Insider Attack Detection for Science DMZs Using System Performance Data

Ross Gegan*, Brian Perry*, Dipak Ghosal*, Matt Bishop*
* University of California, Davis
{rkgegan, bperry, dghosal, mabishop}@ucdavis.edu

***Abstract*—The science DMZ is a specialized network model developed to guarantee secure and efficient transfer of data for large-scale distributed research. To enable a high level of performance, the Science DMZ includes dedicated data transfer nodes (DTNs). Protecting these DTNs is crucial to maintaining the overall security of the network and the data, and insider attacks are a major threat. Although some limited network intrusion detection systems (NIDS) are deployed to monitor DTNs, this alone is not sufficient to detect insider threats. Monitoring for abnormal system behavior, such as unusual sequences of system calls, is one way to detect insider threats. However, the relatively predictable behavior of the DTN suggests that we can also detect unusual activity through monitoring system performance, such as CPU and disk usage, along with network activity. In this paper, we introduce a potential insider attack scenario, and show how readily available system performance metrics can be employed to detect data tampering within DTNs, using DBSCAN clustering to actively monitor for unexpected behavior.**

***Index Terms*—Science DMZ, data transfer node (DTN), scientific workflows, system performance metrics, computer security, insider attack, anomaly detection, machine learning, DBSCAN, clustering.**

I. Introduction

Scientific research depends on the safe transfer of huge quantities of data, in some cases terabytes worth [1]. Research organizations use Science Demilitarized Zone (DMZ) networks, a network model designed to guarantee optimized and reliable transfers through performance tuning and efficient network organization, as well as custom built data transfer nodes (DTNs). This Science DMZ model enables higher performance and more reliable data transfers [2], and help connect research sites to each other as well as cloud computing resources. Science DMZs often avoid typical defense measures such as firewalls in order to optimize performance, instead using Access Control Lists (ACLs) and other forms of detection [3]. However, these defenses are insufficient for handling insider threats. Insider threats are a serious concern, as ensuring data integrity is critical to scientific research [4]. In particular, protecting data confidentiality and preventing data exfiltration are important concerns [4]. Therefore, monitoring for insider data tampering is important to provide the DTNs with extra protection, both to protect their performance and the integrity of the transferred data. DTNs can also be useful in contexts outside of scientific research, such as transfers between cloud service providers [5].

This material is based upon work supported by the National Science Foundation under Grant Number OAC-1739025.

This work examines a method of detecting insider attacks targeting the data stored on or moving through a DTN. Since the insider attack category can be broadly defined, we focus our efforts primarily on detecting data sabotage, considering a possible attack scenario involving SSH obfuscation. We consider a novel method of detecting data tampering which monitors the host performance data to detect file editing events, distinguishing between user file modification and the modification occurring as a result of the DTN's file transfers. The limited range of legitimate DTN operations [6] allows for effective anomaly detection. The anomaly detection method used for this utilizes DBSCAN clustering [7] of host metrics such as CPU utilization, along with checking disk writes and network activity. To recreate the DTN environment, we set up an experimental DTN using a server with a 10 Gbps backbone link to the UC Davis Science DMZ. Our experimental DTN follows the best practices described in ESnet's DTN tuning guide [8]. To emulate scientific data being transferred to the DTN, real traffic and DTN activity is generated by files continually transferred from the Energy Sciences Network (ESnet) test DTNs. As the DTN operates, we continually log and monitor system performance and network behavior, performing real-time monitoring by continually re-clustering the CPU activity and checking the disk and network activity.

The key contributions are as follows:

- We consider a new SSH obfuscation system using PDF files, describing how it could be exploited by insiders as part of the overall attack chain.
- We present a real-time detection method which can quickly identify unexpected file editing on the DTN using DBSCAN, as well as the obfuscated SSH sessions.
- We demonstrate the value of system performance metrics for anomaly detection on DTNs for protecting data integrity.

In Section II we provide background information on the Science DMZ, insider attacks, and system performance metrics, along with related work. In Section III we describe a potential attack scenario we have envisioned where an insider can establish an obfuscated SSH session on the

DTN, giving them the ability to sabotage data. Section IV we provide details on our detection system, in addition to some background on clustering and anomaly detection. In Section V we describe our experimental setup in-depth and in Section VI will discuss and evaluate our results. Finally, in Section VII we summarize our conclusions and discuss future work.

II. Background and Related Work

A. Science DMZ

The Science Demilitarized Zone (DMZ) is a model designed for scaling scientific research, ensuring reliable performance and high rate data transfers [2] between sites. Though the DMZs differ depending on their purpose, DMZs share certain key features. Figure 1 presents a typical science DMZ configuration and its components. A science DMZ is typically connected with a site at the network perimeter, through a border router linking the Science DMZ and the site. Assuming their security policies allow for it, multiple organizations can share access to their Science DMZs [6]. Therefore, the slower site or campus network gains access to the high performance resources of the Science DMZs, allowing high performance data transfers over the wide area network. A critical component is the data transfer node (DTN), which is dedicated to managing the efficient data transfers. For these transfers, the Science DMZ model prioritizes correctness, consistency, and performance, in that order [2]. Different security measures such as data encryption during transfers, and stateless firewalls controlling which DTNs are communicating, help to prevent data exfiltration [6].

