



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

UNSUPERVISED LEARNING OF NETWORK TRAFFIC BEHAVIORS FOR INSIDER THREAT DETECTION

by

Brett L. Rajchel

September 2020

Thesis Advisor:

Vinnie Monaco

Second Reader:

Gurminder Singh

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2020	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE UNSUPERVISED LEARNING OF NETWORK TRAFFIC BEHAVIORS FOR INSIDER THREAT DETECTION			5. FUNDING NUMBERS	
6. AUTHOR(S) Brett L. Rajchel				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) Insider threats are a costly and dangerous problem for government and non-government organizations alike. Considering an insider's inherently privileged level of access on a network, the main principle of network defense—keep potential threats and outsiders out—does not apply to insider threats. Current defenses are largely based on the detection of insider threat indicators and rely on up-to-date datasets. However, insider threat activity is innumerable and as diverse as human behavior itself. We hypothesize that characterizing and examining host and organization behavior demonstrated on a network presents an opportunity to circumvent this problem. Leveraging machine learning to detect behavioral anomalies that indicate the presence of an insider threat would enable network administrators to quickly locate and mitigate such threats before they cause serious damage. We demonstrate this methodology by developing a system that extracts host and organization behavior in three different ways from network traffic and uses population-relative metrics to determine host conformity with organizational norms. After testing the system on an operational network with over 8,000 hosts, we show through a series of case studies that our system is effective in detecting behavioral anomalies and that our behavior extraction methods are complementary.				
14. SUBJECT TERMS machine learning, network traffic analysis, cybersecurity, insider threat, anomaly detection			15. NUMBER OF PAGES 139	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**UNSUPERVISED LEARNING OF NETWORK TRAFFIC BEHAVIORS
FOR INSIDER THREAT DETECTION**

Brett L. Rajchel
Lieutenant Commander, United States Navy
BS, U.S. Naval Academy, 2010

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
September 2020**

Approved by: Vinnie Monaco
Advisor

Gurminder Singh
Second Reader

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Insider threats are a costly and dangerous problem for government and non-government organizations alike. Considering an insider’s inherently privileged level of access on a network, the main principle of network defense—keep potential threats and outsiders out—does not apply to insider threats. Current defenses are largely based on the detection of insider threat indicators and rely on up-to-date datasets. However, insider threat activity is innumerable and as diverse as human behavior itself. We hypothesize that characterizing and examining host and organization behavior demonstrated on a network presents an opportunity to circumvent this problem. Leveraging machine learning to detect behavioral anomalies that indicate the presence of an insider threat would enable network administrators to quickly locate and mitigate such threats before they cause serious damage. We demonstrate this methodology by developing a system that extracts host and organization behavior in three different ways from network traffic and uses population-relative metrics to determine host conformity with organizational norms. After testing the system on an operational network with over 8,000 hosts, we show through a series of case studies that our system is effective in detecting behavioral anomalies and that our behavior extraction methods are complementary.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	CONTEXT AND MOTIVATION.....	1
B.	PROBLEM STATEMENT	3
C.	OBJECTIVES AND SCOPE	3
D.	THESIS OUTLINE.....	3
II.	BACKGROUND AND PREVIOUS WORK.....	5
A.	INSIDER THREATS DETECTION CHALLENGES.....	5
B.	DETECTING ANOMALIES.....	6
C.	NETWORK TRAFFIC ACTIVITY	8
1.	Human Behavior Revealed in Network Traffic	9
2.	Detecting Behavior Anomalies in Network Traffic.....	10
D.	MACHINE LEARNING	13
1.	Machine Learning Challenges	14
2.	Machine Learning Algorithms.....	15
3.	Online and Offline Machine Learning.....	18
E.	SUMMARY	19
III.	METHODOLOGY AND SYSTEM DESIGN.....	21
A.	INTRODUCTION.....	21
B.	METHODOLOGY	22
1.	Extracting Behavior from Network Traffic	22
2.	Detecting Behavior Anomalies.....	27
C.	SUMMARY	34
IV.	SYSTEM IMPLEMENTATION.....	35
A.	INTRODUCTION.....	35
B.	NETWORK TRAFFIC PROCESSING AND BEHAVIOR EXTRACTION	36
C.	CLUSTERING NETWORK HOSTS AND ANOMALY DETECTION.....	38
D.	SUMMARY	41
V.	RESULTS AND ANALYSIS	43
A.	OVERVIEW	43
1.	Host Analysis	43
2.	Cluster Analysis	49

B.	CASE STUDIES.....	54
1.	Time-of-Day Feature Vectors – Host A	55
2.	Time Interval Feature Vectors – Host B.....	60
3.	Country Feature Vectors – Host C.....	65
C.	DISCUSSION	69
VI.	CONCLUSION AND FUTURE WORK	73
A.	SUMMARY	73
B.	CONCLUSIONS	74
C.	FUTURE WORK.....	75
1.	System Efficiency	75
2.	Network Activity Timeframes	76
3.	Population Activity Patterns.....	76
4.	Host Activity Patterns.....	77
	APPENDIX A. VECTORIZE() COMPONENT	79
	APPENDIX B: CLUSTERING() COMPONENT	89
	APPENDIX C: TIME-OF-DAY CLUSTER ANALYSIS.....	105
	APPENDIX D: TIME INTERVAL CLUSTER ANALYSIS.....	109
	APPENDIX E: COUNTRY CLUSTER ANALYSIS	111
	LIST OF REFERENCES.....	113
	INITIAL DISTRIBUTION LIST	119

LIST OF FIGURES

Figure 1.	Data sources for insider threat detection. Source: [1].....	13
Figure 2.	Temporal measurement for human behavior	24
Figure 3.	Temporal and volume measurements for human behavior	24
Figure 4.	Time-of-day feature vectors describing human behavior	24
Figure 5.	Temporal and volume measurements for device behavior	25
Figure 6.	Time interval feature vectors describing device behavior	26
Figure 7.	Country feature vectors describing human-to-device behavior	27
Figure 8.	Preliminary analysis showing host cluster membership	31
Figure 9.	Cluster analysis example.....	31
Figure 10.	Cluster analysis dataset with sample silhouette scores	32
Figure 11.	Markov chain network host cluster transitions	33
Figure 12.	Population-relative and host-relative metrics example.....	34
Figure 13.	System components and functionality	35
Figure 14.	Processing network traffic in parallel	36
Figure 15.	Intermediary TCP and UDP packet representations	37
Figure 16.	Host activity	44
Figure 17.	Average population host interevent analysis	45
Figure 18.	Average population host destination port usage	45
Figure 19.	Clusters occupied CDF	46
Figure 20.	Cluster hops CDF.....	47
Figure 21.	Mean silhouette scores CDF	48
Figure 22.	Markov chain probabilities CDF	48
Figure 23.	Average silhouette scores per feature vector type	49

Figure 24.	Cluster 17 DNS activity	50
Figure 25.	Activity patterns of hosts in the wireless subnet.....	51
Figure 26.	Cluster 19 interevent analysis	52
Figure 27.	Interval distribution CDF	53
Figure 28.	Top country feature vector destinations.....	54
Figure 29.	Host A's cluster analysis.....	55
Figure 30.	Host A's volumetric activity patterns compared to the population	56
Figure 31.	Host A's per-hour activity compared to the population.....	56
Figure 32.	Host A's per-hour activity compared across days	57
Figure 33.	Host A port activity compared to the population.....	58
Figure 34.	Host A's daily activity compared to its subnet	59
Figure 35.	Host A's per-hour activity compared to its subnet	59
Figure 36.	Host A's port activity compared to the subnet.....	60
Figure 37.	Host B cluster analysis.....	61
Figure 38.	Host B top-5 interevent periods compared to the population	62
Figure 39.	Host B's per-hour activity compared to the population.....	62
Figure 40.	Host B's volumetric activity patterns compared to its subnet	63
Figure 41.	Host B's per-hour activity compared across days.....	64
Figure 42.	Host C cluster analysis.....	65
Figure 43.	Host C destination countries	66
Figure 44.	Host C bytes/packet ratio to Canada.....	66
Figure 45.	Host C byte/packet ratio to the United States	67
Figure 46.	Mean bytes sent per day.....	67
Figure 47.	Host C bytes sent to Canada compared to cluster 4 hosts.....	68

LIST OF TABLES

Table 1.	Cluster analysis based on discriminatory features	29
Table 2.	Data collection overview	39
Table 3.	Time-of-day cluster analysis.....	50
Table 4.	Time interval cluster analysis	52
Table 5.	Country cluster analysis.....	53
Table 6.	Destination bytes/packet ratios	66

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ADAMS	Anomaly Detection at Multiple Scales
AOL	America Online
BAIT	Behavioral Analysis of Insider Threat
BBAC	Behavior-Based Access Control
CITI	Clearswift Insider Threat Index
DARPA	Defense Advanced Research Projects Agency
DHCP	Dynamic Host Configuration Protocol
DOD	Department of Defense
DNS	domain name system
HPC	high performance computer
HSC	Hamming supercomputer
HTTP	hypertext transfer protocol
IP	internet protocol
IT	information technology
ITACS	Information Technology and Communications Service
NPS	Naval Postgraduate School
SMB	server message block
SME	subject matter expert
SVM	support vector machines
SVDD	support vector data description
TCP	transmission control protocol
UAB	user behavior analytics
UDP	user datagram protocol
USG	United States Government

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Insider threats are a costly and dangerous problem for government and non-government organizations alike. Modern organizations rely on oftentimes vast and spanning information networks to share mission critical information and conduct daily operations. Information networks are trusted to store sensitive, proprietary, and classified information, the unauthorized disclosure of which can lead to immensely expensive or even deadly consequences. With insider threats on the rise in the United States, information technology (IT) personnel find themselves at the forefront of this struggle.

Considering an insider's inherently privileged level of access on an information networks, the main principle of network defense—keep potential threats and outsiders out—does not apply to insider threats. Typical active defenses against insider threats include crude tripwires which rely on up-to-date insider threat indicators. While not without merit, these defenses can be easily avoided by a determined insider. Furthermore, recent and publicly available information on insider threats is scarce. Notwithstanding, insider threat activities are innumerable: they are as diverse as human behavior itself.

The objective of this thesis is to develop and test a system that extracts network host and organizational behavior from network traffic and detects behavioral anomalies using unsupervised machine learning techniques for the purpose of insider threat detection. The system is developed and tested on Naval Postgraduate School's (NPS) operational network with over 8,000 hosts over a contiguous 6-day period.

The system extracts host and organizational behavior from network traffic in the form of feature vectors using temporal measurements, volumetric measurements, and destination examination. We develop and engineer three different types of feature vectors; time-of-day vectors for describing human behavior, time interval vectors for describing device behavior, and country vectors that characterize human-to-device behavior. The system leverages the parallelism offered by the high-performance computer at NPS to process days of real-world network traffic to create per-host feature vectors of each type.

The system's anomaly detection module uses the K-Means unsupervised machine learning algorithm to cluster network hosts, and examine hosts' cluster membership across a given contiguous time period. The system specifically accounts for hosts with statically assigned IP addresses and wired hosts with DHCP-leased IP addresses. Wireless hosts on the network are not accounted for in our analysis because of the shorter lease times and transient nature of devices on and off the network. The anomaly detection module also uses silhouette scores and a Markov chain to examine host conformity to network trends as defined by the population of active network hosts.

Through a series of case studies, we found that the system is able to effectively detect host behavioral anomalies suitable for follow up by a human analyst. Our results show that our behavior extraction methodology is effective in characterizing host activity, and the resulting feature vector types are complementary in enabling the detection of behavioral anomalies. Future work will focus on the establishment of host-specific baselines over longer periods of observation.

ACKNOWLEDGMENTS

I would like to thank Dr. Vinnie Monaco and Dr. Gurminder Singh for their continuous guidance and encouragement throughout this research project. I would also like to thank Dr. Jeff Haferman and the NPS High Performance Computing team for their support. A big thank you goes to my sons, Teddy and Jack, for keeping me grounded. Finally, I would like to say thank you to my wife, Beth, for her continuous love and support.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Insider threats are a costly and dangerous problem for government and non-government organizations alike. There is a great deal of trust placed in modern information networks to store sensitive, proprietary, and classified information—the unauthorized disclosure of which can lead to expensive or even deadly consequences. Considering an insider’s inherently privileged level of access on a network, the main principle of network defense—keep potential threats and outsiders out—does not apply to insider threats. Current defenses are largely based on the detection of insider threat indicators and rely on up-to-date datasets. However, insider threat activities are innumerable: they are as diverse as human behavior itself. Examining and describing foundational host and organization behavior demonstrated on a network presents an opportunity to circumvent this problem. Leveraging machine learning to detect behavioral anomalies that indicate the presence of an insider threat would enable network administrators to quickly locate and mitigate such threats before they cause serious damage. We demonstrate this methodology by developing a system that extracts host and organization behavior from network traffic.

A. CONTEXT AND MOTIVATION

The CERT Division National Insider Threat Center of Carnegie Mellon University’s Software Engineer Institute defines an insider threat as someone who “has or had authorized access to an organization’s network, system, or data” and “has intentionally exceeded or intentionally used that access in a manner that negatively affected the confidentiality, integrity, availability, or physical well-being of the organization’s information, information systems, or workforce” [1]. The term insider threat has also been used to describe threats such as active shooters—employees who seek to inflict violence in their workplace on fellow coworkers. This thesis pertains to insider threats relating to computer network and information systems as defined by the CERT Division.

Businesses and government organizations rely on oftentimes vast and spanning computer networks to share mission critical information and conduct daily operations. Most organizations simply cannot function without this technology. This makes securing

and protecting networked infrastructure a paramount priority. While cyberspace attacks launched by an external organization, individual, or silent network intruder are most frequently associated with network threats, the Clearswift Insider Threat Index (CITI) annual report 2018, claims employees are responsible for 38% of information technology (IT) security incidents and 75% result from the actions of their extended enterprise (employees, customers, supplier) [2]. Hence insider threats are among the primary threats in cyberspace and must be dealt with in order to maintain resilient computer networks.

According to the “2017 U.S. State of Cybersecurity Survey” conducted by the CERT National Insider Threat Center, insider threat incidents primarily result in critical system disruption and the loss of confidential or proprietary information [3]. Furthermore, 29% of the U.S. businesses, law enforcement, and government agencies surveyed alleged cybercrimes were most costly when committed by insiders [3]. While it is not exclusively the responsibility of IT organizations and personnel to counter insider threats, most United States (U.S.) organizations view IT solutions as their primary means to defend against insider threats [3]. IT as a whole is uniquely positioned to counter insider threats in cyberspace and this survey suggests IT will be provided with the means to do so.

While other countries have seen a decrease in insider threat activity since 2017, insider threat activity has risen in the United States [2]. The U.S. Department of Defense (DOD), not immune to this trend, has placed special emphasis on defending against insider threats in the wake of high-profile insider threats involving Edward Snowden, Chelsea Manning, and Reality Winner [4]. The DOD Insider Threat Program establishes policies and standards for insider threat programs within the department that help prevent espionage, terrorism, unauthorized disclosure of national security information, and degradation of department resources and capabilities as a result of insider threats [5]. This program indicates the consequences of insider threats to the DOD are somewhat unique compared to elsewhere in the U.S. government (USG) or in the commercial sector. The potential for loss of life as a result of the unauthorized disclosure of sensitive or classified information makes detecting and preventing insider threats an even more urgent priority within the DOD.

B. PROBLEM STATEMENT

Due to their inherently privileged level of network access, threats from insiders are dangerous and costly to USG and commercial organizations. Based to the scarcity of up-to-date insider threat information and datasets, many detectors rely on crude and unrepresentative insider threat indicators. Such detectors can typically be easily evaded by a determined insider.

C. OBJECTIVES AND SCOPE

This thesis aims to determine how host and organizational behavior be extracted from network traffic for the purpose of detecting behavioral anomalies associated with insider threats. Additionally, this work will evaluate to what extent machine learning can be employed to detect behavioral anomalies for the purpose of insider threat detection.

The objective of this thesis is to create a system that:

- Extracts host and organizational behavior from network traffic
- Detects anomalous behavior demonstrated by a network host using machine learning techniques and metrics which evaluate host conformity with organizational norms
- Represents a framework for detecting insider threats using behavioral anomalies

This system does not seek to achieve the performance and timeliness needed from a real-world insider threat detection system. Instead, this system will seek to implement a methodology for insider threat detection that could be operationalized with proper consideration for operational requirements.

D. THESIS OUTLINE

Chapter II surveys state-of-the-art efforts to identify and understand how human and device behavior is demonstrated on information networks. It also discusses past and current efforts to detect insider threats using network traffic and/or machine learning

techniques. Chapter III discusses our methodology and initial system design for extracting host and organizational behavior from network traffic and using the extracted behavior for detecting behavioral anomalies. Chapter IV describes the system implementation, including how the system engineers feature vectors which describe host behavior and how unsupervised machine learning techniques are used to detect behavioral anomalies. Chapter V summarizes our results with an overview and a series of case studies. Chapter VI summarizes this work, states our conclusions, and discusses opportunities for future work.

II. BACKGROUND AND PREVIOUS WORK

This chapter discusses insider threat detection challenges, anomaly detection for the purpose of insider threat detection, research efforts to derive host and organizational behavior from network traffic activity, and automated state-of-the-art insider threat detection methodologies and technologies that use machine learning.

A. INSIDER THREATS DETECTION CHALLENGES

After several decades of research, insider threat detection is still an unsolved problem. Given that denying an adversary or wrongdoer access to a network is the primary means of defending one's network, how does one deny access to someone who already has access? Despite knowing more than ever before about insider threat behavior in the past, insider threats continue to evolve as network technology rapidly advances [6]. This phenomenon leaves researchers scrambling trying to find a new set of semi-conclusive indicators that could help detect future insider threats. To further complicate the problem, acquiring ground truth data on how insider threat behavior currently manifests on networks is exceedingly difficult because victimized organizations are often fearful of releasing such data for legal or privacy reasons, or fear of tarnishing their public reputation. This is a significant obstacle for researchers since many proposed detection techniques require ground truth data to measure detection accuracy [7].

