[10] J. R. Binkley and S. Singh. An Algorithm for Anomaly-based Botnet Detection. In *USENIX SRUTI*, 2006.

[11] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian. Anomaly Extraction in Backbone Networks Using Association Rules. In *IMC*, 2009.

[12] M. J. Chapple, N. Chawla, and A. Striegel. Authentication Anomaly Detection: A Case Study on a Virtual Private Network. In *ACM MineNet*, 2007.

[13] H. Choi, H. Lee, H. Lee, and H. Kim. Botnet Detection by Monitoring Group Activities in DNS Traffic. In *IEEE CIT*, 2007.

[14] E. Cooke, F. Jahanian, and D. McPherson. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In *USENIX SRUTI*, 2005.

[15] G. Dewaele, K. Fukuda, P. Borgnat, P. Abry, and K. Cho. Extracting Hidden Anomalies Using Sketch and non Gaussian Multiresolution Statistical Detection Procedures. In *ACM SIGCOMM LSAD*, 2007.

[16] J. François, S. Wang, R. State, and T. Engel. BotTrack: Tracking Botnets Using NetFlow and PageRank. In *IFIP TC 6 Networking Conf.*, 2011.

[17] F. C. Freiling, T. Holz, and G. Wicherski. Botnet Tracking: Exploring a Root-cause Methodology to Prevent Distributed Denial-of-service Attacks. In *ESORICS*, 2005.

[18] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-independent Botnet Detection. In *USENIX Security*, 2008.

[19] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-driven Dialog Correlation. In *USENIX Security*, 2007.

[20] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *NDSS*, 2008.

[21] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and Detecting Fast-Flux Service Networks. In *NDSS*, 2008.

[22] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy. Studying Spamming Botnets Using Botlab. In *USENIX NSDI*, 2009.

[23] I. T. Jolliffe. *Principal Component Analysis.* Springer-Verlag, 1986.

[24] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale Botnet Detection and Characterization. In *USENIX HotBots*, 2007.

[25] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data. An Introduction to Cluster Analysis.* Wiley, 1990.

[26] J. Levine, R. LaBella, H. Owen, D. Contis, and B. Culver. The Use of Honeynets to Detect Exploited Systems Across Large Enterprise Networks. In *IEEE IAW*, 2003.

[27] C. Livadas, R. Walsh, D. Lapsley, and W. Strayer. Using Machine Learning Techniques to Identify Botnet Traffic. In *IEEE LCN*, 2006.

[28] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs. In *ACM SIGKDD KDD*, 2009.

[29] J. Nazario and T. Holz. As the Net Churns: Fast-flux Botnet Observations. In *MALWARE*, 2008.

[30] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi. FluXOR: Detecting and Monitoring Fast-Flux Service Networks. In *DIMVA*, 2008.

[31] R. Perdisci, I. Corona, D. Dagon, and W. Lee. Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces. In *ACSAC*, 2009.

[32] A. Ramachandran and N. Feamster. Understanding the Network-level Behavior of Spammers. In *ACM SIGCOMM*, 2006.

[33] A. Sperotto, R. Sadre, and A. Pras. Anomaly Characterization in Flow-Based Traffic Time Series. In *IEEE IPOM*, 2008.

[34] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *ACM CCS*, 2009.

[35] W. Strayer, R. Walsh, C. Livadas, and D. Lapsley. Detecting Botnets with Tight Command and Control. In *IEEE LCN*, 2006.

[36] R. Villamarín-Salomón and J. C. Brustoloni. Bayesian Bot Detection Based on DNS Traffic Similarity. In *ACM SAC*, 2009.

[37] A. Wagner and B. Plattner. Entropy Based Worm and Anomaly Detection in Fast IP Networks. In *IEEE WETICE*, 2005.

[38] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan. Detecting Algorithmically Generated Malicious Domain Names. In *IMC*, 2010.

[39] S. Yadav and A. N. Reddy. Winning With DNS Failures: Strategies for Faster Botnet Detection. In *SECURECOMM*, 2011.

[40] T. Yen and M. K. Reiter. Traffic Aggregation for Malware Detection. In *DIMVA*, 2008.

[41] J. Zhang, R. Berthier, W. Rhee, M. Bailey, P. Pal, F. Jahanian, and W. H. Sanders. Safeguarding Academic Accounts and Resources with the University Credential Abuse Auditing System. In *DSN*, 2012.