The DTN connects directly to the Science DMZ router, serving as a high-performance server responsible for managing all of the incoming and outgoing data. As such, it is a critical component, and the security of the Science DMZ depends on protecting the DTN and its data. The DTNs do not typically run many applications, they are used almost solely for parallel data transfers (commonly performed using GridFTP [9]), along with some performance monitoring performed by tools like perfSONAR [10], and system maintenance [2], [6]. This simplicity not only improves the efficiency of file transfers, it also improves the security of the DMZ by allowing strict access control to be implemented, minimizing the attack surface. The Science DMZ model is flexible, meaning no two DTNs will be identical in terms of hardware and software [11], and more or less user access to the DTNs is possible depending on the security policy. For example, shell access might be restricted on the DTN in some cases [2]. However, the basic usage can be expected to remain similar across organizations. This is helpful to keep in mind when considering detection of attacks, as the range of normal and acceptable behavior is much more limited and predictable than a general purpose system. This narrower range of normal network and host activity makes practical anomaly detection based on system performance metrics in these environments more feasible [12]. Since access to the DTN is strictly controlled, insider attacks from within an organization might be the primary concern.

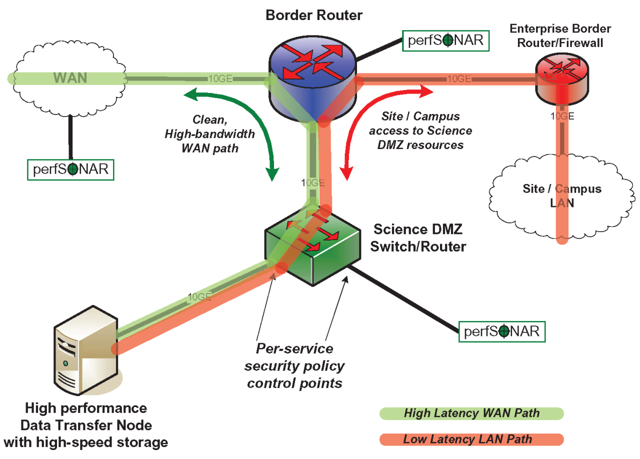

Border Router
perfSONAR
Enterprise Border Router/Firewall
WAN
10GE
Clean, High-bandwidth WAN path
Site / Campus access to Science DMZ resources
10GE
perfSONAR
10GE
Site / Campus LAN
Science DMZ Switch/Router
10GE
perfSONAR
Per-service security policy control points
High performance Data Transfer Node with high-speed storage
High Latency WAN Path
Low Latency LAN Path

Fig. 1: A typical Science DMZ (reprinted from [13]).

B. Insider Attacks

An insider attack can describe any case where the attack is performed by somebody with legitimate access to a system [14]. Clearly, this encompasses a wide range of attack types, from data sabotage and leaking to blackmail and fraud. Many different insider attack taxonomies have been created [15] [16]. Generally speaking, we need to first consider the insider's profile, whether they are intentionally malicious or if they are unintentionally causing damage, if they are masquerading as authorized or legitimately authorized, and what is their intended role within the system [15]. The attacks also vary in terms of the level of sophistication and the attacker's knowledge of the target, their personal motivation and goals. We need to consider the scope and targets of the attack - are they attacking the network, the operating system, applications, or stored data? In some cases, the attacks will occur across multiple levels but become more noticeable on a particular level. For instance, data tampering is noticeable at the application or data level, while exfiltration can be observed at the network level [15].

Figure 2 shows the insider attack chain, and some actions that might be taken during the different steps of an attack. Attackers can be classified into one of three categories depending on their actions and role in the chain of an attack [15]. Masqueraders perform reconnaissance and imitate legitimate users to set up an attack, while traitors and unintentional perpetrators execute the attack through extracting data, sabotaging data or some other part of the overall system. In our experiments, we focus primarily on the "actions on objectives" portion of the attack sequence, looking at data tampering detection in the DTN environment. However, in Section III, we will consider an SSH obfuscation method which could be used

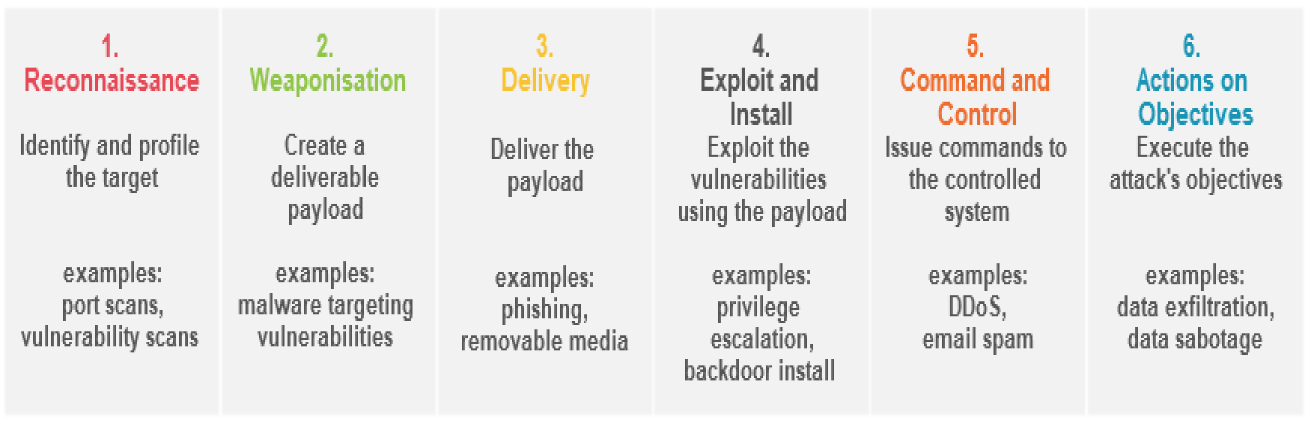

1. Reconnaissance
Identify and profile the target
examples: port scans, vulnerability scans
2. Weaponisation
Create a deliverable payload
examples: malware targeting vulnerabilities
3. Delivery
Deliver the payload
examples: phishing, removable media
4. Exploit and Install
Exploit the vulnerabilities using the payload
examples: privilege escalation, backdoor install
5. Command and Control
Issue commands to the controlled system
examples: DDoS, email spam
6. Actions on Objectives
Execute the attack's objectives
examples: data exfiltration, data sabotage