Unlike network intruders whose attacks originate from outside the victim network, attacks launched by insiders originate from inside the victim network. This simple observation helps explain why detecting insider attacks is more difficult for cybersecurity analysts than detecting outsider attacks. As explained by Kelly in [8], "Insider attacks differ from outside in that outsiders are more easily detected as intruders when their means of access are found to be illegitimate." This difficulty is the root cause of why insider threat analysts often must unfortunately result to forensic analysis post-attack rather than detecting and preventing the attack [7].

Despite the inherent difficulties, the challenge of detecting insider threats and the emphasis levied by the government and commercial sector alike continues to attract

researchers and spur innovation. At its core, detecting insider threats remains an innately human problem; one that must consider the complex psychological, psychosocial, and organizational factors explaining why and how an insider becomes an insider threat [9]. While this behavioral analysis is a critical part of countering insider threats and must be considered alongside technical solutions, we focus on identifying clues that reliably indicate insider threat behavior and the technical methodologies that could be used to find these clues.

B. DETECTING ANOMALIES

Anomaly detection for the purposes of cybersecurity has been thoroughly studied for over 30 years dating back to Denning’s statistical model for detecting network intrusions in 1987 [10]. Anomaly detection systems used for the purposes of detecting malicious network activity rely on having an accurate view of normal network activity in order to recognize and flag abnormal network activity [11]. Every anomaly detection system assumes that malicious network activity has fundamentally abnormal characteristics not shared by normal or benign network activity. Moreover, anomaly detection systems assume these inherent (and oftentimes unknown) abnormal characteristics or attributes can never be fully hidden or obfuscated despite an adversary’s best attempts to do so.

Non-threatening insiders conform to an “organizational design” where they can be expected to behave in accordance with their organization’s rules, social norms, and patterns [8]. For this reason, experts agree that anomaly detection is an important factor in detecting insider threat activity [12]. Researchers have sought to tailor and operationalize anomaly detection systems to discover insider threat activity on computer networks. These detectors typically develop an understanding of normal host or organizational network activity and use this baseline to flag deviations from the baseline.

As pointed out by Sommers and Paxton in [11], anomaly detection systems typically generate a high number of false positives because they assume every deviation from normal is malicious and discount contextual factors that could describe these deviations as benign. This is the basis for one of Kelly’s main assertions in [8] that any

insider threat detection system must include a human insider threat analyst to “determine if anomalous inside network behavior, if not otherwise explained, is a threat.” This assertion explains why it is critical to identify conclusive attributes that describe insider threat activity which help draw a clear line between benign and malicious activity. In short, any detection system should seek to automate the process of finding insider threats to the furthest extent possible, thereby reducing the burden on human analysts, but not discount the need for them in the overall system. The technical solution proposed in this thesis does not attempt to remove humans from the system, instead it attempts to compliment other means of countering insider threats.

Perhaps one of the earliest insider threat detection systems that leveraged anomaly detection was in 2009 when Cuputo et al. introduced Elicit. Elicit generated user-specific datasets by collecting information from a user’s network traffic, digital object usage, and ancillary contextual information [13]. Their detector uses Bayesian networks to generate a threat score for a user’s daily activity based on the probability of the activity being malicious—or indicative of an insider threat. Elicit collected and processed network traffic to create per-user “information-use events” corresponding to activities in which a user read, wrote, or printed documents stored on an organization’s intranet [13]. Furthermore, Elicit gathered user events corresponding browsing and searching. Elicit used additional contextual information including users’ job titles, departments, and office locations to derive more meaning from information-user events, allowing Elicit to identify suspicious events when a user deleted or copied a document maintained by a different department, for example [13]. Caputo et al.’s Bayesian network “encodes the probabilities of occurrence for each activity for benign and malicious users” based on organizational norms, which were determined with the help of insider threat subject matter experts (SME) [13].

The Defense Advanced Research Projects Agency (DARPA) project Anomaly Detection at Multiple Scales (ADAMS) further leveraged anomalies in large datasets in order to detect insider threats in USG networks [14]. DARPA used machine learning techniques to identify user anomalous activity. One of ADAMS’s most significant contributions was identifying anomalies most typically generated by insider threats. ADAMS confirmed through empirical data that, as expected, malicious insiders fetch more

sensitive information than benign insiders, send more information outside of their organization, and are more active than benign insiders [14]. These insider threat activity observations paved the way for future generations of detectors by identifying and refining a set of confirmed insider threat clues.

Legg et al. [15] acknowledge the need for robust automation and diverse data sources to detect anomalies associated with insider threats. In addition to other records, their system harvests email, web, computer access, and physical building access logs for network users to construct “daily observation profiles.” Their system assesses the profiles to generate alerts based on “threshold-based anomalies” and “deviation-based anomalies.” Legg et al. envisioned their system as part of a bigger insider threat detection system where human analysts could “reject a detection result, which then refines the parameters within the system, to minimize the false positive rate for future observations.”

C. NETWORK TRAFFIC ACTIVITY

Using data generated from network traffic to spot anomalies is a common thread among many insider threat detection systems. Network users generate copious amounts of diverse metadata by executing simple network operations: checking email, web browsing, and using chat applications for example. Network traffic refers to the activity on a network, typically in terms of packets or flows. Network operators and researchers use network traffic measurements to understand network use patterns and trends for the purposes of network optimization in terms of performance, robustness, policy enhancements, and security. Network traffic analysis is typically conducted by examining packet or flow metrics. Depending on the desired insight, meaningful network traffic metrics include source, destination, length, and duration. The usefulness of these metrics is why network flows are sometimes used in network traffic analysis. Flows generalize and organize network traffic based on the source and destination of transmitted packets. Flows are typically aggregated by sampling packets in order to reduce overhead associated with collecting and examining every packet. The objective of one’s analysis dictates their use of individual packets or flows. Using flows is often desirable because of the reduced

overhead and because modern sampling techniques yield flows that accurately represent network activity [16].

Researchers and insider threat analysts recognize network traffic as a valuable source of information to describe a user’s daily activity and possibly identify insider threat activity [6]. When one uses network traffic as a source of data in detecting insider threats, there is an implicit assumption that the network traffic one generates reflects the threat’s behavior. Benign traffic is generated by insiders who conform to organizational norms, whereas malicious traffic is generated by insiders who demonstrate insider threat behavior. Given this assumption, we can expect insider threats to express behavioral anomalies via the network traffic they generate.

1. Human Behavior Revealed in Network Traffic

If we agree a population’s uncoerced behavior expressed on personal social network accounts accurately reflects true thoughts, motivations, and ideas, we can transitively extend this to assume a population’s digital footprint at the metadata-level also accurately expresses its behavior. Researchers hypothesize that web activity (revealed from raw network traffic) reveals organizational dynamics [17], [18]. Gonçalves and Ramasco conducted an experiment to determine if “laws govern the frequency with which a person visits a given website, or is each individual intrinsically unique” [17]. They analyzed the logs of an Emory University web server that covered a 292-day period. Their analysis showed that a population’s common behaviors can be inferred from individual server requests [17]. As intuition suggests, they found the circadian rhythm of a population can be inferred along with the impact of holidays on a population. Their results also showed that individual network users from the same population demonstrate similar request patterns to a server for the same website. Furthermore, this result indicates that the usage patterns of a work-oriented website are similar. This observation is useful in our pursuit of insider threats, because it likely indicates that misuse is distinct (anomalous) from the typical usage demonstrated by the population.

In [18], Radicchi was also able to infer a notion of normal human behavior on the web from which we could detect anomalies. Radicchi used a diverse dataset of America

Online (AOL) search queries, eBay feedback messages, and Wikipedia logging actions. This dataset captures user interevent times between the user actions on each of the three websites. Radicchi attempts to find universal rules that govern the spontaneous actions of users on the web by examining interevent times. He concludes that “the shape of the single user’s interevent time distributions is strongly influenced by the total numbers of operations performed by the users” [18]. However, he also finds a universal rule for human activity on the web by applying relative weights to interevent times. In summary, Radicchi was able to infer patterns that define user activity and global activity. Whereas in [17] the authors are able to draw conclusions about humans who have a common membership in an organization (revealing organizational dynamics and patterns), Radicchi’s dataset applies to a larger subset of the global population, allowing him to make global inferences about human activity on the web. For the purposes of insider threat detection, Radicchi’s work suggests humans can be profiled based on web activity interevent time distributions and behavioral anomalies detected thereafter. Furthermore, behavioral anomalies could be recognized on a global scale—an invaluable characteristic for making an insider threat detector portable and relevant to a diverse set of networks.

2. Detecting Behavior Anomalies in Network Traffic

Network traffic is a natural source of data for recognizing the human behavior on the web as mentioned in [17] and [18]. Carefully considered measurements taken from appropriate network vantage points offer a plethora of diverse information about a network’s operations and users [19]. If interpreted correctly, network traffic packet captures allow us to track most of an insider’s activities, perhaps most notably their client-server connections. However, this diversity also offers unique challenges to anomaly detection systems. As explained by Sommers and Paxton in [11], “Even within a single network, the network’s most basic characteristics ... can exhibit immense variability, rendering [anomaly detection systems] unpredictable over short time intervals” [11]. Anomaly detection systems have difficulty classifying bursts of activity which happen regularly in typical networks [11]. Due to variability of network traffic and the resulting

high likelihood of reported false positives, it is critical that we lower the cost of false positives and carefully evaluate what anomalies the system detects and why [11].

In 2000, Mahoney and Chan introduced a network traffic anomaly detector that used packet header values and ranges to detect abnormalities [20]. Their work proved that some network attacks could be detected at higher accuracy than previously seen without using the application payload, destination address, or port numbers (which were previously believed to be necessary for a successful detector). They also found their best results were obtained when their model examined “each field and each packet separately” [20]. While the computational cost of not using network traffic aggregation techniques is potentially high, it allows the model to successfully function at a low level without knowledge of the protocols being used on the network. This feature allows such a detector to operate successfully across different networks with different uses. Considering the wide range of network use in the DOD, the portability of such a system could be invaluable for detecting insider threat on various DOD networks.

Unlike Mahoney and Chan, Lakhina et al. in 2005 argued using network flow captures allows an anomaly detector to search for a larger and more diverse set of anomalies across a network [21]. While they acknowledge the challenge of handling these big data sources, they argue using network flows allow “general network anomaly diagnosis” instead of diagnosing localized anomalies in subsections of the network. The generality they aspired to achieve in their anomaly detection system is an important characteristic in today’s world where old network attacks are constantly evolving, and new network attacks are constantly arising. They aspired to create a general anomaly detection system that “was not restricted to any predefined set of anomalies” [21]. In the spirit of achieving generality in their system, Lakhina et al. classify anomalies simply as events that disturb network flow feature distributions, and use entropy to identify anomalous changes in the distributions [21]. While they implement their systems differently, Lakhina et al. and Mahoney and Chan furthered the idea that network traffic anomaly detectors can be effective and highly versatile.

A recurring theme in most of the proposed insider threat detection systems or anomaly detection systems, is that any such system should be a system of systems with

layers of diverse security measures. There is not a single tool or mechanism that will detect insider threats or network traffic anomalies with near-perfect accuracy by itself. This is an accepted idea by the research community and insider threat experts alike [1], [11], [20]. With this layered approach in mind, most modern insider threat detection systems leverage multiple sources of data and include network traffic to develop a more complete picture of user activity and behavior. Commonly used data sources are shown in Figure 1. Splunk User Behavior Analytics (UAB) and IBM QRadar UAB are examples of such modern insider threat detection systems [22], [23]. As advocated in [21], IBM advertises that its QRadar UAB system uses network flows and logs to develop per user activity baselines from which it can identify user anomalous activity and generate user risk scores [23].

By developing what they call Beneficial Intelligent Software (Ben-ware), McGough et al. provide insight into how Splunk and IBM identify anomalous human behavior via network traffic [24]. Ben-ware is a low-weight, distributed hierarchical system designed for typical organizations with heterogeneous networks. Ben-ware uses low-level host-based probes that collect “events” that define user behavior on the network. The events include system log-ins, user interaction with external storage devices, file use, and HTTP requests. The HTTP request information includes number of HTTP requests, out of hours HTTP requests, and HTTP requests on black-listed sites. Perhaps the most unique aspect of Ben-ware is its use of “human factors and analysis to influence and develop AI techniques,” allowing Ben-ware to limit the number of false positives that plague many insider threat detectors [24]. This unique feature of Ben-ware identifies users who are more likely to be a “criminal threat” rather than those who exhibit negligent but non-malicious intent [24]. Collaboration with criminologists and behavior analysts enabled McGough et al. to integrate this additional feature set into Ben-ware, and they claim that the information they used is common and already possessed by many organizations through routine human resource processes. McGough et al.’s implementation of Ben-ware to detect insider threats confirms that network traffic is a key data source in detecting behavioral anomalies. However, by integrating other human factors into their system, they acknowledge that network traffic is an insufficient data source by itself.

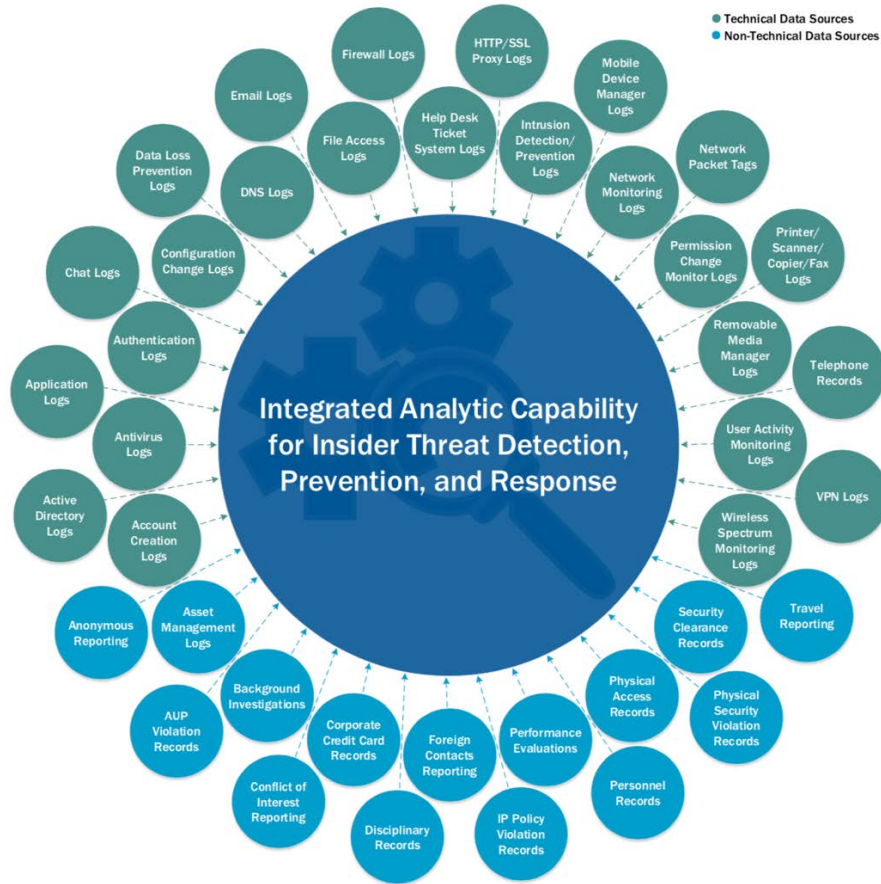


Figure 1. Data sources for insider threat detection. Source: [1].

D. MACHINE LEARNING

Many insider threat detection systems apply statistical analysis and machine learning algorithms to big data in order to identify anomalous behavior exhibited by insider threats. Given there are not unique signatures or absolute rules associated with insider threat activity, detectors using machine learning are an attractive solution. Instead of attempting to explicitly program a detector to recognize predetermined insider threat indicators, a detector using machine learning seeks to recognize insider threat and benign activity by learning from examples. In essence, such a detector generates its own rules and indicators from the presented data. This automation offered by machine learning enables security personnel to glean valuable insight from big unprocessed data that a human would otherwise be unable to interpret. If we envision an insider threat detection system as a

collection of subsystems, machine learning allows security personnel to maintain a supervisory role over an organization's insider threat program; keeping them out of the minutia of any single subsystem and enabling them to make top-level inferences from intelligence (instead of raw data or information) derived from multiple subsystems. While not without its own set of challenges, machine learning is a valuable facet of insider threat detection and its implementation in existing detectors and prototypes proves this.

1. Machine Learning Challenges

Machine learning algorithms are commonly classified as supervised, unsupervised, or semi-supervised. Supervised learning, relying on ground truth or labeled data to train a model, is oftentimes infeasible for insider threat detection for several reasons. First, there are few publicly available datasets labeling insider threat activity [6], [11]. Supervised machine learning algorithms seeking to classify an object or activity—particularly artificial neural networks—thrive and produce their best results when they observe and learn from numerous examples of the object or activity they are trying to classify. Victim organizations of insider threat in the commercial sector and government are reluctant to release insider threat data because of its sensitive nature, frequently containing personal information or private details of an organizations activity [11], [25]. Second, insider threat data—for example, insider threat network traffic—is minuscule when compared to benign network traffic. As a result, the rare datasets that do contain insider threat activity are highly imbalanced, which are not conducive for supervised machine learning algorithms such as decision trees [26], [27]. Third, it is difficult for supervised machine learning models to accurately classify insider threat activity due to its diversity. The most common insider threat activities (excluding espionage or unintentional damage) are IT sabotage, theft of intellectual property, and fraud [1]. Although they are all examples of insider threat activity, these activities are very diverse, and the set of all possible actions taken by an insider threat to execute these activities is also large and diverse. This diversity and uncertainty constitute a significant problem for supervised machine learning models, especially for those that rely on commonalities between cases of the same class to classify new cases.