Fig. 2: The cyber attack chain [15]. In the first five steps, the insider acts as a ”masquerader” to gain control of the system. The final actions can also be executed by a traitor or an unintentional perpetrator within the organization, skipping the initial steps.

as part of a full insider attack chain. This method can be used to conceal the SSH protocol, making the exchange of protocol related packet data within PDF files. To a network IDS monitoring the DTN, this activity would appear as normal file transfers. We will explain its functioning at a high level, and consider how we can detect the obfuscated SSH sessions before any data tampering occurs.

As expected from the wide variety of attacks, there are many categories of insider attack defense. Our focus is on the ”Detection and Assessment” branch of defense [15], in particular we consider anomaly-based detection and unsupervised detection of insider attacks. In this paper, one of our goals will be to try detecting insider threats on the DTN using novel data sources and techniques. Therefore, we will try to detect insider threats using host performance data such as CPU usage and disk writes, applying DBSCAN clustering [17] to create a detection scheme well-suited for the DTN environment.

C. *Related Work*

Detecting insider attacks is a very broad research area, considering a range of topics based on the types of attackers and the form of the attacks. For further reading, Homoliak, Ivan, et al. [16] provides a survey of the large variety of taxonomies for insider attacks, covering taxonomies both for attacks and defense techniques. Liu et al. [15] provides another look at insider attack taxonomies. A CERT guide [14] gives an in-depth description of insider attacks and best practices for mitigating or eliminating them. In our case, the focus is primarily on data sabotage by an insider who at one point was granted legitimate access to a system, and using machine learning to detect that anomalous behavior. A number of different data sources have been used for this form of insider detection [16]. However, using performance data for insider threat detection like our method appears less common.

Some recent papers have been written on securing Science DMZs specifically. Nagendra et al. [3] introduces a tool called SciMon, designed to protect Science DMZ DTNs. Machine learning based anomaly detection appears uncommon, and these papers do not consider system performance metrics for detection. However, these metrics have been applied to detecting insider attacks in other contexts. Nikolai et al. [18] describes a method of detecting insider data theft in IaaS cloud environments using a mixture of system metrics such as CPU and memory usage and network metrics such as the number of network bytes sent or received. Oppermann et al. [19] describes how a simulated insider attack in a cloud environment can be detected by monitoring CPU usage along with network traffic.

III. Data Obfuscation Attack Scenario

In order to establish how an insider might gain the ability to tamper with data on the DTN, we consider one potential attack scenario using data obfuscation. This can be considered as the ”Masquerader” step shown in Figure 2. We make the assumption that the attacker is an insider who at one point had been granted legitimate access to the system, and the privileges necessary to modify data on the DTN. In this case, it is possible that the insider could setup a backdoor, allowing them to continue remotely modifying the DTN files after officially losing access. However, the attacker would like to avoid being caught running an SSH session, which might be blocked by the DTN [2]. Therefore, we consider a method of SSH obfuscation which could be used to remotely execute the attack. By hiding the raw SSH data in another file type, we can have a hidden SSH session. Many types of files could be used, including image files. In our case, the SSH sessions are obfuscated by hiding SSH data within PDF files.

This PDF obfuscation method involves creating a stealth SSH connection by concealing the protocol data within PDF files before transmitting on the network. If both the sender and receiver are able to decode the obfuscation, a covert connection can be setup. Assuming the network IDS does not perform deep packet inspection to block packets containing PDF data, the SSH session with the target will appear as normal file transfers over GridFTP. Since we know DTNs do not perform this type of check, and GridFTP file transfers are the expected behavior, it is likely this method would allow the attacker to establish a stealth SSH session, giving them the capability to execute their attack.

The basic functioning of the PDF obfuscation method is shown in Figure 3. The obfuscation server is setup on the DTN, while the insider is running the client on their own machine. As the endpoints exchange packets, the obfuscation program checks if the packet contains obfuscated data. If it does, then it will deobfuscate the data by inserting it into a PDF and reading from it, sending that raw data to the original application - SSH in this case. If the packet does not contain obfuscated data, the program inserts the raw SSH protocol data into the PDF to obfuscate it, before sending it through the tunnel. By masquerading as legitimate traffic, the insider gains the ability to perform their tampering on the DTN while evading detection by network intrusion detection systems. Between this and tampering with user logging, the insider could evade detection through traditional means. Therefore, alternative means of detection such as monitoring system performance metrics could improve upon traditional insider attack defense.

Regardless of the precise method used to gain or maintain access, the attacker has the ability to modify data, putting any files stored on the DTN during the transfer process at risk. The following section describes our method of detecting file editing on the DTN, and how file editing can be distinguished from the disk writes caused by standard DTN transfers.

IV. Data Sabotage Detection

Previous work has demonstrated how certain host performance metrics can be linked to network activity, and can be used to detect some insider attacks [19]. In addition, logs of host activity (system calls, user command histories) and network data are common data sources used to perform anomaly detection for different forms of insider attacks [15]. However, detection schemes relying on forensic logging can be hampered if the attacker is able to modify these logs [20], and interpreting the log data can be difficult or time-consuming [21]. Therefore, we want to consider additional data sources. Normally, as a dedicated component of the Science DMZ, a DTN is expected to perform only a narrow range of tasks, mostly related to moving data and performance monitoring. The performance metrics can be expected to remain within a consistent range, which allows us to more easily predict typical system performance [2] [13]. Therefore, we take advantage of the difference in system performance during normal activity and file modification to help detect unexpected file editing events which might occur during an insider attack, as well as detecting obfuscated SSH sessions.

A. Performance Metrics Monitored

In order to achieve this, we monitored host performance metrics stored in procfs - **system CPU usage**, **user CPU usage**, **disk writes**, **interrupts**, and **context switches**. Figure 6 shows how the performance metrics change over the course of a 10GB file transfer to the DTN, and figure 7 shows how the CPU changes during file editing. Standard file editing increases CPU usage noticeably, along with the obvious spike in the data written to the disk. However, a large file transfer to the DTN causes a similar increase in total CPU usage and disk writes which can mask the file editing when the two events overlap, and a clever attacker could arrange for editing to coincide with large transfers. Fortunately, the effects on CPU usage are distinguishable by looking at the user and system CPU usage. Network file transfers increased the system CPU (often significantly weighted on one core), while the user CPU usage remains stable. Meanwhile, if the files are large enough, file editing will noticeably increase the user CPU usage while the system CPU usage remains stable, as shown in figure 8. Based on this, CPU usage is the primary means used for detecting file editing events.

B. Clustering Performance Metrics

To actively monitor these performance metrics, we continually cluster the user CPU usage using **DBSCAN** (density-based spatial clustering of applications with noise), a widely used clustering algorithm [22]. Although other clustering methods could also be applied, this particular clustering algorithm was selected for anomaly detection because it is specifically designed for clustering noisy data [17]. This is necessary when monitoring network and system performance metrics, which will have noticeable noise even in a relatively predictable environment. As a form of unsupervised learning, DBSCAN allows us to distinguish between normal and unusual performance data without performing training or using predefined threshold values.

Although an in-depth comparison of clustering methods is beyond the scope of this paper, it is worth noting that DBSCAN is simple and flexible compared with other classic clustering methods such as k-means clustering. Any cluster shape is possible, and there is no need to define the number of clusters beforehand. The only necessary parameters are the maximum distance between the points within a cluster, ϵ, and the minimum points required to form a cluster, **MinPts**. There are three types of points - core points, border points, and noise. Core points are within ϵ distance of two or more points, while border points are within ϵ distance of just one other point. Noise points are not within ϵ of any other points. If at least **MinPts** points are connected as core or border points, then that becomes a cluster.

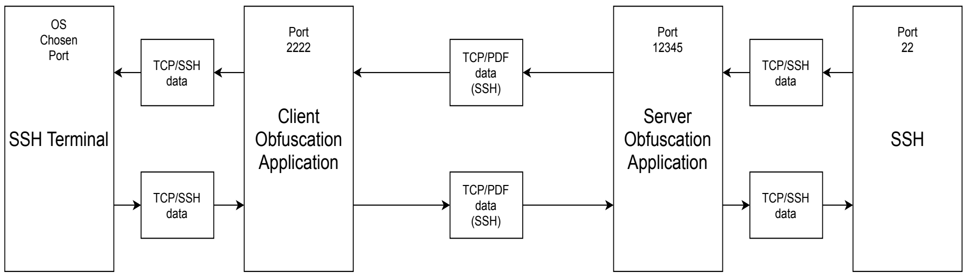


Fig. 3: PDF obfuscation overview. SSH protocol data is hidden within PDF data sent over the network. This obfuscation method is one possible method an insider could use to stealthily establish an SSH session and perform data sabotage on a DTN. Although we chose to use PDF files, the SSH data could be placed into other data types as well, such as image files.

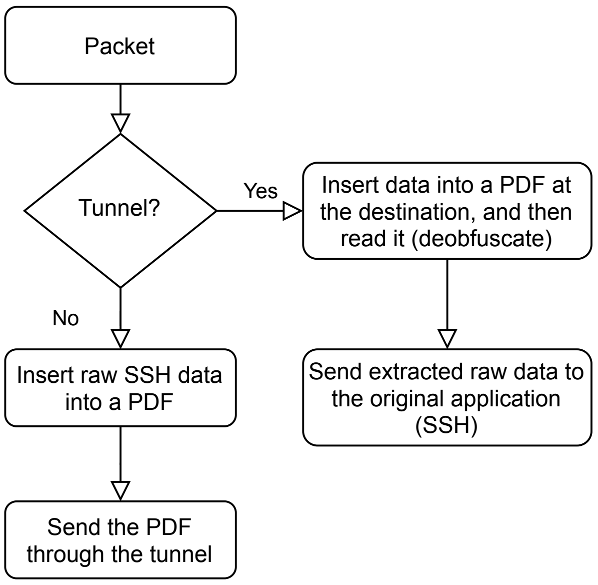

Packet
Tunnel?
Yes
Insert data into a PDF at the destination, and then read it (deobfuscate)
Send extracted raw data to the original application (SSH)
No
Insert raw SSH data into a PDF
Send the PDF through the tunnel

Fig. 4: PDF obfuscation tunneling method. The PDF obfuscation program either places the raw SSH protocol data into a PDF before sending it through the tunnel, or extracts the raw protocol data arriving through the tunnel. The tunneled data (PDF data containing obfuscated SSH data) is inserted into a PDF at the destination. The obfuscation application then extracts the raw SSH data and sends it to SSH.