Due to the difficulties in using supervised machine learning to implement insider threat detectors, researchers and innovators often resort to semi-supervised or unsupervised machine learning. Unsupervised machine learning algorithms are typically used in an exploratory fashion, grouping similar objects together thereby isolating anomalous outliers. While it might seem intuitive that unsupervised learning can be applied to insider threat detection, which tries to discover behavioral anomalies that are by definition dissimilar from normal behavior, unsupervised learning in practice is difficult to operationalize [11]. Although Sommers and Paxton explain why, “despite extensive academic research one finds a striking gap in terms of actual deployments of such systems [anomaly detectors]” in terms of intrusion detection systems, we believe the same explanation applies to insider threat detectors leveraging anomaly detection.

2. Machine Learning Algorithms

In this section, we examine (and in some cases, reexamine) several insider threat detection systems for the purpose of evaluating the specific machine learning algorithms they used. We find that many times the machine learning algorithms attempted are predicated by the data on which the authors chose to train and/or evaluate their machine learning model. This is an unfortunate reality due to (as already discussed) the lack of real-world insider threat datasets. For this reason, a short discussion on the common types of insider threat datasets and their merits and inferiorities is warranted.

Insider threat datasets used to train and/or evaluate detectors using machine learning can broadly be classified as synthetic or real. CERT proposed a popular publicly available synthetic insider threat dataset that was generated to mimic a realistic organization with insider threat instances [15], [25], [26]. Others such as McGough et al. choose to generate their own synthetic data based upon insider threat scenarios on which they test and train their detectors [24]. Being able to recognize and correctly classify benign data is important for reducing the number of false positives a detector produces. For this reason, insider threat detectors are also oftentimes exposed to real, anonymized data from operational networks sourced from a university, for example, that is supportive of academic research in the area of insider threats [28]. Despite the popularity of synthetic data,

researchers widely agree there is no substitute for real data collected from an operational network when training and/or evaluating detectors because of the inherent difficulties in adequately representing the diversity of human behavior in synthetic data [11], [25]. Simulating insider threat data, while still not as good a real-world data, signifies a worthy step in the right direction. Harilal et al. simulated insider threat data by designing a “gamified competition” in which participants organized into teams and competed against each other in a corporate scenario [29]. Participants in the game were incentivized through scenarios in the game to act as masqueraders and traitors, thereby creating realistic insider threat activity traces [29].

a. *Semi-supervised Machine Learning Approach*

While more difficult to implement in insider threat detection systems due to the dataset limitations we have already discussed, researchers have experimented with using semi-supervised machine learning to detect insider threats when labels are available for at least a small subset of their data. Within their ADAMS project, DARPA simulated insider threat activity through an experiment they called the Behavioral Analysis of Insider Threat (BAIT) framework. Participants in the experiment were permitted to fetch, edit, write, print, and/or send documents or data for the purpose of exfiltrating the data without being detected. These participant actions were used by the research team as dataset features for their machine learning model to detect insider threat activity. The experiment resulted in a highly unbalanced and mostly unlabeled dataset [14], [30]. DARPA demonstrated the use of bootstrapped semi-supervised machine learning algorithms based on traditional learning algorithms including Support Vector Machines (SVM) and multinomial Naïve Bayes to differentiate the behavior of malicious users and benign users [14], [30].

Mayhew et al., who developed Behavior-Based Access Control (BBAC) seeking to defend DOD networks from insider threats, also use a combination of supervised and unsupervised machine learning algorithms to detect insider threat behavior [31]. BBAC ingests network traffic to include network flow information, data generated from higher-level transport protocols, and audit records from DNS servers [31]. BBAC also ingests and processes application-level content such as email and chat messages. From the network

traffic and application-level content it ingests, BBAC extracts features derived from user TCP connections, HTTP requests and responses, email and chat messages, and collaborative web page edits (specifically MediaWiki). BBAC uses the K-means clustering, SVM, and decision tree algorithms in a unique way to “improve scalability and reduce false positives” [31]. BBAC uses the K-Means algorithm to cluster network users and network devices based on their observed behavior in order to establish a baseline of normal activity for each category. The authors claim that “in addition to increasing accuracy through grouping entities with similar behavior into clusters,” this methodology also improves system performance in terms of speed by allowing them to logically partition data and distribute it to separate cluster nodes for training and classification [31]. BBAC uses SVM classifiers to apply supervised machine to each cluster in order predict whether specific network users and network devices are behaving anonymously. To support integration with human analysts, BBAC also uses decision trees. Mayhew et al. derive a set of rules that define each K-means cluster and use these rules to build a decision tree for each cluster of network users and network devices. Being significantly more intuitive and easier to understand than other classifiers, the decision trees allow human analysts to understand why a user or device was assigned to a specific cluster over another. The decision trees also allow human analysts to assign users or devices with little or no collected data to a cluster by the K-means algorithm if needed [31].

b. Unsupervised Machine Learning Approach

Caputo et al. used real network data collected from an operational corporate network to test Elicit [13]. Initially they employed anomaly detection to identify “suspicious behaviors that violate organizational norms” including network volumetric anomalies [13]. Despite their data not being labeled, they were able to use hypotheses about insider threat behavior formed by consulting with insider threat SMEs to obtain probabilities and dependencies of anomalies being associated with insider threat activity. These probabilities and dependencies were used to create Bayesian network which evaluated an overall threat score for each user [9]. Although, Bayesian methods are

typically associated with supervised learning, Elicit initially relies on unsupervised learning techniques to identify anomalies.

For Ben-ware, McGough et al. used Support Vector Data Description (SVDD) and clustering techniques to identify outliers as insider threats [24]. SVDD, a classifier developed from the popular Support Vector Classifier, was used to calculate “minimum bounding hyper-spheres” around clusters of network users from which they could identify outliers [32]. One of their key observations was the increase in detection performance when they used a clustering technique such as K-means clustering and then applied SVDD to the resulting cluster. McGough et al. hypothesize that this method allows them to capture user sub-profiles that are a result of a user changing roles within their organizations or performing multiple roles [24].

3. Online and Offline Machine Learning

Online machine learning learns from data streams whereas offline machine learning learns from batches of data. Training on continuous data streams allows a model to be incrementally updated as it ingests new data. In this way, the model adjusts its parameters in accordance with new network usage patterns, thereby enabling it to make more informed predictions as well as increasing its overall performance. Aside from the obvious performance advantages, online machine learning techniques offer several other practical advantages over batch machine learning techniques for anomaly detectors. As a result of not having to train on an isolated batch of data before being operationalized, anomaly detectors leveraging online learning techniques can be implemented immediately. Such a model establishes a baseline for normal activity over time, gradually increasing its ability to provide valuable predictions. Lastly, due to the real-time nature of online learning, streaming data is not cached for analysis at a later time. This allows an insider threat detector leveraging online learning to make real-time detections, allowing human analysts to take action immediately against suspected insider threat activity, if warranted [33].

Although the seamlessness of online machine learning is difficult in practice to implement on an operational network, there have been many research pursuits we can learn

from in this regard [11]. Tuor et al. use an online unsupervised machine learning approach leveraging a deep and recurrent neural network model to detect insider threats [33]. One of the novelties they achieve in their system is presenting an individual user’s activity to the model in a single and continuous stream of data. Tuor et al. extract user features from system logs. Specifically, they extract user categorical data (role, project, department, etc.) and user “counts” that include, for example, “the number of uncommon non-decoy file copies from removable media between the hours of 12:00 p.m. and 6:00 p.m.” [33]. While their use of deep and recurrent neural networks increases the complexity of implementing online learning, they are able to overcome this by only allowing the neural networks to observe a single sample once and training one recurrent neural network per user [33]. In order to allow their system to adapt to new user activity and readjust baselines, they continuously update a finite user-specific data structure [33].

Similar to Tuor et al., Böse et al. translate streaming data into user-specific streams to implement online anomaly detection for the purpose of detecting insider threats [34]. These user-specific streams become representative feature vectors that are used by their model to detect anomalous behavior. Unlike in [33], each feature vector encompasses a user’s current session and a different model is trained for each feature vector. Their feature vectors are generated from HTTP logs, logon events, email messages, and removable media usage. Each user session feature vector has five attributes: start time, session duration, number of emails sent, the presence of removable media (Boolean) and number of HTTP requests. Subsequently, they use the K-nearest neighbor algorithm to cluster user sessions and produce an anomaly score; if the point meets a specified threshold, it is classified as an anomaly [34].

E. SUMMARY

This chapter introduced current research efforts and the state-of-the-art technologies and methodologies used to detect insider threats. We discussed the use of anomaly detection to detect behavioral anomalies expressed by insider threats not conforming to organizational conventions. Specifically, this chapter discussed how human and organizational behavior is expressed via network traffic, and how the proper

application of network traffic analysis can characterize human behavior. We also discussed the use of machine learning as part of a greater insider threat detection system to automate the fine-grained analysis of big data. The following chapter describes the integration of network traffic analysis and machine learning to detect behavioral anomalies indicative of insider threats.

III. METHODOLOGY AND SYSTEM DESIGN

This chapter introduces and discusses our methodology and system design for extracting human and device behavior from network traffic and identifying behavioral anomalies for the purpose of detecting insider threats caused by both users and misconfigured devices.

A. INTRODUCTION

As discussed in Chapter II, network traffic is a popular source of information commonly leveraged by insider threat detection systems to track and analyze hosts' network activities [22], [23], [31]. In addition to explicitly revealing one's network activity, network traffic can also reveal human and organizational behavior to include usage patterns and preferred network services [17], [18]. However, extracting behavior from network traffic is an inexact and difficult task. Those who endeavor to do so must contend with the high volume and diversity characteristic of network traffic [11]. Furthermore, acquiring suitable network traces is oftentimes challenging in itself for several reasons. First, for reasons already discussed in Chapter II, organizations are oftentimes reluctant to share network traffic [11]. Second, when collecting network traces, one must employ strategies addressing vantage point, accuracy, and calibration (to name a few) which strive for soundness. Paxton describes pursuing soundness in Internet measurements as, "developing confidence that the results we derive from our measurements are indeed well-justified claims" [19].

In order to develop an understanding of the human and device behavior embedded in network traces, packets must be synthesized in a manner which suits our task. For insider threat detection, such a synthesis must help us characterize a network host's activity. This activity allows us to characterize and describe a network host's normal network activity and compare it to the activity of all the hosts on the network. Deviations from the normal activity demonstrated by the population of hosts on the network are considered anomalous. While this is useful for establishing a baseline for the purposes of detecting behavioral anomalies, we also consider more direct insider threat indicators and integrate these into

our synthesis. Furthermore, such a synthesis must be compatible with the methodology we intend to implement to identify anomalous insider threat behavior. As a result, our system will seek to extract behavior in the form of feature vectors, upon which we can apply machine learning techniques and statistical analysis, as desired.

B. METHODOLOGY

The following paragraphs describe our methodology for implementing an insider threat detection system which focuses on detecting behavioral anomalies instead of relying solely on potentially unreliable and unknown insider threats signatures. Simply stated, we intend to treat insider threat detection as an anomaly detection problem, foremost striving to characterize undisguisable human and device behavior demonstrated on a network. Our system design broadly consists of two parts, one which extracts behavior from network traffic and another that detects behavioral anomalies which are potentially indicative of an insider threat.

1. Extracting Behavior from Network Traffic

We consider the way in which we extract behavior from network traffic to be the most important part of our system. If we are able to accurately and succinctly describe the behavior of network hosts (within the context of insider threat detection) from sound network traces, then we assess a tailored machine learning pipeline will be able to identify behavioral anomalies, indicating a possible insider threat. Partially based on the findings in [17] and [18], we use temporal measurements, volumetric measurements, and destination examination to describe human and device behavior and engineer corresponding features.

Temporal metrics allow us to observe when a host is active. Volume metrics allow us to observe the activity level of a host. For destination examination, inspecting the destination ports of only TCP and UDP packets sent by a human or device allows us to see preferred ports, and derive the network service a host is using [35]. Understanding the network service allows us to reasonably characterize activity type. Destination examination is foremost used as a way to engineer features that enable us to see underlying patterns and

preferences. However, destination examination also enables us to uncover potentially more direct indications of insider threats. Anomalous TCP and/or UDP connections to questionable destination countries for the purposes of data exfiltration is an infamous feature of past insider threats and is of particular concern to the USG [1]-[4].

For the purposes of our system, we consider human behavior to be the result of network activity initiated at a device which was directly influenced by a human user. That is, human behavior is the result of activity demonstrated by a device that would not have been initiated without direct human intervention. For example, human activity includes HTTPS requests and subsequent DNS queries generated as a result of web browsing. Device behavior is demonstrated by device activity that was initiated without, or with indirect, human intervention. Examples of device activity include automatic checks for software updates and security patches. It is important to note that human and device behavior can be—and are oftentimes—exhibited on a single machine. In order to precisely capture different behavioral nuances for the purpose of anomaly detection, we extract human and device behavior differently. However, we conduct destination examination in the same manner for humans and devices.

The following paragraphs describe our design for different feature vectors that characterize human, device, and human-to-device behavior separately. However, these descriptions are not intended to be mutually exclusive—we expect human activity to affect the features primarily used to describe device behavior and vice versa. But, we do expect each feature vector type to describe its given subject better than the other subjects.

a. Characterizing Human Behavior

In regard to recognizing human activity patterns, we consider activity over the duration of a single day. In order to capture patterns at a finer level, we also consider activity within single hours. These time ranges allow us to characterize when a user is active holistically over the course of a day; from this information we can infer typical and atypical patterns given sufficient time to observe a user. More specifically, we can reasonably infer a user’s typical circadian rhythm, when they log onto the network to start working, and when they take a break for lunch, to list a few examples. Figure 2 shows our

feature vectors before accounting for volume and destination examination. Booleans indicate user activity did or did not occur during a given hour.

Host	Hour 0	Hour 1	...	Hour 23
Host A	False	True	...	True
Host B	True	False	...	False

Figure 2. Temporal measurement for human behavior

To address volume, we count the number of packets each user sends in each hour of a given day. Layering the volume measurement onto our existing feature vectors results in feature vectors with numeric values instead of Booleans, as shown in Figure 3.

Host	Hour 0	Hour 1	...	Hour 23
Host A	0	4242	...	7766
Host B	9003	0	...	0

Figure 3. Temporal and volume measurements for human behavior

Finally, in order to characterize a user’s activity type, we inspect the destination port of each packet sent by the user. We count packets sent to the same destination port during the same hour. Applying this measurement results in volumetric information broken down by destination port, as seen in Figure 4. Instead of accounting for every possible destination port, we select and enumerate the most commonly used destination ports across the entire network. Hereafter, we will refer to these feature vectors as “time-of-day” vectors.

	Port 443				...	Port 22			
Host	Hour 0	Hour 1	...	Hour 23	...	Hour 0	Hour 1	...	Hour 23
Host A	34	0	...	8088	...	0	0	...	997
Host B	0	4242	...	0	...	0	0	...	1212

Figure 4. Time-of-day feature vectors describing human behavior

b. Characterizing Device Behavior

We characterize device behavior in a similar way to how we characterize human behavior. As previously mentioned, in order to differentiate between human device activity, we measure time and volume in a slightly different manner in order to account for the high number of requests per time unit characteristic of devices, and the bursty network usage associated with humans. This difference also allows us to more accurately profile their respective behaviors [17]. The network services used as result of device activity will oftentimes in itself allow us to also differentiate it from human activity. For example, connections to destination port 8014 for Symantec security updates is likely the result of automated device activity. Like humans, devices have preferred destination ports as a result of how they were configured.

Instead of measuring volume over days and hours, as we did for humans in Figure 4, we measure volume within discrete time ranges for each considered port. That is, we count the number of packets transmitted by a device within a given time range, measuring the time from when the previous packet was sent to when the next packet is sent. For example, if device A exhibits the activity demonstrated in Figure 5, it would result in the binned feature vector also portrayed in Figure 5. This histogram of packet interarrival times characterizes the rates at which upstream traffic is generated by a particular host.

Device A	Packet number	1	2	3	4	5
	Time sent (μ s)	1	3	4	7	8
	Interevent time (μ s)		2	1	3	1

Device	(0-1] μ s	(1-3] μ s	(3-5] μ s
Device A	2	1	1

Figure 5. Temporal and volume measurements for device behavior

In order to account for how devices typically transmit packets in a regular and high frequency fashion, the time range bins have to be small—we assess on the order of microseconds. Adding the measurements that were a result of our destination examination,

our device feature vectors (hereafter referred to as “time interval” feature vectors) appear in their final form, as shown in Figure 6.

	Port 8014				...	Port 22			
Host	(0-1] μ s	(1-3] μ s	...	(45-60) mins	...	(0-1] μ s	(1-3] μ s	...	(45-60) mins
Host A	1001	5005	...	0	...	19	98	...	0
Host B	58	607	...	0	...	6	4500	...	0

Figure 6. Time interval feature vectors describing device behavior

c. *Characterizing Human-to-Device Behavior*

For human-to-device behavior we broaden our focus and examine how humans interact with distant devices across a network in a single day. While the time-of-day feature vectors allow us to specifically target human behavior and purposely disregard external factors, our third set of feature vectors should allow us to see how humans interact with destination devices. We shift our attention from the temporal metrics that allowed us to precisely characterize human and device behavior and further examine destinations and volumetric attributes. Instead of inspecting the destination port as we did to characterize human and device behavior, we instead examine the destination IP address and lookup its location by country. To measure volume, we count the packets sent by a device on the network to all destination IP addresses in a single country. Furthermore, we add the lengths of the packet payloads sent to a given country to get the total amount of data sent to a destination country.