The relative simplicity and ability to handle noise well also allows us to easily cluster data in real-time. Using Python, we created two scripts which continually create new small clusters every 10 seconds using the per-second data we gather. The first script clusters **user CPU usage** to detect unexpected file modification, while the second script clusters **disk writes** to detect obfuscated SSH sessions. With ϵ=4 and **MinPts**=4, the baseline user CPU usage values will be placed into the same cluster. However, during a file editing event, outliers or additional clusters of higher values will appear. Once this is detected, we check if the disk writes are greater than 1MB. If outliers or extra clusters appear alongside disk writes, we report an anomaly indicating file editing. Figure 5 shows the clustering results when 10MB of file editing coincides with a 10GB file arriving on the DTN.

C. *Machine Learning for Anomaly Detection*

Clustering and other forms of machine learning-based anomaly detection are effective at detecting new behavior which does not match the expected data patterns [23]. Therefore, machine learning is often applied towards predictive classification problems. However, anomaly detection based on machine learning must address a classification problem, defining properly what is ”normal” and what is ”abnormal”. Sommer et al. [24] discusses the necessity of avoiding *closed world assumptions* when applying anomaly detection based on machine learning. A *closed world assumption* is defined by Witten et al. [12] as *the idea of specifying only positive examples and adopting a standing assumption that the rest are negative is called the closed world assumption*. Sommer et al. argues that in many cases where machine learning is used for anomaly detection, there is too broad of a scope and that anomalies are inappropriately considered attacks by default (the semantic gap), leading to excessive false positives. Anomaly detection is better applied towards detecting known attacks versus novel ones. In our experiments, we attempt to bridge the semantic gap by carrying out normal baseline activity alongside the attack in order to establish a ground truth. In addition, we focus on a narrow scope - the limited functionality of DTNs results in more predicatable activity. Therefore, machine learning is suitable for anomaly detection in this context. Furthermore, although we did not have access to campus-level DTN data, we generate real data using a test DTN, emulating scientific workflows by receiving data from other test DTNs. Our experimental setup is discussed in the following section.

V. Experimental Setup

A. *DTN Testbed*

To simulate a real Science DMZ data transfer node (DTN), we setup an experimental system acting as a DTN, a PowerEdge T630 server which we refer to as D. D has a RAID-10 set of 8 1TB 7.2K RPM SATA 6 Gbps hard drives, 32GB 2133MT/s RDIMM memory capacity, and contains two Intel Xeon E5-2637 v3 3.5GHz processors. Just like our campus DTN, D is connected to a 100 Gbps wide-area network, called CENIC, through a 10 Gbps

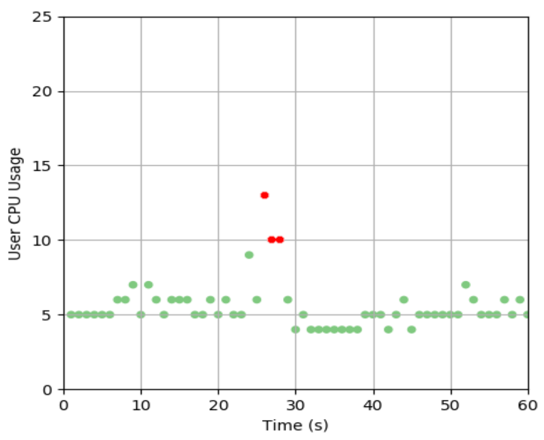


Fig. 5: DBSCAN clustering of the user CPU usage during a 10MB file editing event coinciding with a 10GB file transfer. Although the CPU usage fluctuates, it will not normally increase beyond a certain value during a transfer. If the user CPU usage spikes without being clustered, this suggests unusual user activity. The red spike coinciding with disk writes indicates file editing, while the green represents ordinary CPU usage.

backbone link. D continually requests and receives data from three different Energy Science Network (ESnet) test DTNs through Globus GridFTP. To simulate the behavior of an ordinary DMZ, D randomly requests different sizes of files from these test DTNs. These file transfers continue over the course of the day. Table I shows the sizes of the files requested from the test DTNs. To simulate the different distributions seen on a real NERSC DTN [25], we have two potential file size distributions, with an equal value for the total expected data received (750 GB per day). The first case is a large number of small files, while the second case is a small number of large files.

TABLE I: GridFTP transfer distributions

Distribution	Potential File Sizes	Interval
Normal	10M, 50M, 100M, 1G, 10G, 50G	1-30 minutes
Large	10G, 50G	60-75 minutes
Small	10M, 50M, 100M, 1G	5-40 seconds

B. Data Sabotage

Through the PDF obfuscation method discussed in Section III, the attacker can establish an SSH session with the target, while hiding the SSH session to secretly execute the data tampering script. Once the attacker gains access, either through masquerading or legitimate access, they gain the ability to modify data. The specific nature of the data is not significant. We choose to use `tstat` logs, which are commonly stored during network monitoring [26]. Using familiar data, we can make assumptions about how an attacker might want to alter the data set. The attacker might want to completely destroy the stored data, or they might want to selectively alter the data by changing the entries in one field. Selectively changing the data could be useful, allowing the attacker to extract the legitimate data sets while sabotaging the data left on the system. Modifying particular data fields could also be employed to manipulate research results, in the case of science data. Therefore, we consider different cases of data tampering in our experiments, by varying the degrees to which the data is altered. A knowledgeable attacker, familiar with the data, might only need to alter a few lines to get their desired result. Alternatively, they might alter large portions of the dataset. The difficulty of detecting these changes depends on the amount of files changed, but also on distinguishing between the attacker's file modifications and the changes caused by the file transfers. The nature of the DTN implies that files will not normally be modified outside of file transfers. However, the attacker could potentially send traffic while the data sabotage script runs, or, if the attacker is aware of when file transfers are occurring, they might set their script to coincide with transfers. In addition, some log files could be routinely written on the DTN for monitoring [6].

C. Detection

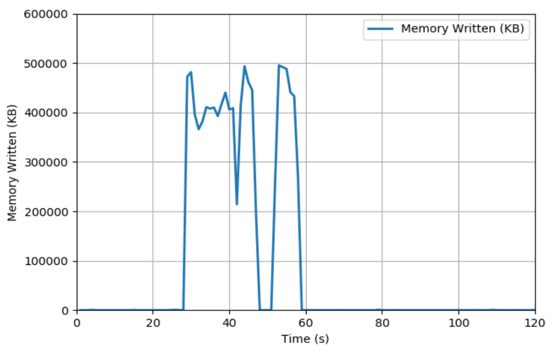
We consider different sources of data in our experiments. Host and network performance data is gathered from **procfs**, as well commonly available tools such as **collectl** [27]. In addition to logging these values over a 24 hour period, we created two Python scripts to cluster the data in real time - one script for detecting file modification, and another script for detecting obfuscated SSH sessions. Each scripts works by gathering 10 seconds worth of per-second data, clustering it using DBSCAN clustering [7] to look for anomalies, then repeating. We cluster **disk writes** to detect obfuscated SSH sessions, and cluster **user CPU usage** to detect file modification. All of the points should cluster under normal conditions, meaning an anomaly is detected when a second cluster appears. When monitoring for file detection, we also considered noise outside the baseline cluster as an anomaly, since user CPU usage remains within a small range under normal conditions.

VI. Results and Evaluation

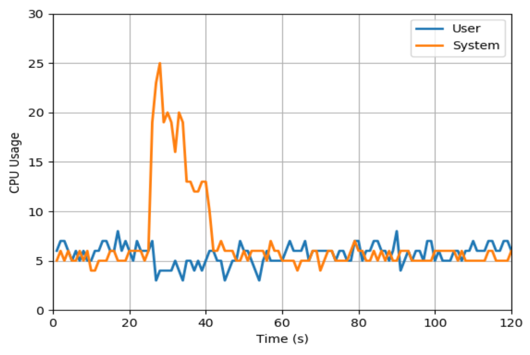
We considered two insider attack cases - **data sabotage** through file editing, and **data exfiltration** through obfuscation. By monitoring the performance metrics and taking into account network activity, we can detect both of these cases. We will discuss the various performance metrics we measured, and how they were affected by file transfers, file editing, and PDF obfuscation. In addition, we will explain our detection results and some limitations of our approach.

A. Data Sabotage

The most obvious impact on the performance metrics during file editing will be an increase in disk reads and writes, and increased CPU usage. Figure 8a. demonstrates the effects file editing had CPU had on user and system CPU usage. Other metrics, such as interrupts and context



(a) Data written to disk



(b) User and System CPU usage

Fig. 6: Host and network performance during a 10GB GridFTP transfer. Interrupts and context switches do not increase significantly. CPU usage, primarily system CPU usage, increases noticeably.

switches, did not increase significantly while observing file editing. Figure 7 shows how the user CPU usage, representing the time spent on user level processes, increases significantly as the amount of overwritten data increases, though it stabilizes around 30MB. Meanwhile, the system CPU usage (the time spent on kernel tasks) only slightly increases, remaining relatively stable regardless of increased disk writes. By observing increased disk writes coinciding with increased user CPU usage, we can easily identify a file editing event.

This is more complicated while file transfers are occurring. Small file transfer events don't impact the disk writes or CPU usage significantly enough to affect this detection. However, larger file transfers similarly cause large spikes in both CPU usage and disk activity. Figure 6 demonstrates the effect of a 10GB file transfer to D using GlobusFTP on various performance metrics over the course of two minutes. Since there are generally few programs running on the DTN, the baseline CPU and disk activity on the DTN remains low. When the transfer occurs around 20 seconds, we see a large spike in data being written to the disk and CPU usage, as expected. Other metrics like interrupts and context switches increase slightly during a large file transfer, and more than during file editing, but not significantly enough to be useful for detection. Therefore, these transfers can produce similar activity and it's conceivable that an attacker might attempt concealing file tampering by editing files while a large file transfer is ongoing.

In order to detect data tampering while a large file transfer event is occurring, we need to consider how the host performance differs. The most obvious difference is the insider must edit the files through user processes, meaning user CPU usage will increase noticeably during these file editing events. We apply DBSCAN to cluster normal user CPU activity and detect outliers. Figure 5 shows how the clustering appears when the `tstat` files are edited during a 10GB file transfer. When a small amount of file data is modified, short spikes in the user CPU usage will create outliers, while normal variations form the largest, primary cluster. Sustained periods of file editing will form a smaller clusters above the primary cluster. The appearance of either outliers or an unexpected cluster, combined with a spike in disk reads and writes, indicates files are being overwritten. Clustering can reliably identify on-going data modification, even if it coincides with a large GlobusFTP transfer.

Although the detection is effective, it is unable to detect small file modifications which do not significantly increase the CPU usage. Looking at figure 7, we see that if the total data overwritten is below 10MB, we cannot reliably detect that event, because it falls within the normal performance range. Therefore, it is possible an attacker could evade detection by only editing small portions of data at a time. Future work should consider what additional metrics could be leveraged to detect this form of data tampering. However, most of the science data arriving on the DTN is likely to be in the form of large files, and modifying these files will necessitate disk activity above the threshold for detection.