We assess this balance between utilizing destination and volume metrics to describe human-to-device behavior allows us to see the volumetric patterns of how a network host interacts with a spatial grouping of destination devices. While it would be interesting and beneficial from an insider threat detection perspective to analyze the patterns by which a network host interacts with specific destination IP addresses throughout the course of a given day, this would be unbeneficial for the machine learning techniques we have chosen to employ for anomaly detection and has some other considerable drawbacks as well. For example, as we will discuss later, such a methodology would require us to examine how the union of network hosts interacts with the union of destination IP addresses. This would

require our feature vectors to have thousands of features (columns), and the computational costs would not be beneficial for providing an operational solution for insider threat detection. Querying destination IP addresses by country also allows us to potentially detect indications of data exfiltration by examining the ratio of the number of packets versus the amount of data sent to a country. Figure 7 provides an example of how we intend to characterize human-to-device behavior with what we call “country” feature vectors.

Host	Packet Count				Byte Count			
	U.S.	UK	...	Russia	U.S.	UK	...	Russia
Host A	1001	503	...	156	560004	25004	...	4589
Host B	58	607	...	0	670	36005	...	0

Figure 7. Country feature vectors describing human-to-device behavior

While not shown in Figures 4, 6, and 7, we normalize each feature vector separately in order to represent each value as a probability. This benefits the machine learning techniques we use to detect behavioral anomalies. Of note for the country feature vectors, we normalize the union of packet count columns and the union of byte count columns (Figure 7) separately due to how these columns represent separate but related activity.

2. Detecting Behavior Anomalies

The second part of our system design is focused on detecting behavioral anomalies as they occur over time to identify potential insider threat activity. The use of statistical methods and measurements has proved effective in insider threat detection [31], [36]. However, in many cases, statistics are applied to broad insider threat actions and indicators observed in the past, such as file transfers and system calls. While these measurements are useful for detecting network use anomalies, they do not offer the granular insight we wish to pursue for detecting behavioral anomalies. Furthermore, an insider threat detection system which relies on statistical measurements is guaranteed to only detect anomalies in the data from which it has chosen to collect, process, and analyze. Stated differently, the system’s effectiveness is solely reliant on the data from which the models were trained, and if the wrong data is used (e.g., the data is biased in some way), the system may fail. This makes the designers of such a system reliant on current and relevant insider threat

datasets and indicators, which do not exist. Even if the insider threat research community had perfect knowledge of past insider threats and their indicators, the manner in which one could seek to overcome or subvert network defense mechanisms to conduct an insider attack is too vast and diverse for a system to reasonably enumerate and analyze using statistical measurements and methodologies. Insider threat activity is as diverse and unbounded as human behavior. We seek to overcome this problem by disregarding suspected insider threat indicators and attempting to detect human behavioral anomalies using machine learning techniques.

a. Clustering Network Hosts

As discussed in Chapter II, machine learning has been widely utilized in insider threat detection systems. Given the lack of ground truth regarding malicious activity in our data, the lack of known insider threat indicators, and the subtleties and diversity of human behavior, unsupervised machine learning will be our primary means of detecting abnormal behavior. Unsupervised clustering algorithms allow us to group network hosts according to behavioral similarities, and these clusters may form the basis in characterizing how host behavior changes over time. In addition, clustering offers valuable insights on a macro-level regarding external factors and phenomena that affect how a network is being utilized holistically. Corroborating the macro and micro-level trends revealed by clustering with other information about the network (i.e., static versus Dynamic Host Configuration Protocol (DHCP) leased IP addresses, and working hours versus non-working hours), offers a sense of ground truth that could help explain the behavioral inconsistencies discovered through unsupervised models. However, a significant drawback to using an unsupervised model is the lack of feedback regarding how and why a sample (in our case, a network host) was clustered or grouped in the manner it was. To address this problem, we attempt to use feature selection and supervised learning techniques to describe clusters and isolate the most descriptive attributes of a cluster's occupants.

Despite issues with accuracy and stability, the K-Means clustering technique is an attractive solution due to its simplicity and convergence speed (particularly with large datasets with many dimensions) [37], [38]. K-Means utilizes randomly selected centroids

to group samples based on variance and iteratively recomputing the centroids as the algorithm clusters additional samples and converges. K-Means implementations such as Scikit-learn offer the ability to observe each sample's distance from specified centroids, providing a measurement of how similar or dissimilar a sample is with other samples in the same cluster [39]. Harking back to our lack of ground truth data, we must determine the ideal number of specified clusters to use with K-Means. The well-known silhouette score allows us to measure how well our clusters are defined using distances between samples sharing the same cluster membership, and distances between samples in the next closest cluster [40].

b. Cluster Analysis

Despite their usefulness in other pursuits, K-Means cluster distances and silhouette scores do not offer the required context needed for insider threat detection and are alone insufficient for determining how network hosts are grouped. While these metrics are valuable in determining the number of clusters we should use, they do not reveal why hosts with certain behavioral characteristics are grouped. For example, ideally, we want to be able to describe the clusters shown in Table 1 according to their most discriminatory features and possibly make several tentative assessments.

Table 1. Cluster analysis based on discriminatory features

Cluster ID	Number of Hosts	Discriminatory Features	Tentative Conclusion(s)
Cluster A	364	(1) low activity level in the morning (2) high port 443 use in the afternoon (3) high port 22 use in the afternoon (4) sends packets at irregular internals	(1) Workstations used by humans
Cluster B	2001	(1) packets sent at regular intervals (2) packets sent at high frequency	(1) Demonstrate device behavior (2) Printers

This table provides an example of how we would like to understand and analyze clusters. For the purpose of this example, the "Discriminatory Features" column lists information we would likely have to ascertain from several features in the dataset.

In order to analyze each cluster, after first clustering hosts with K-Means we use dimensionality reduction and feature selection techniques to discover each cluster's most descriptive features. Typically used to reduce the time it takes to train a classification model, dimensionality reduction via feature selection is the process of removing the least descriptive features from the dataset for a given cluster or label while attempting to preserve classification accuracy and recall. Each feature in the dataset is assigned a score, and all but the top n scoring features are removed. In our case, we use feature selection to isolate and identify each cluster's prominent features. To simplify this process, we leverage a one-versus-all strategy where we judge a given cluster against the union of all other cluster samples. Stated otherwise, we cast this as a binomial classification problem specifically for the purpose of cluster analysis.

Through feature selection we are able to learn any given cluster's most discriminatory features, which allows us to draw limited conclusions about the membership of the cluster (based on which of the three datasets we are analyzing). However, while these features might be the most discriminatory, they may not be effective in discriminating one cluster from another. To determine the degree to which these features help differentiate one cluster from another, we train and test a decision tree classifier with only the top n features outputted by the feature selection process and inspect the classifier's accuracy, precision, and recall. Through this process of feature selection and bootstrapping a supervised classifier, we are able to quantify the effectiveness of the clustering.

c. Identifying Anomalies

While analyzing how the clusters themselves are formed describes organizational behavioral trends (to include anomalies), we are primarily interested in individual host behavior relative to individual baselines and macro-level network utilization trends for the purpose of insider threat detection. Here, we must take a closer look at cluster membership and how members are clustered differently over time. Of note, without a labeled dataset, declaring that a network host is demonstrating anomalous behavior is ultimately a subjective judgment call. After network hosts are clustered across the observation period, we must quantitatively describe why a host is behaving normally or abnormally. This leads

us to the creation of a cluster analysis dataset which will describe how each network user is clustered over a given time period. This dataset will be the primary tool we use to detect behavioral anomalies.

Foremost, we need to note the cluster in which a host appeared on each day over a given timeframe. This measurement also allows us to easily count the number of days a host was active on the network. If a host did not maintain enough presence on the network over the chosen timeframe, the sample size may be too small to declare the host's behavior anomalous or normal. Figure 8 shows how we will track the cluster in which a host appeared on a given day by determining which cluster each host belongs to on each day and on how many days each host is active.

Host	8Feb Cluster	9Feb Cluster	10Feb Cluster	11Feb Cluster	Days Active
Host A	0	0	0	0	4
Host B	1	2	NA	4	3

Figure 8. Preliminary analysis showing host cluster membership

The next two metrics we add to our cluster analysis help describe the level of consistent or inconsistent behavior a host is demonstrating despite how they may be hopping between clusters (in the case of Host B in Figure 8). For example, while Host C in Figure 9 appears to be hopping between clusters, one might conclude they are behaving consistently due to how they are hopping between the same two clusters. Whereas in the case of Host D in Figure 9, one might conclude they are behaving abnormally due to their membership in several different clusters. Denoting the number of hops and number of unique clusters, as shown in Figure 9, describes these phenomena.

Host	8Feb Cluster	9Feb Cluster	10Feb Cluster	11Feb Cluster	Days Active	Hops	Unique
Host C	0	0	1	0	4	2	2
Host D	1	2	NA	4	3	2	3

Figure 9. Cluster analysis example

Again, using Figures 8 and 9 as the basis for our example, the next logical question may be how similar or dissimilar is Host C's behavior to the other members of cluster 0 on 8 Feb? Is Host C considered an outlier in cluster zero relative to the other hosts in cluster

0? Is Host C close to the cluster 0 centroid relative to the other members of cluster 0? To help answer these questions, we use the sample silhouette score as a metric to determine whether the host’s behavior on a given day was appropriately clustered. In Figure 10 for example, Host C’s silhouette score on 8 Feb indicates it was likely appropriately placed in cluster 0, whereas Host D’s silhouette score on 8 Feb indicates it may have been inappropriately clustered in cluster 1 [39], [40].

Host	8Feb Cluster	...	11Feb Cluster	Days Active	Hops	Unique	8Feb_SS	...	11Feb_SS
Host C	0	...	0	4	2	2	0.78	...	0.68
Host D	1	...	4	3	2	3	0.11	...	0.24

Figure 10. Cluster analysis dataset with sample silhouette scores

Apart from using feature selection to identify each cluster’s most discriminatory features, we have focused on how network hosts are clustered relative to themselves. While this is useful for establishing per-host baselines, we need to address how hosts are clustered over time relative to other network hosts. For example, in isolation it may seem peculiar if a host appears in the same cluster for four consecutive days, but then suddenly changes on the fifth day to cluster 6. However, this could be viewed as normal relative to the entire population of hosts if we observe 50% of the population shift to cluster 6 on the same day. Such a massive distribution shift of hosts among each respective cluster is likely due to an outside factor (e.g., a mandatory online training event in which most hosts must participate, or a shift in business working hours) and not an isolated shift in host behavior. While understanding the specifics of such factors or events is not critical for anomaly detection, it is important to capture how individual hosts react to the imposition of these outside factors or events. Using our previous example, if a host does not shift to cluster 6, this could be cause for further investigation.

In order to reflect cluster distribution changes, and the significance of each host’s movement between clusters across days, we use a Markov chain to compute the relative probability of a host’s membership in a cluster on a given day. Given an observation period, a Markov chain calculates the probabilities of all the possible sequences of state changes. In machine learning, Markov chain models can be used to predict the most likely state

changes. In our case, cluster identifiers represent states. We consider the union of all possible clusters in a given period and note the percentage of hosts whom transitioned from one state to another. This allows us to measure the likelihood—or the suspiciousness—of a host’s cluster membership on a given day. Figure 11 uses a scaled-down example to show our methodology in analyzing how hosts move (transition) between clusters across three consecutive days.

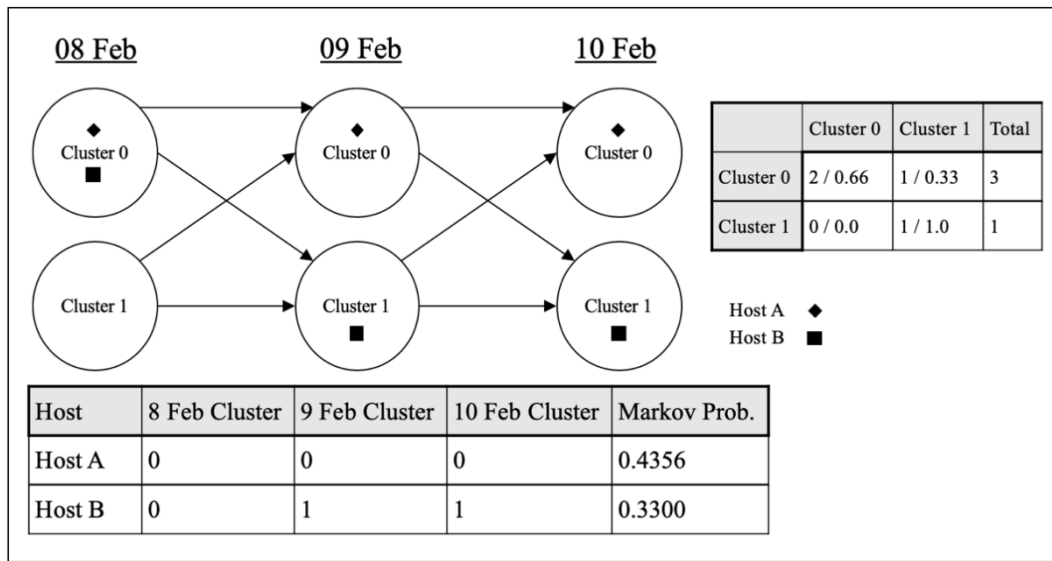


Figure 11. Markov chain network host cluster transitions

Figure 12 attempts to demonstrate the differences between host-specific and population-relative metrics with a simple example featuring 2 clusters across 3 days with 4 hosts. For population-relative metrics, Host T (triangle) would have the highest Markov probability because it follows the most likely transitions (as defined by the population) from 8 to 10 Feb. Host S (star) would likely have the highest silhouette score because it is consistently clustered close in proximity to its corresponding cluster’s centroid (solid black circles). For host-specific metrics, we might also say Host S conforms consistently with its established host baseline because it does not cluster hop and occupies 1 unique cluster for the entirety of the timeframe. Whereas Host T may not conform with its established per-host baseline because it hops clusters twice and occupies 2 unique clusters.

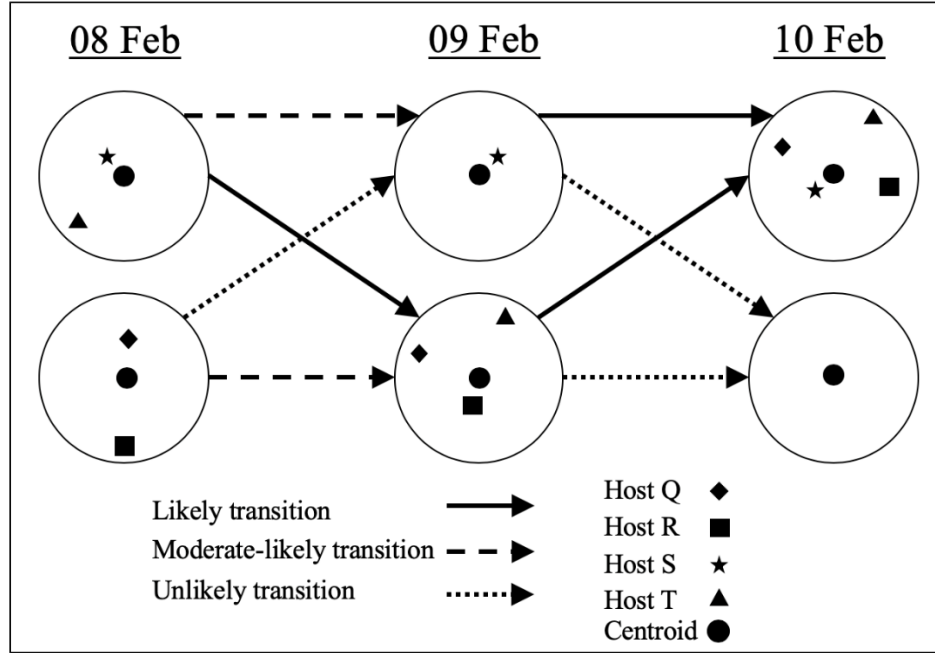


Figure 12. Population-relative and host-relative metrics example

The cluster analysis allows us to examine how a host is behaving relative to historical host trends and overall population trends. Examining the clusters themselves allows us to explain why a host or group of hosts is clustered in a given manner by describing the typical behavior demonstrated by the members of a given cluster. Given a sufficient observation period, our methodology allows us to derive both host-specific behavior baselines and the suspiciousness of a host's behavior relative to the population.

C. SUMMARY

This chapter described our methodology for extracting human and device behavior from network traffic in order to detect behavioral anomalies potentially associated with insider threats. First, we described how we engineer tailored feature sets which separately characterize human and device behavior, and how we use unsupervised machine learning—specially the K-Means algorithm—as the basis for detecting behavioral anomalies. We discussed the cluster metrics we use to describe how K-Means clusters network hosts, and the measurements we derive to form host-specific baselines and compare hosts to the population of hosts.

IV. SYSTEM IMPLEMENTATION

This chapter introduces a system framework that implements the methodology we discussed in Chapter III. This chapter also discusses how our system contends with large packet capture (PCAP) files to engineer the host-specific feature vectors, and how the unique nature of our dataset impacts our intended use of unsupervised machine learning clustering techniques to describe a host’s activity

A. INTRODUCTION

Our proof-of-concept system is designed to extract human and device behavior from large PCAP files and detect behavioral anomalies in accordance with the methodology we described in Chapter III. The focus of our system is to precisely characterize the behavior demonstrated by humans and devices on a network, and to extract this behavior in the form of descriptive feature vectors. In this section, we will introduce and discuss how we designed and implemented such a system. Figure 13 provides a broad, high-level overview of our system’s components and functionality.

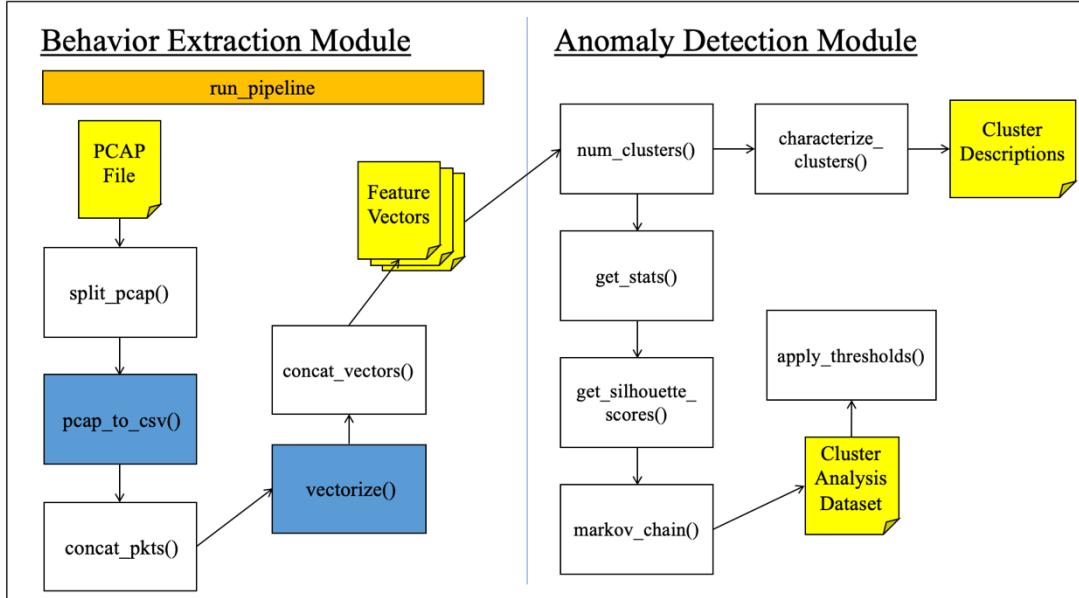


Figure 13. System components and functionality

In order to test our system on traffic from an operational network, we coordinated with the Naval Postgraduate School (NPS) Information Technology and Communications Service (ITACS) center to facilitate the collection of network traces in the form of PCAP files. On our behalf ITACS established a 10Gbps network tap that recorded approximately 5.5TB of PCAP files—nearly all the traffic that originated and ended inside the network over a 7-day timeframe. This tap recorded only the first 70 bytes of each packet; payloads were discarded. Of note, while the resulting PCAP had a non-zero packet loss during periods of high traffic volume, we assess this loss to be sufficiently low to test our proof-of-concept system.

B. NETWORK TRAFFIC PROCESSING AND BEHAVIOR EXTRACTION

The network traffic collection consisted of hourly PCAP files, with the maximum amount of network traffic per day totaling 919GB. In order to scale our system’s functionality to this volume, we utilized NPS’s High Performance Computer (HPC), the Hamming Supercomputer (HSC). To process the PCAP files, the behavior extraction module (Figure 13) operates on 24 PCAPs (1 day) at a time. Figure 14 depicts how we leveraged the parallelism offered by the HSC to transform arbitrary chunks from a PCAP file and reassembled them to form time-of-day, time interval, and country vectors.

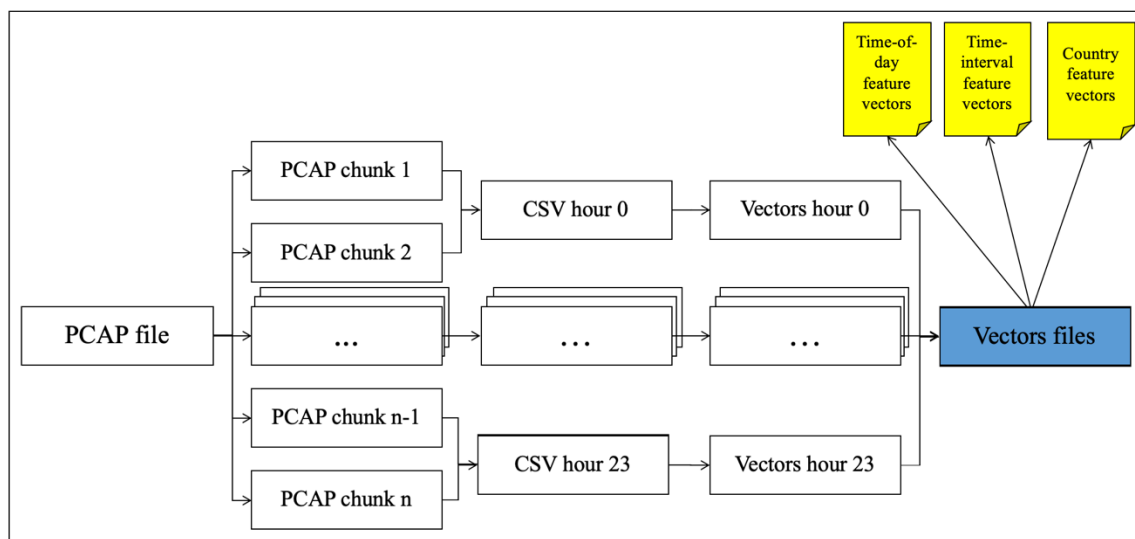


Figure 14. Processing network traffic in parallel

Run_pipeline() automates the behavior extraction module by utilizing the Slurm cluster management and scheduling software on the HSC [41]. After splitting hourly PCAP files into manageable sized fragments, the pcap_to_csv() component parses the PCAP fragments and extracts the required data to create each set of feature vectors for all observed NPS hosts. All observed TCP and UDP packet are represented in an intermediary state by a line with seven attributes, as shown in Figure 15. Packets utilizing other protocols are discarded.

Time	Source IP	Destination IP	Protocol	Source Port	Destination Port	Length
1219005803	26.12.3.89	26.12.0.1	17	49843	80	70
1219005804	26.12.3.90	26.12.0.1	6	49844	22	40
1219005805	26.12.3.81	26.12.0.1	6	49845	443	52

Figure 15. Intermediary TCP and UDP packet representations

While observing other protocols may be potentially useful (particularly for characterizing device behavior), our behavior extraction methodology requires port information as described in Chapter III. Furthermore, due to their dominance in our network traces—and in Internet traffic holistically—TCP and UDP packets provide the volume needed to precisely characterize behavior [42].

After mining the seven required attributes, the concat_pkts() component groups the packet representation files by hours 0 to 23 and the vectorize() component (Appendix A) transforms them into the respective time-of-day, time interval, and country feature vectors files. In order to reduce the number of columns and the overall dimensionality of the resulting feature vectors, vectorize defines a set of destination countries and a set of destination ports that are used consistently as features. As a result, each network host is characterized using common destination ports and countries. For the country feature vectors, the vectorize() component creates the set of destination countries by including all resolvable destination countries from a given day. Vectorize() utilizes MaxMind GeoIP to lookup the destination country for each destination IP address, discarding those that cannot be resolved and grouping the remaining destination countries in a designated bin [43].

For the time-of-day and time interval feature vectors, using all possible destination ports would result in 1,572,840 columns in the time-of-day feature vector files, and 13,041,465 columns in the time interval feature vector files. Given the number of network hosts we are considering each day, the resulting dimensionality would impose impractical computational costs in processing the data and detecting anomalies. Consequently, the system is currently designed to utilize the top-12 most frequently used ports during the observation period, but is adjustable to support any percentage or number. Ports that are not part of the top-12 are binned together and represented in the feature set. Specifically, for the time interval feature vectors, the time interval columns are logarithmically spaced from 0 seconds to 960 seconds (15 minutes) in order to capture transmission behavior at a granular level.

Of note, the current implementation of our system processes a larger than average daily network trace in approximately 8 hours on the HSC. Vectorize() is particularly costly, taking about 6 hours to run to completion under the current resource constraints on the HSC. While our implementation was not focused on performance, this is clearly a limitation in the context of insider threat detection. Ideally, such an insider threat detection system would be able to detect behavior anomalies in near real time, alerting analysts and security personnel as soon as possible. Admittedly, additional programming techniques and frameworks (several were discussed in Chapter II) could have been implemented in our system to speed-up our implementation.

C. CLUSTERING NETWORK HOSTS AND ANOMALY DETECTION

The feature vectors produced in the behavior extraction module enable efficient and rapid clustering and analysis of terabytes of PCAP files in the anomaly detection module. The anomaly detection module generates the data and context necessary to identify anomalous host behavior. As described in Chapter III, in order to detect behavioral anomalies, the anomaly detection module seeks to evaluate a host's behavior against the population's behavior. Determining a host's cluster membership, the number of unique cluster's they occupy, and their number of cluster transitions (hops) could also be useful in deriving host baselines over a given timeframe. Determining and evaluating host silhouette

scores and Markov chain probabilities is useful in understanding a host’s behavior relative to the population. Predictably, the conciseness of these evaluations is dependent on the length of the observation period. For example, shorter time periods (on the order of 7 or less days) are likely insufficient to accurately form host behavior baselines. However, 7 days is likely an insufficient duration for insider threat detection in an operational setting. In the absence of sufficient time to form representative host baselines, one would need to rely on the detection of behavioral anomalies relative to the population of network hosts. Considering the data at our disposal, we expect to rely more so on host silhouette scores and Markov chain probabilities to detect anomalous behavior. The data collection period resulted in 7-days’ worth of noncontiguous PCAP files. Table 2 summarizes the contiguous 6-day portion of the data and provides important operational context for the purpose of insider threat detection.

Table 2. Data collection overview

Date	8 Feb	9 Feb	10 Feb	11 Feb	12 Feb	13 Feb
Day	Saturday	Sunday	Monday	Tuesday	Wednesday	Thursday
DHCP NPS Hosts	4,435	4,435	7,423	7,631	7,497	7,327
Static NPS Hosts	479	498	490	484	519	526

This table provides an overview of the data we will use to test our system and imparts the feasibility of detecting anomalous behavior for the purpose of insider threat detection to include forming representative host behavior baselines. “DHCP Hosts” and “Static Hosts” show the number of active DHCP leased or statically assigned IP addresses on each respective day.

The noncontiguous nature of our test data demonstrates several benefits and challenges we may experience in detecting behavior anomalies. The mix of week and weekend days allows us to infer network-wide and host-specific trends demonstrated on working and non-working days. Furthermore, 6 consecutive days of observation (8 through 13 Feb) allows the formation of at least partially representative host baselines. However, the lack of repeated same-day observations (with the exception of 8 and 15 Feb) do not

allow us to compare host-specific behavior demonstrated on specific days of the week. Also, the lack of a fully contiguous observation period (no collection on 14 Feb) denies us from measuring cluster transition probabilities to 15 Feb with a Markov chain.

Based on our chosen methodology and nature of our data, we assess our system needs at least 2 days of observation to identify potentially anomalous behavior. In fact, with the exception of silhouette scores, the metrics our system utilizes are practically irrelevant without at least 2 days of observation. As a result, we intend to evaluate our system over the 6 days of continuous capture (8 through 13 Feb). Considering the population-relative metrics we intend to use, we assess our chosen methods lend themselves more so to detecting anomalies across a contiguous timeframe.

While not attributed to the nature of the data at our disposal, evaluating hosts whom are using DHCP-leased addresses presents another challenge in our analysis. Depending on the leasing period enforced by the network (e.g., wireless DHCP addresses change daily, wired DHCP addresses change less often), we should expect DHCP-leased addresses to be used by different hosts on a daily basis. As a result, behavior from leased addresses over a period of time may appear erratic in our analysis whereas in reality it could be behavior demonstrated by several different hosts. We expect this to be particularly true as it applies to leased wireless addresses. However, we expect leased addresses assigned to wired devices to change infrequently—particularly within the short period represented in our data. For this reason, when identifying hosts whom appear to be demonstrating anomalous behavior, we evaluate leased and static addresses differently, and further differentiate wired and wireless leased addresses. We have chosen to evaluate static and wired DHCP-leased hosts with the assumption that the wired DHCP-leased hosts will seldomly be assigned a new IP address during the 6-day observation period. This assumption is based on our experience using the network. Furthermore, we assess the few new IP address assignments to wired DHCP-leased hosts that occur during the observation period will negligibly alter our results.

Broadly, the anomaly detection module (as shown in Figure 12) combines feature vectors of a single type across a chosen time period and clusters network hosts using the K-Means algorithm after determining the ideal number of clusters in the `num_clusters()`

component using silhouette scores. It subsequently attempts to characterize the outputted clusters according to their most descriptive features in the `characterize_clusters()` component. Characterizing the outputted clusters is not done specifically for the purpose of anomaly detection. Instead, cluster characterization allows us to understand why clusters are formed in a given manner and determine the descriptiveness of the clusters' top features in order to gain operational insight.

As described in Chapter III, the `get_stats()`, `get_silhouette_scores()`, and `markov_chain()` components compute the number of cluster transitions, the number of unique clusters occupied, the sample silhouette scores, and Markov chain probabilities for a single host over a given timeframe. In order to reduce the number of 0% probability cluster transitions, we use a different K-Means model in the `markov_chain()` component from the model used in the `get_silhouette_scores()` and `characterize_clusters()` component. We reduce the k in K-Means model used to produce cluster transition probabilities so it does not exceed 8. In cases where the `num_clusters()` component selects a k smaller than 8, we use the k selected by `num_clusters()`. In cases where the outputted k from `num_clusters()` exceeds 8, we must consider the Markov transition probabilities in isolation without the context provided by `characterize_clusters()`.

The silhouette score computed for each host for each day are averaged to provide a single representative value for how closely hosts are related to their assigned cluster. The `apply_thresholds()` component identifies the most suspicious hosts by filtering hosts with the lowest Markov chain probabilities and mean silhouette scores. To a lesser degree, the number of host cluster transitions and unique clusters occupied are also considered.

D. SUMMARY

In this chapter, we introduced our system design. Our system design implements our methodology and successfully contends with the real-world challenges associated with working with network traffic and machine learning to identify potential behavioral anomalies associated with insider threats. Specifically, we discussed details regarding scaling our system to process terabytes of real-world network traces, and the challenges and solutions associated with dealing with high-dimension feature vectors. Lastly, we

outlined how we intend to test our system across the 6-day period of contiguous network activity, and how we intend to deal with hosts with leased versus statically assigned IP addresses.

V. RESULTS AND ANALYSIS

This chapter presents and discusses the system’s results. First, we provide an analysis of the system’s results in respect to network hosts and clusters. Next, we present a series of case studies where we seek to explain why a host is indeed behaving anomalously when the system flags a host as such.

A. OVERVIEW

This section states and examines several broad, network-wide conclusions we can draw using our behavior extraction methodology. In short, we provide a holistic view of the network’s behavior—in some cases digging deeper to explain our conclusions.

As described in Chapter IV, the data at our disposal encompasses a 7-day, noncontiguous period. In order to test the system in a realistic manner, and ensure the Markov chain is utilized as intended, we chose to test the system over the contiguous period from 8 to 13 Feb. Also, for the sake of the Markov chain metrics, we have chosen to evaluate DHCP network hosts that were active for all days in the chosen period under the assumption that most DHCP hosts do not change IP addresses during the 6-day observation period (and those that do change will negligibly alter our results).

1. Host Analysis

The system observed 8,838 unique IP addresses over the 6-day observation period. 537 were hosts with statically assigned addresses, 3,305 were hosts with DHCP wireless addresses, and 4,996 hosts had DHCP wired addresses. Figure 16 shows that most of the hosts were active for all days of the observation period. Assuming most of the wireless subnet hosts are not on campus Saturdays and Sundays, it is expected that the majority of the wireless hosts were active on all 4 of the weekdays we observed.

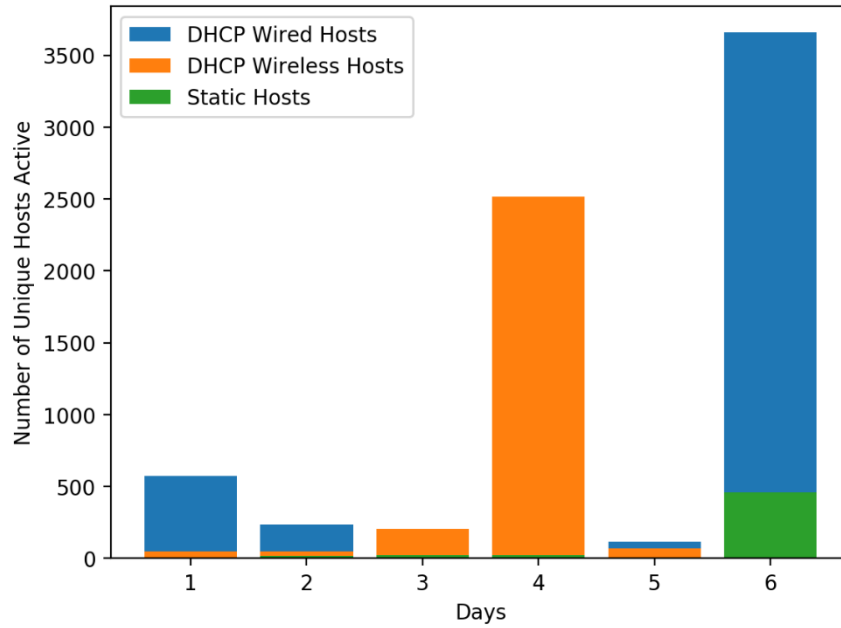


Figure 16. Host activity

Figures 17 and 18 show the network-wide most prevalent packet interevent times and port usage according to the average population host over the observation period. The top-5 interevent times are dominated by small ranges—the smallest possible range (0.0– 1.1×10^{-6} seconds) being the most prevalent. As shown in Figure 24, the Server Message Block (SMB) protocol over port 445 is most prevalent. The SMB protocol facilitates file sharing and is commonly used on typical intranets. Of note, in Figure 18 we have excluded the other binned ports the system did not enumerate. The average population host sent approximately 22,000 packets over other destination ports during the observation period.