B. *Data Exfiltration*

Data exfiltration can be performed through ordinary GridFTP transfers, or by sending data through an obfuscated channel. In the first case, it is likely that the transfer could be detected by ordinary security measures, such as Zeek network intrusion detection. However, if it is being leaked through obfuscated protocols, an insider could extract the data while evading detection. Therefore, we must be able to detect the PDF obfuscated SSH sessions.

The PDF obfuscation method has a clear impact on the system, because communication between the client and server depends on frequent writes to the PDF files. Figure 9a shows how the disk writes increase during the obfuscated SSH session. Figure 9b shows how this increase in disk writes appears when clustered. The consistently higher disk writes per second leads to new clusters being formed in addition to the primary cluster representing the baseline disk writes. During periods with no file transfers to the DTN, this is simple to detect. However, files transfers could also cause sustained spikes in disk writing. If an attacker can ensure their activity on the DTN coincides with these transfers, detection could become more difficult. We can achieve more reliable detection by considering the ratio of disk writes to the amount of data written per second. Since the PDF obfuscation program writes only small amounts of data to the disk each time, the ratio will be much larger than during most file transfers.

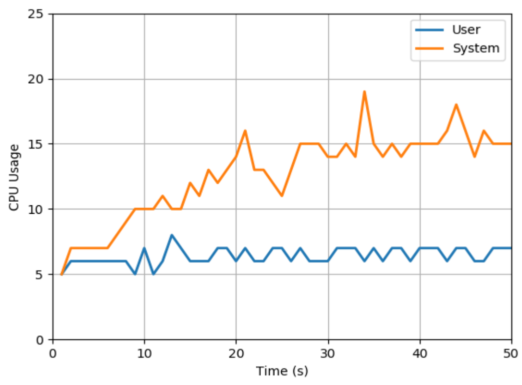
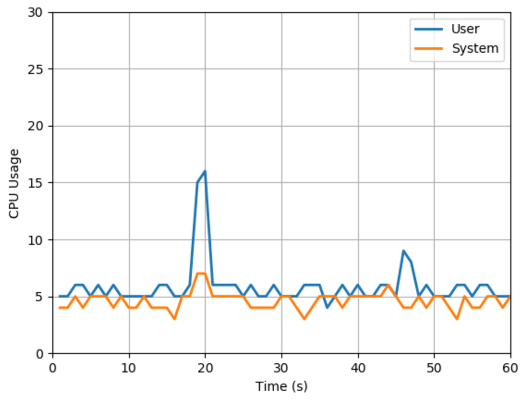


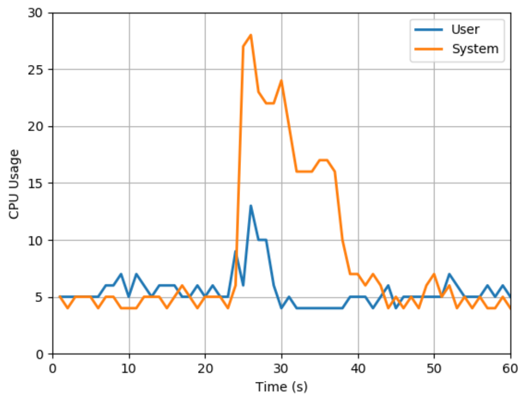
Fig. 7: Change in CPU usage during file editing events. We can see that user CPU usage increases as more file data is overwritten, while system CPU usage remains relatively stable.

C. Real-Time Detection

Since the user CPU usage and disk writes remain within a small enough range under normal conditions, we do not need a large sample size to identify file modification or the obfuscated SSH channel. In our experiments, we found that 10 data points was sufficient to detect these anomalies through clustering. Therefore, DBSCAN can use a low **MinPts** values, and the detection can be performed quickly. After collecting 10 seconds worth of data, the values are clustered and checked for anomalies. If noise points or more than one cluster appears, we report an anomaly. This method was able to reliably identify file modification without false positives past 10MB, where the increase in user CPU usage becomes large enough to form outliers and new clusters (Figure 7). By clustering the ratio of disk writes to data written to disk, we could also reliably identify an on-going obfuscated SSH session. The obfuscated SSH session performs more disk writes when characters are being typed, which will lead to a new small cluster being formed. If two or more total clusters appear, we can identify the anomaly. Sending three characters at a normal pace in the obfuscated SSH session is sufficient to create a new cluster and trigger the alert. Therefore, the operations the attacker can perform using the obfuscated SSH session are severely limited, and exfiltrating large amounts of data through the obfuscated channel is no longer feasible.



(a) CPU activity during data editing

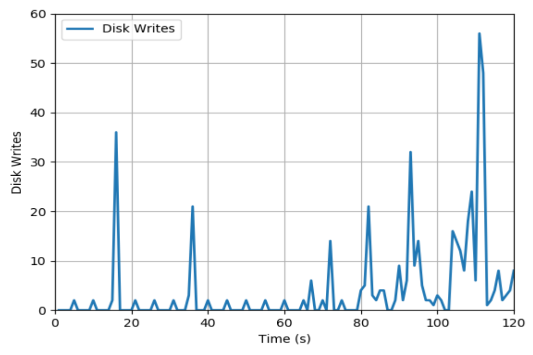


(b) CPU activity during data editing alongside file transfer

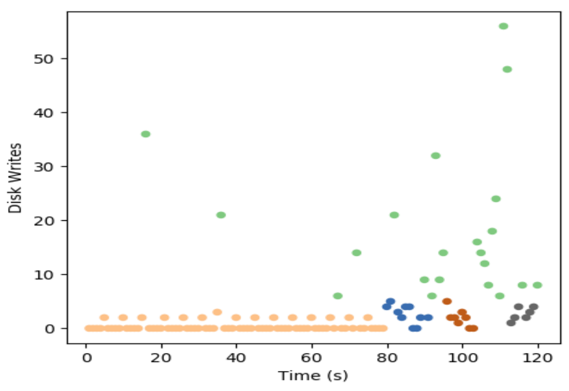
Fig. 8: CPU Usage over the course of editing 50MB worth of files. The spike in user CPU usage is visible even if the editing occurs during a large data transfer.