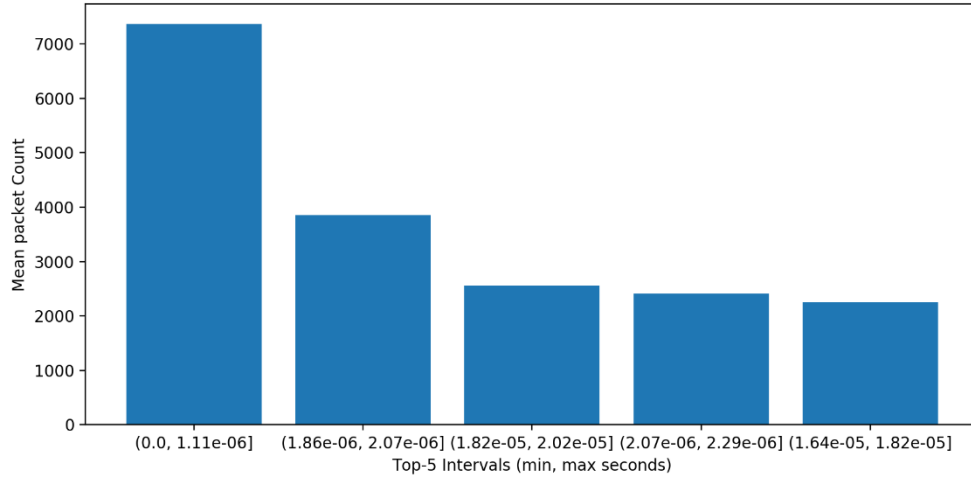


Figure 17. Average population host interevent analysis

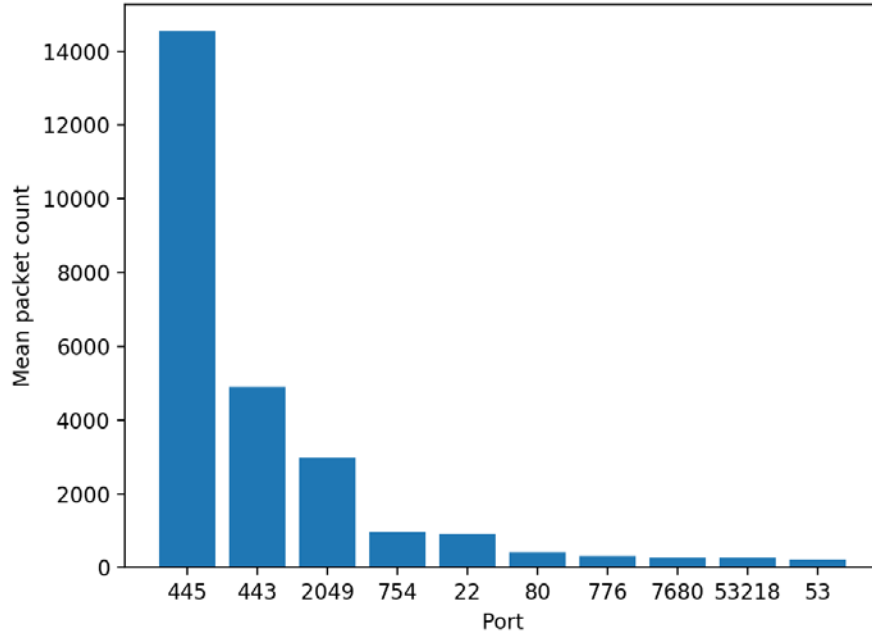


Figure 18. Average population host destination port usage

With the selected number of k clusters for the time-of-day feature vectors being double the k that was selected for the time interval feature vectors, it is expected that the time-of-day hosts are more likely to occupy a higher number of clusters throughout the period. As shown in Figure 19, the greatest proportion time-of-day hosts occupy 6 distinct clusters throughout the observation period. Whereas the greatest proportion of time interval

hosts occupy 1 distinct cluster throughout the observation period. Again, commensurate with k , country vector hosts never occupied more than 3 distinct clusters.

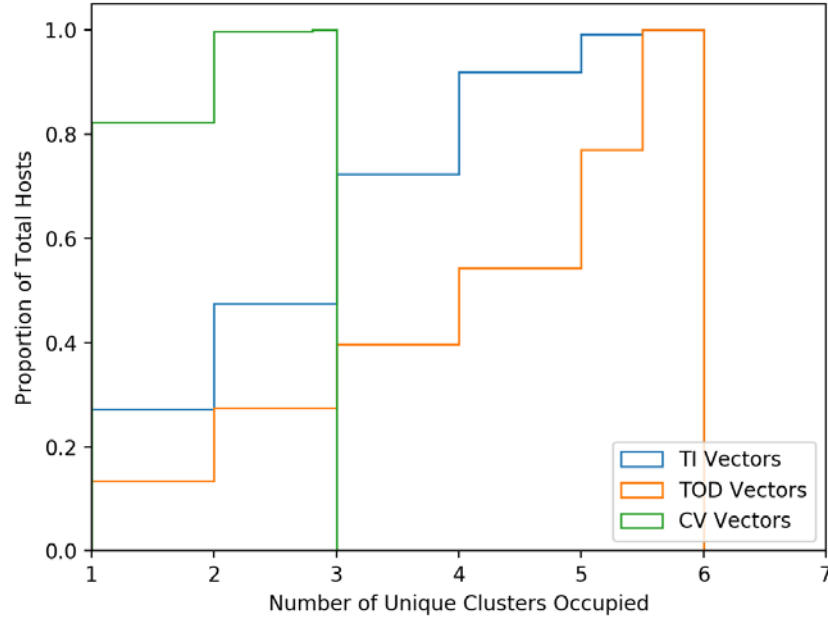


Figure 19. Clusters occupied CDF

Similar to Figure 19, Figure 20 shows time-of-day hosts were most likely to change their cluster membership (cluster hop) the maximum amount of times (5 transitions for a period of 6 days), whereas time interval hosts were most likely to never hop clusters throughout the observation period. In isolation, the number of unique clusters a host occupies or the number of times a host changes cluster membership reveals behavior variance, but is not particularly useful for the purpose of insider threat detection without additional context. A host's corresponding Markov chain probability and mean silhouette score computed over the observation period provides context which allows us to determine the significance of host cluster movement. For example, if not rationalized by population-relative metrics which characterize the population's behavior (in our case, Markov chain probabilities), a host changing cluster membership could be significant. Host-relative metrics could also help explain cluster movement. In this case, having host-unique baselines could help justify cluster movement if the movement is consistent with a host's

past behavior. However, we assess our 6-day observation period is insufficient for developing host-unique behavior baselines. The lack of repeated same-day observations (i.e., multiple Monday observations) in our data and the data’s brevity hinder our ability to build host-unique behavior baselines. As a result, we will rely on the context provided by Markov chain probabilities and mean silhouette scores to drive the identification of behavioral anomalies.

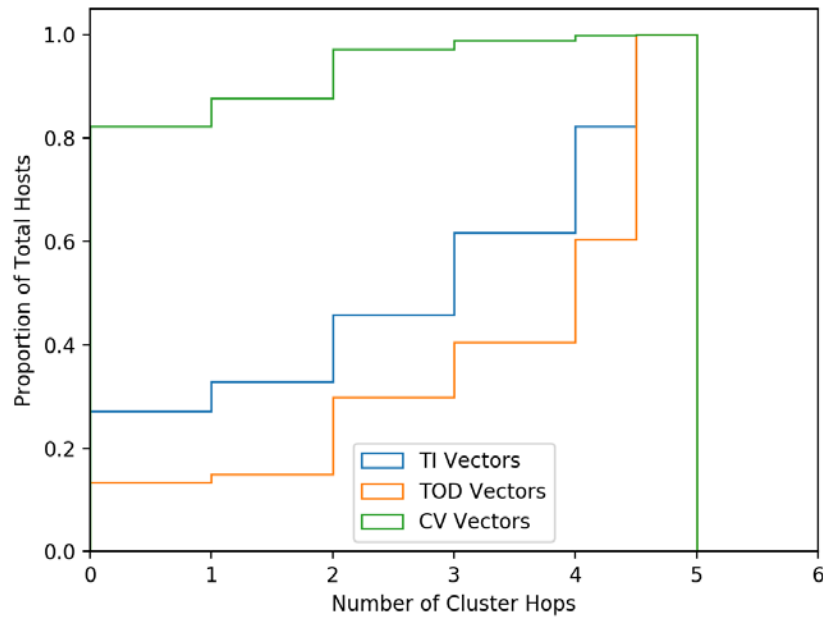


Figure 20. Cluster hops CDF

Figures 21 and 22 show the cumulative distributions of mean silhouette scores and Markov chain probabilities for the data. These graphs are the basis for the `apply_thresholds()` component in the system’s anomaly detection module. Our system selects the hosts with the lowest scores (1%) as potentially anomalous and flags them for further investigation. The highest-scoring flagged time-of-day hosts (in theory, demonstrating the least anomalous behavior of the flagged hosts) have a mean silhouette score of 0.3051 and a Markov chain probability of 9.64×10^{-7} , respectively. The corresponding time interval hosts have a mean silhouette score of 0.1962 and a Markov chain probability of 5.17×10^{-8} , respectively. Finally, the corresponding country vector

hosts have a mean silhouette score of 0.6900 and a Markov chain probability of 2.09×10^{-5} , respectively.

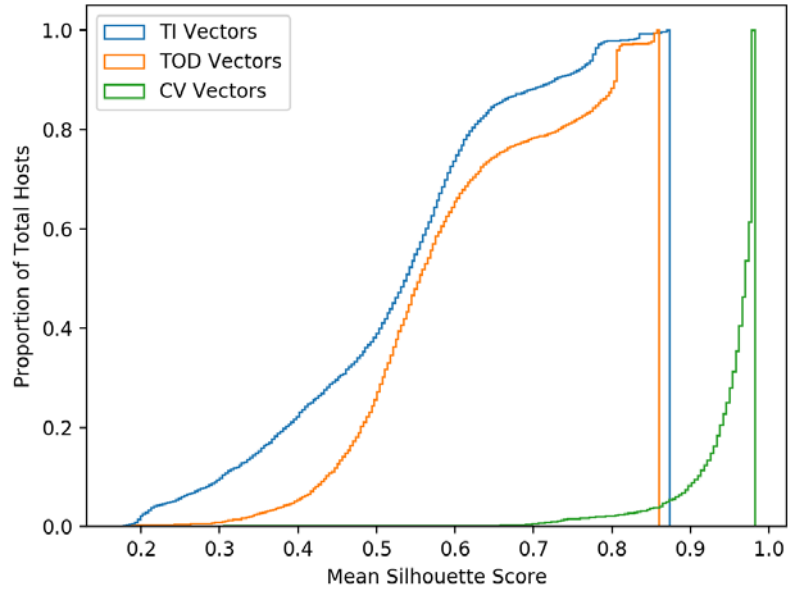


Figure 21. Mean silhouette scores CDF

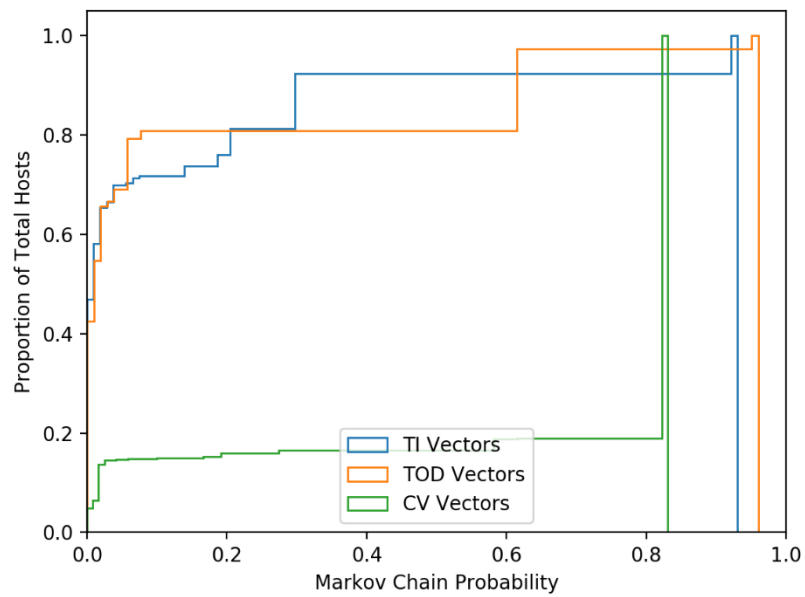


Figure 22. Markov chain probabilities CDF

2. Cluster Analysis

We use silhouette scores to determine the ideal number of clusters. Figure 23 shows the average silhouette scores for 4 to 150 clusters per each feature vector type. In order to limit the number of clusters (k), we select the highest scoring k less than or equal to 70. In the case of the time-of-day vectors, only marginally higher silhouette scores are achieved over $k=40$. We describe the clusters from each feature vector type separately using the cluster analysis methodology (`characterize_cluster()` component) described in Chapter III in the following paragraphs.

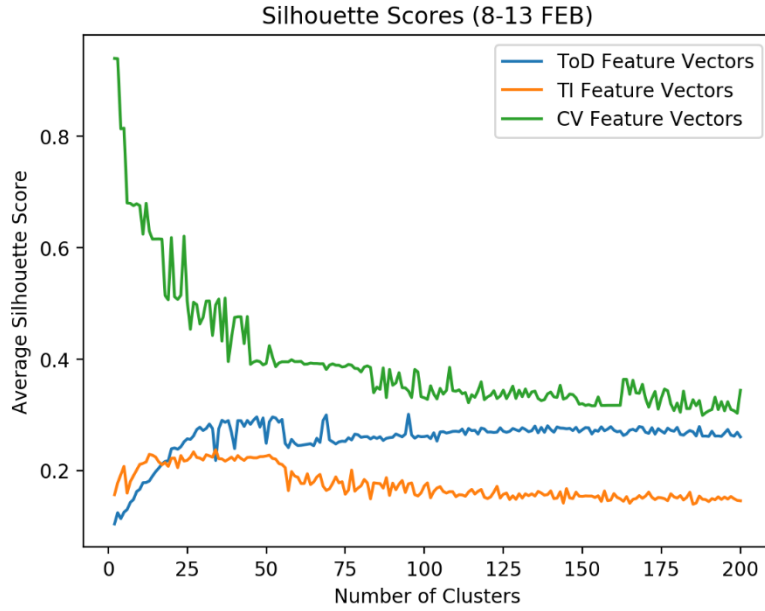


Figure 23. Average silhouette scores per feature vector type

Table 3 summarizes the 65 clusters for the time-of-day feature vectors. 60 of the 65-top scoring (most discriminative) features for each cluster were unique, which shows significant cluster diversity. The decision tree metrics (described in Chapter III) show how well a decision tree can classify hosts from a given cluster using a one-versus-all strategy. The metrics from each cluster's decision tree are averaged below in Table 3.

Table 3. Time-of-day cluster analysis

Cluster distribution	Largest cluster (hosts)	5563
	Smallest cluster (hosts)	106
	Median cluster size (hosts)	435
	Mean cluster size (hosts)	642.22
Cluster distinctiveness (decision tree metrics)	Mean accuracy	0.995
	Mean precision	0.919
	Mean recall	0.920

As designed, the clusters’ top scoring features allow us to characterize a given cluster’s hosts. For example, cluster 17’s top-3 features include port 53 during hours 16, 17, and 18. Figure 24 shows the comparatively high port 53 usage associated with hosts in cluster 17, likely driven by a relatively high number of DNS queries during working hours.

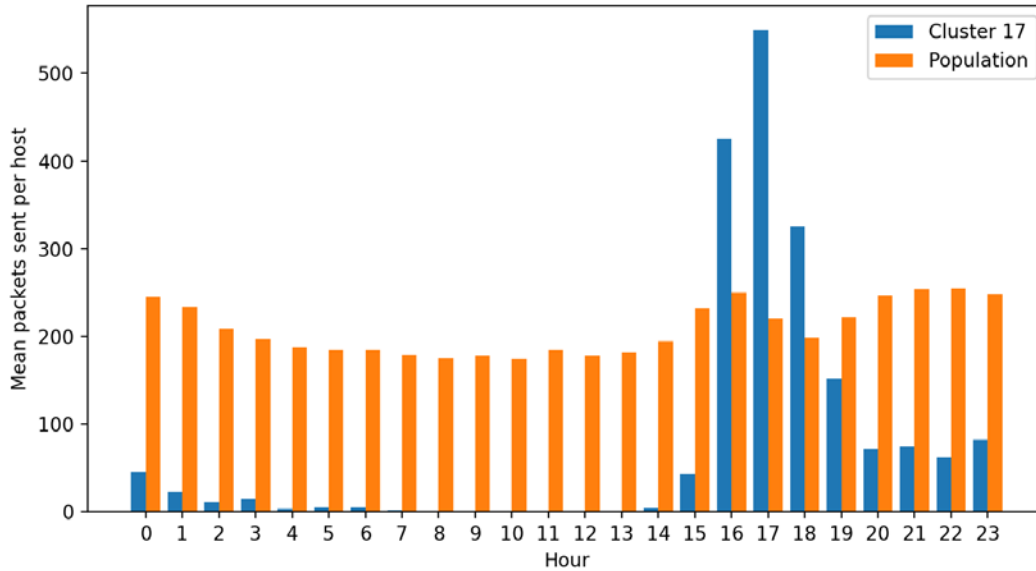


Figure 24. Cluster 17 DNS activity

In addition to providing information about preferred network services, as expected the time-of-day feature vectors allow us to infer a sense of daily schedule and rhythm. For example, observing the per-hour network activity in Figure 25 generated by hosts in the

wireless subnet shows a clear circadian rhythm. We assess the majority of hosts in the wireless subnet are humans and not purely automated devices. Adding 8 hours to the values shown on the x-axis in Figure 25 shows local pacific standard time.

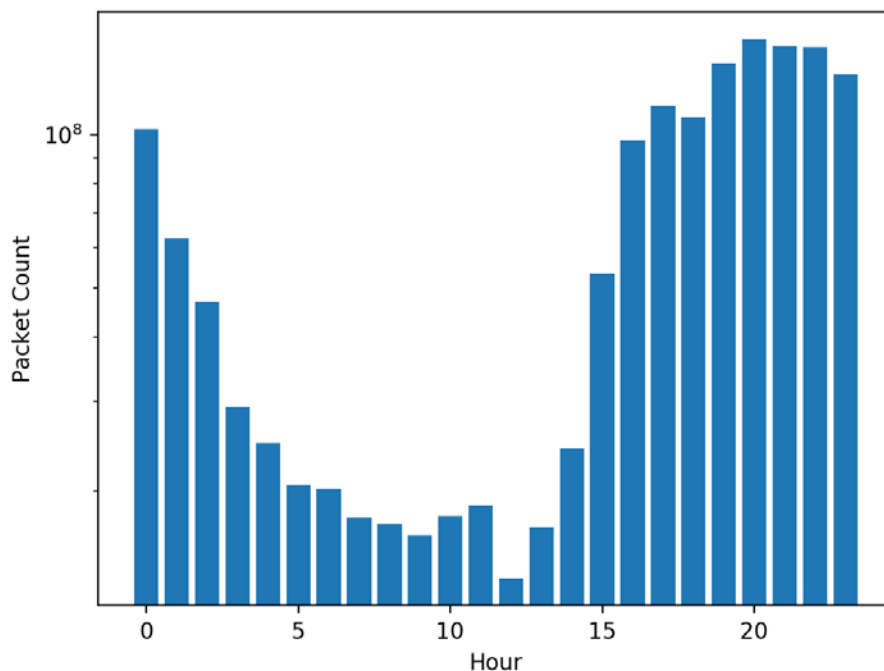


Figure 25. Activity patterns of hosts in the wireless subnet

Table 4 summarizes the 27 clusters for the time interval feature vectors. Twenty-two of the 27-top scoring features for each cluster were unique. In accordance with its design, the time interval features allow us to differentiate between clusters primarily comprised of mostly automated hosts demonstrating device behavior, and clusters primarily comprised of hosts controlled by humans which demonstrate human behavior. For example, as shown in Figure 26, cluster 19 hosts demonstrate repeated activity in one particular time interval. This potentially indicates higher levels of automation in cluster 19 hosts. In Figure 27, we contrast the distribution of activity among the wireless subnet’s top-5 intervals to cluster 19’s respective (different) top-5 intervals. Assuming hosts in the wireless subnet are controlled by humans and demonstrate lower levels of automation, as we might expect the wireless subnet’s activity is more evenly distributed compared to

cluster 19. As shown in Figure 27, the proportion of cluster 19’s activity in its top interval nearly doubles the wireless subnet’s activity in its top interval. This activity allows us to assume cluster 19 is primarily comprised of hosts with higher levels of automation.

Table 4. Time interval cluster analysis

Cluster distribution	Largest cluster (hosts)	5021
	Smallest cluster (hosts)	122
	Median cluster size (hosts)	1139
	Mean cluster size (hosts)	1546.07
Cluster distinctiveness (decision tree metrics)	Mean accuracy	0.987
	Mean precision	0.926
	Mean recall	0.922

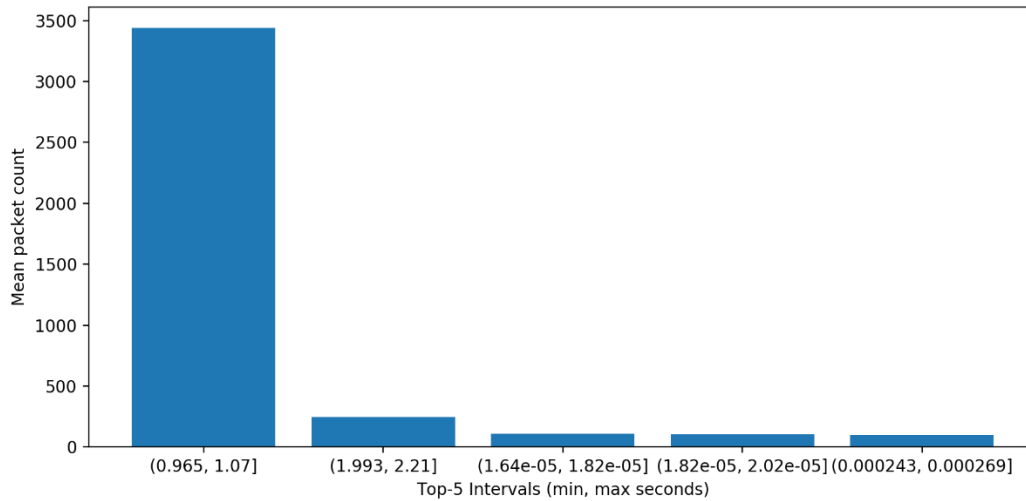


Figure 26. Cluster 19 interevent analysis

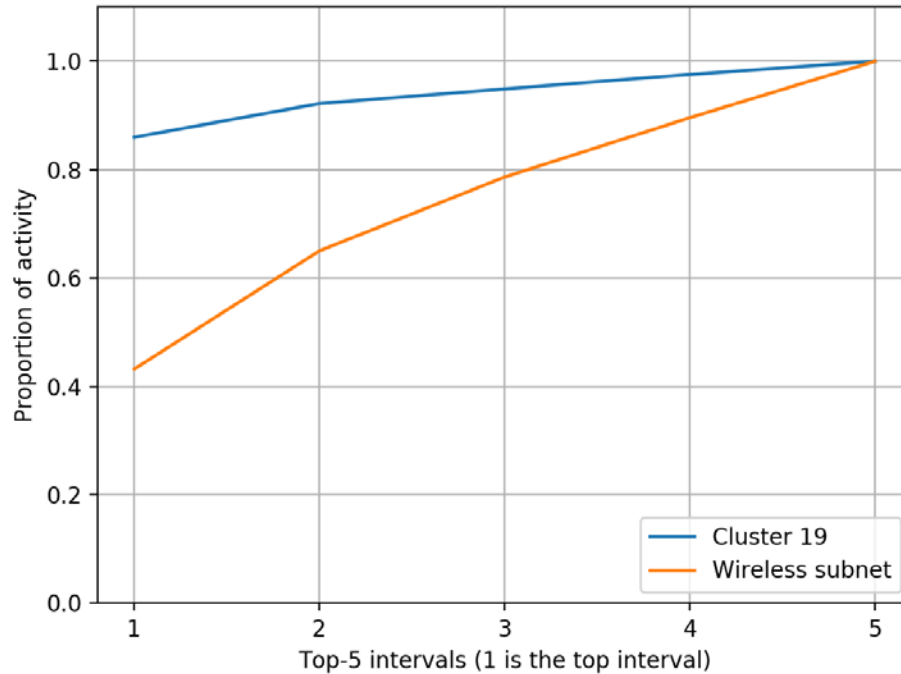


Figure 27. Interval distribution CDF

Table 5 summarizes the 5 clusters for the country feature vectors. Only 3 of the top-5 scoring features for each cluster were unique. Initial indications suggest the current design of the country features vectors is not advantageous for behavioral anomaly detection. One observation that could help explain this is that the vectors are particularly skewed with the vast majority of packets and data being sent to destinations in the United States as shown in Figure 28.

Table 5. Country cluster analysis

Cluster distribution	Largest cluster (hosts)	32936
	Smallest cluster (hosts)	67
	Median cluster size (hosts)	214
	Mean cluster size (hosts)	6879.20
Cluster distinctiveness (decision tree metrics)	Mean accuracy	0.997
	Mean precision	0.972
	Mean recall	0.931

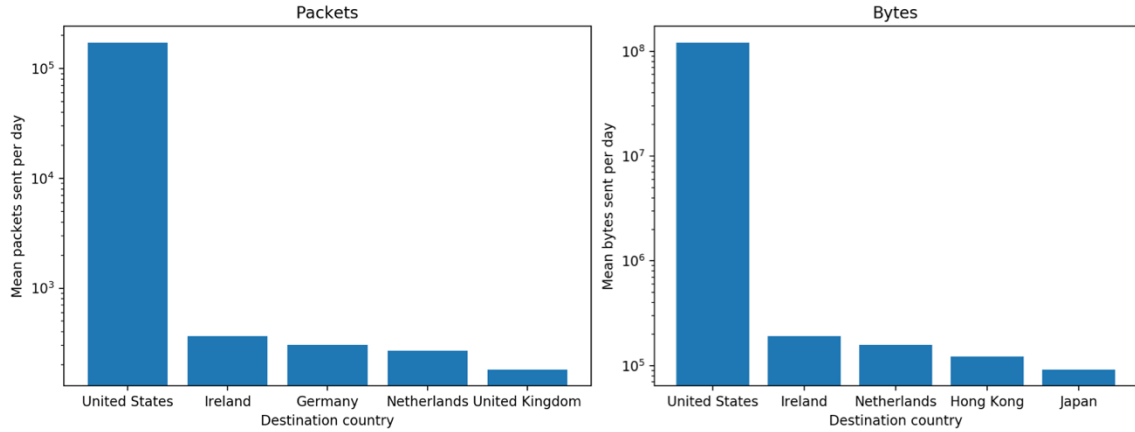


Figure 28. Top country feature vector destinations

B. CASE STUDIES

The primary objective of the following case studies is to determine if our methodology is effective in identifying behavioral anomalies. Second, through these case studies we hope to achieve an understanding of the advantages and disadvantages offered by each feature vector type in identifying behavioral anomalies. Third, we will seek to understand which population-relative metric (mean silhouette scores and Markov chain probabilities) is more effective in identifying behavioral anomalies.

In the following case studies, we select and examine hosts the system has flagged as potentially anomalous. Specifically, we select one flagged host from each feature vector type for further investigation in an attempt to determine if indeed they are demonstrating anomalous behavior. The mean silhouette scores and Markov chain probabilities are the first indicator that a host could be behaving anomalously relative to the population of network hosts. However, we use a variety of techniques in the following case studies in order to assess whether a flagged host is behaving anonymously or not. Our techniques include host subnet identification and examination, observing movement in and/or between clusters, and observing the raw network traffic generated by a host. We primarily examine the hosts' features which led to it being flagged. For example, we examine per-hour activity and port usage for the time-of-day hosts. However, we circumstantially examine other characteristics if they appear relevant to an assessment of the host in question.

Perhaps most importantly, these metrics and observations are always compared to the population-relative metrics. We also (at times) compare a host to the hosts in its subnet to provide additional operational context—especially when the subnet’s purpose is known and understood. When possible, we attempt to account for how a particular host is behaving compared to how it has behaved in the past. However, given the duration and composition of our data, we consider host-relative metrics to be less relevant.

1. Time-of-Day Feature Vectors – Host A

Host A was flagged by the system due to its low Markov chain probability. As shown in Figure 29, Host A earned the lowest Markov chain probability and a mean silhouette score of 0.4466 which is at the 14th percentile of all observed hosts. Of note, the next lowest Markov chain probability was 1.08×10^{-9} —significantly higher than Host A’s.

8 – 13 Feb Clusters	Hops	Unique	Days Active	Mean SS	Markov Prob.	Type
5 2 4 5 2 5	5	3	6	0.4466	4.08×10^{-11}	DHCP wired
Transition Probs.	0.0014	0.0241	0.4516	0.0014	0.0018	

Figure 29. Host A’s cluster analysis

Examining Host A’s cluster movement (reflecting erratic behavior) helps explain its low Markov chain probability. However, while other hosts had 5 hops and occupied 6 unique clusters, none had a lower Markov chain probability. From this we can conclude Host A made exceedingly unlikely cluster transitions across the observation period—as shown in Figure 29. Figure 30 helps further explain Host A’s low Markov chain probability and its unlikely transitions. Figure 30 shows Host A’s contradictory activity patterns using volume. With the exception of the 8th to the 9th and the 12th to the 13th, Host A’s activity increases when the population’s mean decreases and vice versa. Particularly noticeable is Host A’s divergence from the population from the 9th to the 10th where on the 10th Host A’s activity differs from the population mean by approximately 20,000 packets.

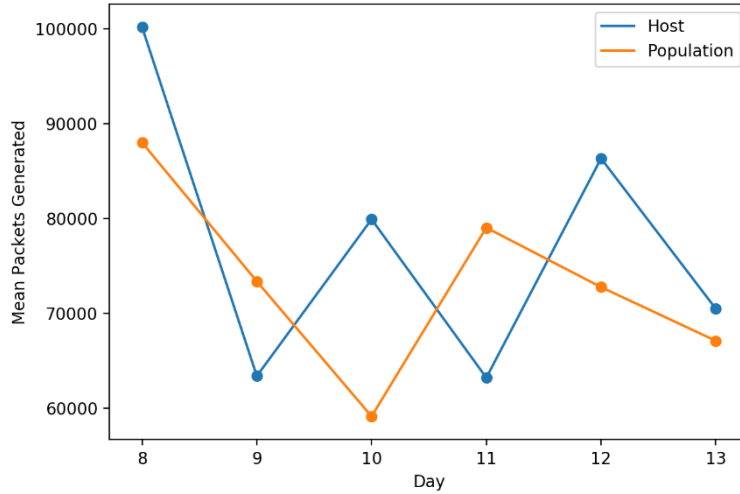


Figure 30. Host A’s volumetric activity patterns compared to the population

While Figure 30 shows Host A’s volumetric inconsistencies relative to the population, we must address the other metrics accounted for in the time-of-day feature vectors. Host A also appears to be behaving anonymously based on time metrics. Figure 31 shows Host A’s per-hour activity across the entire observation period compared to the population’s mean activity. There are spikes in activity from Host A during hours 0, 1, 7, and 11. However, perhaps most striking is Host A’s divergent activity between hours 17 and 21 — one of the population’s lowest activity periods.

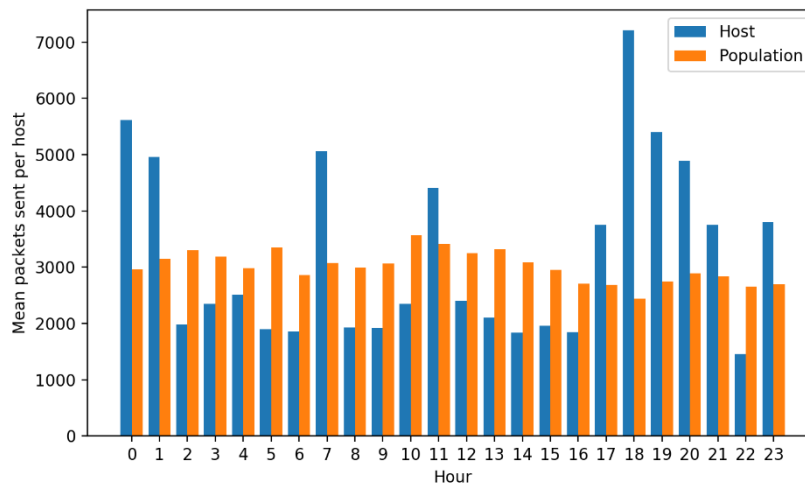


Figure 31. Host A’s per-hour activity compared to the population

Using the information gleaned from both Figures 30 and 31, we can examine Host A’s activity on the 10th during hours 17 through 21 compared to the population mean on the 10th during the same hours. As shown in Figure 32, (showing differences in per-hour activity between the 9th and 10th) the activity exhibited by Host A during hours 17 through 21 does significantly differ from the population mean. However, Figure 32 also shows stark differences during other hours on the 9th and 10th. Furthermore, Figure 32 also reveals possible host-relative anomalies in Host A’s behavior. Without a Host A behavioral baseline to refer to with repeated same-day observations, we cannot claim this is a host-relative behavioral anomaly, but inconsistencies during hours 8 through 17 on the 9th compared to the 10th, as well as the spike in traffic following a period of low traffic volume, do warrant further investigation.

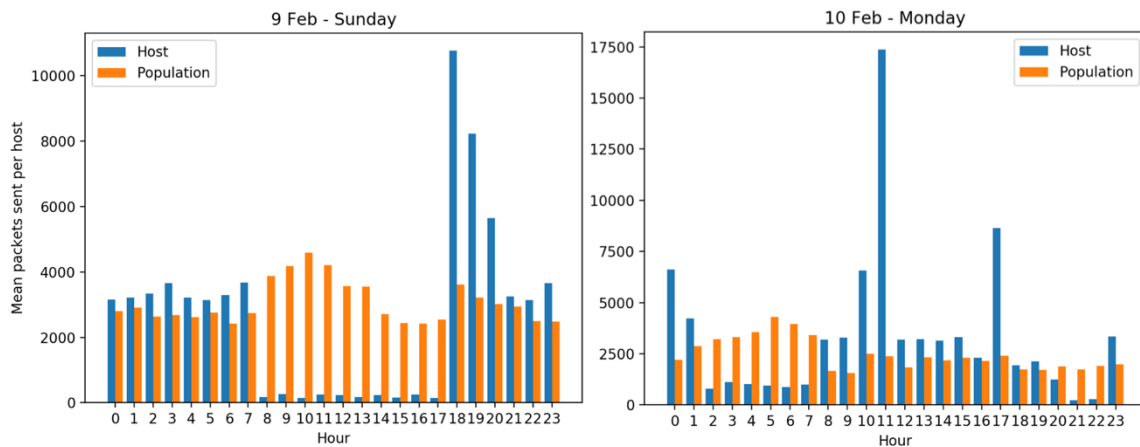


Figure 32. Host A’s per-hour activity compared across days

Host A divergences concerning port usage are less stark, highlighting the sensitivity of our methods to temporal behavior rather than port behavior. More specificity in the ports we enumerated, specifically the ports grouped together in the “other” column in the time-of-day feature vectors, could have resulted in more telling results. However, closer examination of Host A’s most frequently utilized ports—binned other ports and 445—compared to the populations yields notable results. Examining Host A’s overall port activity compared to the population shows that Host A uses port 445 significantly more. Looking at this activity per-hour shows more striking results. Figure 33 shows Host A’s

sharp increase in port 445 activity between hours 15 and 20 while the population's mean port 445 activity decreases. Also, while Host A's other port activity increase aligns with the population increase, it increases well above that of the population.