VII. Conclusions and Future Work

This work suggests that system performance metrics such as CPU usage and disk writes, along with network performance metrics such as the amount of incoming traffic, can be used together to help identify unwanted data modification in a DTN environment. The limited range of applications and predictable system performance of the DTN environment allows clustering based anomaly detection using DBSCAN to function effectively. Using DBSCAN clustering to detect abnormal CPU activity, along with checking for coinciding disk and network activity, we can predict when unexpected file editing is occurring and distinguish it from a large file transfer, even if the insider has taken efforts to conceal it from other means of detection. Furthermore, this detection can be used in real-time by repeatedly clustering the performance metrics gathered by common tools such as `collectl`.



(a) Disk writes over time



(b) Disk write clusters

Fig. 9: Disk write activity can detect an obfuscated SSH session. The obfuscated SSH session begins at 80 seconds. Although this obfuscation method can evade traditional detection methods, the frequent writes to PDF files can be used for detection. The three new small clusters appearing after 80 seconds indicate obfuscated SSH activity.

In the future, this form of detection could be enhanced by considering additional performance metrics, as well as combining it with other methods of insider attack detection, such as system call monitoring.

References

[1] J. Crichigno, E. Bou-Harb, and N. Ghani, "A comprehensive tutorial on Science DMZ," *IEEE Communications Surveys & Tutorials*, 2018.

[2] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The science dmz: A network design pattern for data-intensive science," *Scientific Programming*, vol. 22, no. 2, pp. 173–185, 2014.

[3] V. Nagendra, V. Yegneswaran, and P. Porras, "Securing ultra-high-bandwidth science dmz networks with coordinated situational awareness," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pp. 22–28, ACM, 2017.

[4] P. J. Hawrylak, G. Louthan, J. Hale, and M. Papa, "Practical cyber-security solutions for the science dmz," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*, pp. 1–6, 2019.

[5] W. Hong, J. Moon, W. Seok, and J. Chung, "Enhancing data transfer performance utilizing a dtn between cloud service providers," *Symmetry*, vol. 10, no. 4, p. 110, 2018.

[6] S. Peisert, E. Dart, W. Barnett, E. Balas, J. Cuff, R. L. Grossman, A. Berman, A. Shankar, and B. Tierney, "The medical science dmz: a network design pattern for data-intensive medical science," *Journal of the American Medical Informatics Association*, vol. 25, no. 3, pp. 267–274, 2018.

[7] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.

[8] E. Lawrence Berkeley National Lab, "DTN tuning," 2019.

[9] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The globus striped gridFTP framework and server," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, p. 54, IEEE Computer Society, 2005.

[10] PerfSONAR, "PerfSONAR," 2019.

[11] Y. Qin, A. Simonet, P. E. Davis, A. Nouri, Z. Wang, M. Parashar, and I. Rodero, "Towards a smart, internet-scale cache service for data intensive scientific applications," in *Proceedings of the 10th Workshop on Scientific Cloud Computing*, pp. 11–18, 2019.

[12] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[13] S. Peisert, W. Barnett, E. Dart, J. Cuff, R. L. Grossman, E. Balas, A. Berman, A. Shankar, and B. Tierney, "The medical Science DMZ," *Journal of the American Medical Informatics Association*, vol. 23, no. 6, pp. 1199–1201, 2016.

[14] D. M. Cappelli, A. P. Moore, and R. F. Trzeciak, *The CERT guide to insider threats: how to prevent, detect, and respond to information technology crimes (Theft, Sabotage, Fraud)*. Addison-Wesley, 2012.

[15] L. Liu, O. De Vel, Q.-L. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1397–1417, 2018.

[16] I. Homoliak, F. Toffalini, J. Guarnizo, Y. Elovici, and M. Ochoa, "Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures," *ACM Computing Surveys (CSUR)*, vol. 52, no. 2, pp. 1–40, 2019.

[17] M. Hahsler, M. Piekenbrock, and D. Doran, "dbscan: Fast density-based clustering with r," *Journal of Statistical Software*, vol. 25, pp. 409–416.

[18] J. Nikolai and Y. Wang, "A system for detecting malicious insider data theft in iaas cloud environments," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2016.

[19] A. Oppermann, F. G. Toro, F. Thiel, and J.-P. Seifert, "Anomaly detection approaches for secure cloud reference architectures in legal metrology.," in *CLOSER*, pp. 549–556, 2018.

[20] D. M. Cappelli, A. P. Moore, and E. D. Shaw, "A risk mitigation model: Lessons learned from actual insider sabotage," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2006.

[21] M. Bishop, D. Gollmann, J. Hunker, and C. W. Probst, "08302 abstracts collection–countering insider threats," in *Dagstuhl Seminar Proceedings*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

[22] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, p. 19, 2017.

[23] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[24] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy*, pp. 305–316, IEEE, 2010.

[25] A. Giannakou, D. Gunter, and S. Peisert, "Flowzilla: A methodology for detecting data transfer anomalies in research networks," in *2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*, pp. 1–9, IEEE, 2018.

[26] A. F. M. M. M. Meo, M. Munafo, and D. Rossi, "10-year experience of internet traffic monitoring with tstat," 2020.

[27] M. Seger, "Collectl," 2014.