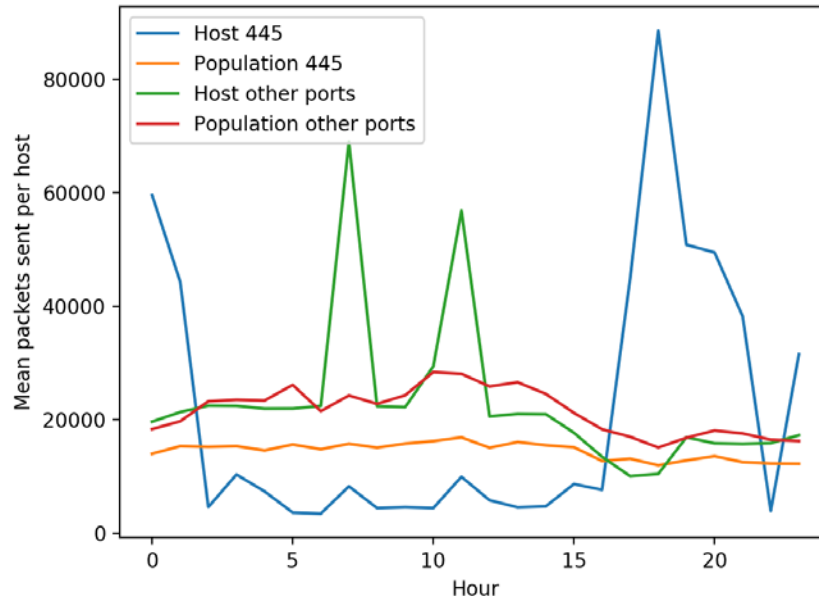


Figure 33. Host A port activity compared to the population

We have shown that Host A, at times, demonstrated anomalous behavior relative to the population of network hosts. However, this is in comparison to a population that contains a wide range of device types (servers, printers, phones, etc.). Comparing Host A to other hosts in its subnet may reveal differences that are either location or device specific. We observed activity from at least 460 other hosts in Host A's subnet, and 12 of the 40 hosts flagged by the system for low Markov chain probabilities were members of Host A's subnet. Based on this information, it is possible an unusual subnet-wide event prompted the subnet to behave unusually during the observation period. Figures 34 and 35 help explain Host A's anomalous activity. Figure 34 shows synchronized activity changes with the exception of the 12th to the 13th. Figure 35 also shows mostly synchronized activity shifts on an hourly basis between Host A and the subnet mean. The similar activity shifts between consecutive hours and consecutive days could indicate a shared pattern or event that influences the entire subnet's (including Host A) activity.

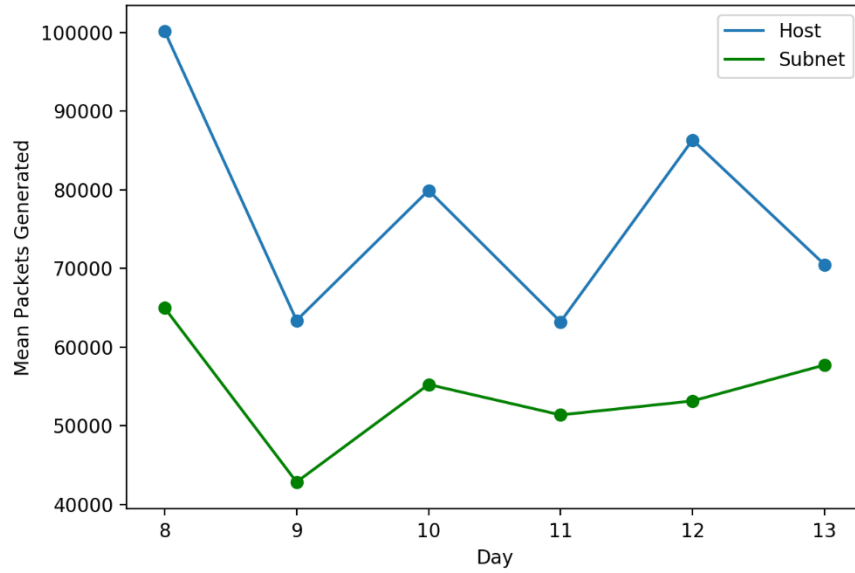


Figure 34. Host A’s daily activity compared to its subnet

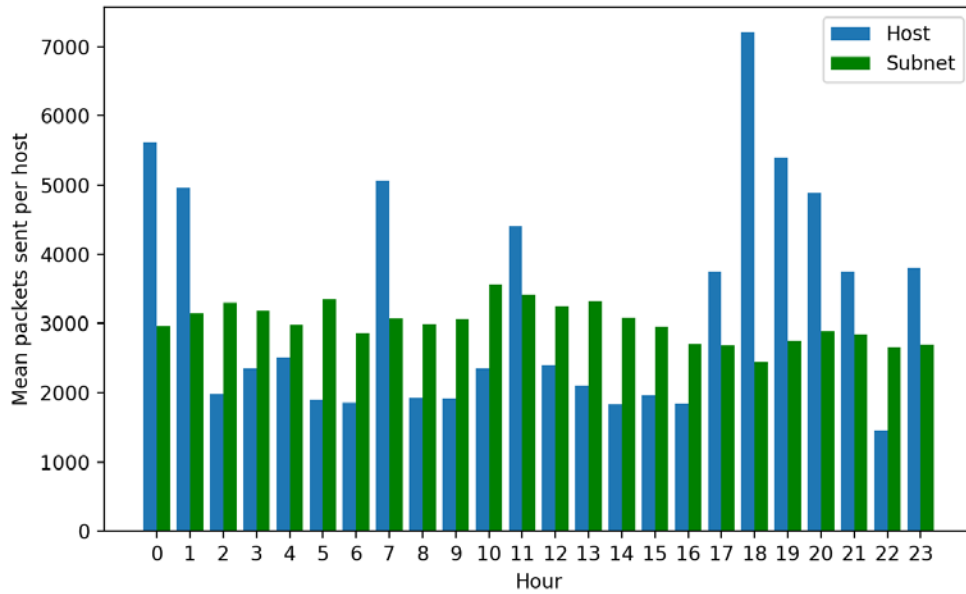


Figure 35. Host A’s per-hour activity compared to its subnet

This does not explain the greater volume demonstrated by Host A compared to the subnet mean—particularly between hours 17 and 21. Host A has the resoundingly lowest Markov chain probability and has a mean silhouette score in the 26th percentile of all subnet hosts. Port usage could help explain this, drawing a distinction between the subnet’s

behavior. As shown in Figure 36, while port 445 and port 443 activity is similar, Host A has significantly more other port activity, and less port 22 activity.

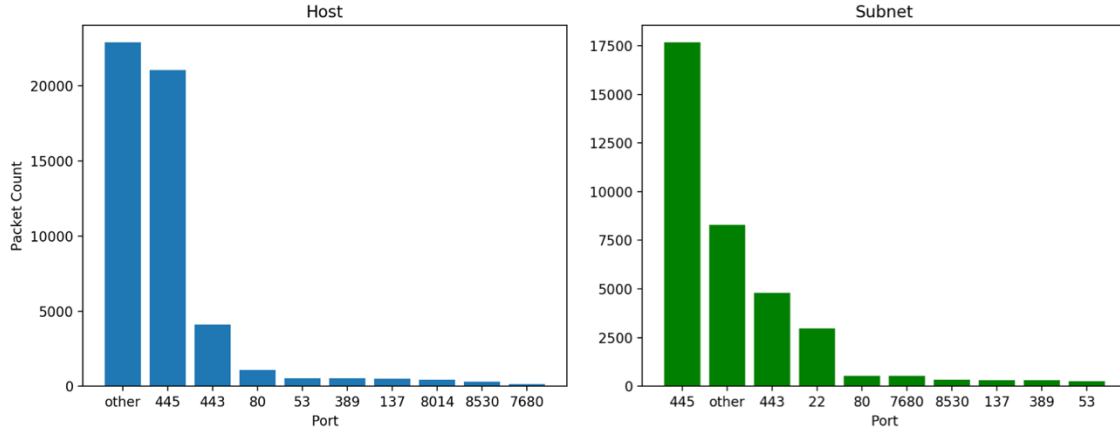


Figure 36. Host A's port activity compared to the subnet

We have shown that Host A's behavior is significantly different from the population of network hosts during the observation period. Shear activity levels (based on volume) between Host A and the population contradict one another on a daily and hourly basis. We have also shown that while Host A and its subnet have mostly corresponding daily and hourly activity patterns, Host A's port utilization and volume is significantly different from its subnet.

2. Time Interval Feature Vectors – Host B

Host B was flagged by the system due to its low mean silhouette score. Host B has the lowest mean silhouette score of all the flagged time interval hosts. As seen in Figure 37, Host B appeared in the same cluster on all days in the observation period, resulting in 0 cluster hops. Despite this lack of cluster movement and consistent behavior, its Markov chain probability appears below the mean (0.4230) of other hosts that had 0 cluster hops. Of note, Host B consistently appears in a large cluster that does not appear to be very well defined. This is potentially a result of the fact that K-means places all points into a specified number of clusters, sometimes resulting in grouping of samples despite potential

dissimilarities. Cluster 13’s most discriminating features allowed the system to correctly classify 90% of cluster 13 hosts—well below the 98% average of all time interval feature vector clusters. For additional context, we were provided information that Host B is in a subnet used for energy management controllers associated with heating, ventilation, and air conditioning on campus. We observed activity from 57 other hosts in this subnet. These 57 hosts appear in in cluster 13 more than any other cluster.

8 – 13 Feb Clusters	Hops	Unique	Days Active	Mean SS	Markov Prob.	Type
6 6 6 6 6 6	0	1	6	0.1742	0.3019	DHCP wired
Transition Probs.	0.7864	0.7864	0.7864	0.7864	0.7864	

Figure 37. Host B cluster analysis

Given this context and the unique nature of Host B, we might expect its behavior to be vastly different from the behavior of the population. Furthermore, given cluster 13’s lack of distinctive features compared to other clusters, we should expect more variability in behavior from hosts in cluster 13. For these reasons, it is unlikely that comparisons between Host B and the population, or Host B and other hosts in cluster 13 will be useful from an insider threat perspective. However, this likely indicates that Host B is indeed demonstrating anomalous behavior in respect to the population and cluster 13.

As suspected, Host B’s behavior significantly differs from the population. Figure 38 shows Host B is significantly less active than the population’s average host. It also shows its top-5 intervals do not intersect with those of the population’s average host. Of note, Host B’s top-5 interevent periods are all larger than the population’s top-5 interevent periods. Regarding port utilization, Host B exclusively uses ports outside of the top ports we enumerated. Through the same methodology, we find that the average cluster 13 host is more similar to the population’s average host than Host B—demonstrating a similar activity level and having the same top-2 interevent periods as the population.

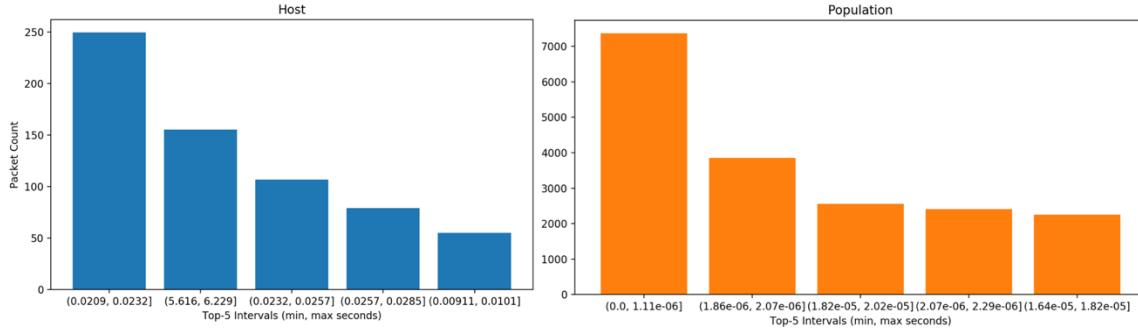


Figure 38. Host B top-5 interevent periods compared to the population

While not accounted for in the time-interval feature vectors, we are able to find similarities between Host B and population per-hour activity patterns. As already stated, the activity levels relative to volume are significantly different, but Host B and the population are both least active during hour 18 as shown in Figure 39.

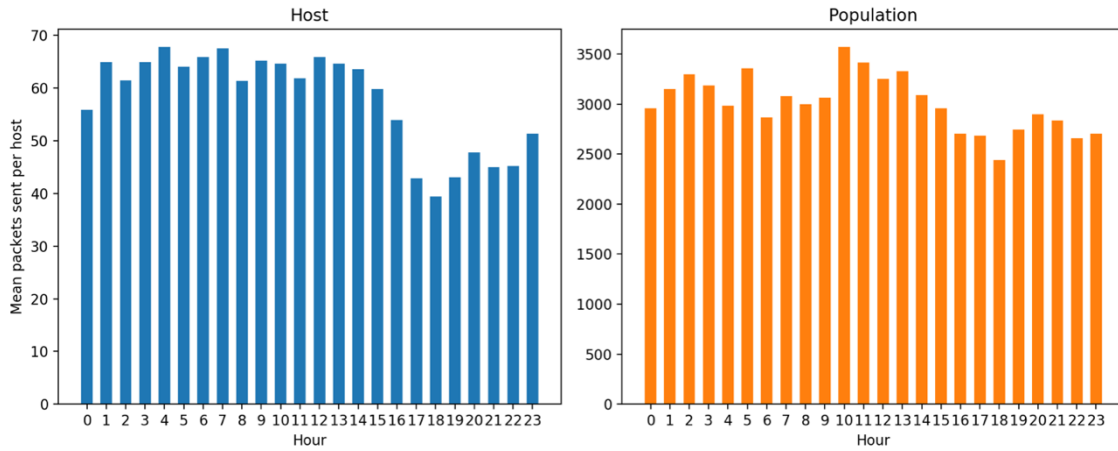


Figure 39. Host B's per-hour activity compared to the population

Despite some similarities in Host B's per-hour activity, it is clear that Host B demonstrated anomalous behavior relative to the population. And based on its mean silhouette score, it is also an outlier in respect to its cluster. However, in this case, Host B's identity as an energy management controller partly explains why it is behaving differently and why it was subsequently flagged by our system. Given its unique nature, we would

have to rely on a more direct comparisons with other subnet members or itself (host-relative) to determine if it is perhaps an insider threat.

Figure 40 shows Host B exhibits similar per-day activity patters across the observation period. Notably, Host B exhibits less activity than the average host from its subnet. The other notable exception is Host B’s behavior from the 8th to 9th. The subnets port utilization does not lend any insight to explain this exception—like Host B, subnet hosts also exclusively use other ports we did not enumerate.

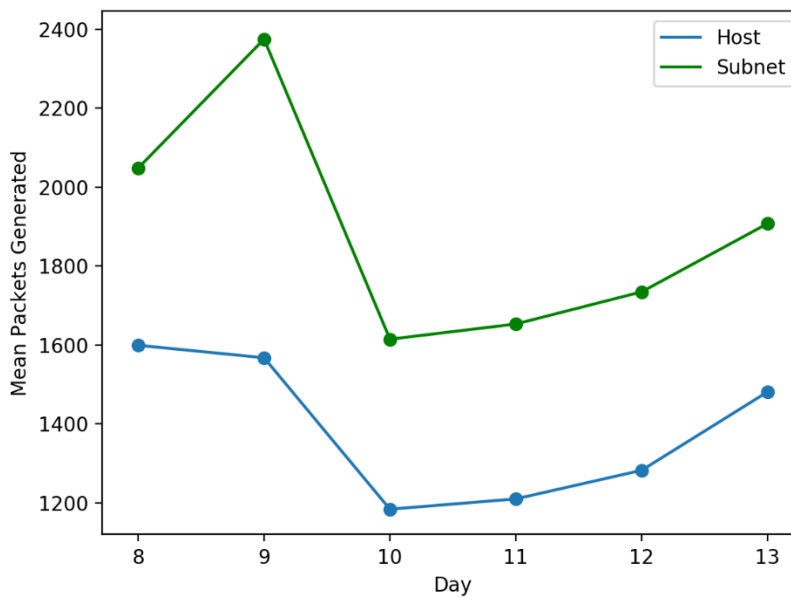


Figure 40. Host B’s volumetric activity patterns compared to its subnet

Figure 41 helps us understand the differences in behavior from the 8th to the 9th. Considering the otherwise low activity levels on the 9th, the subnet’s mean per-hour activity spikes during hours 10 and 14. While this could be cause to investigate other hosts in the subnet, it does not show Host B is behaving anonymously relative to its subnet.

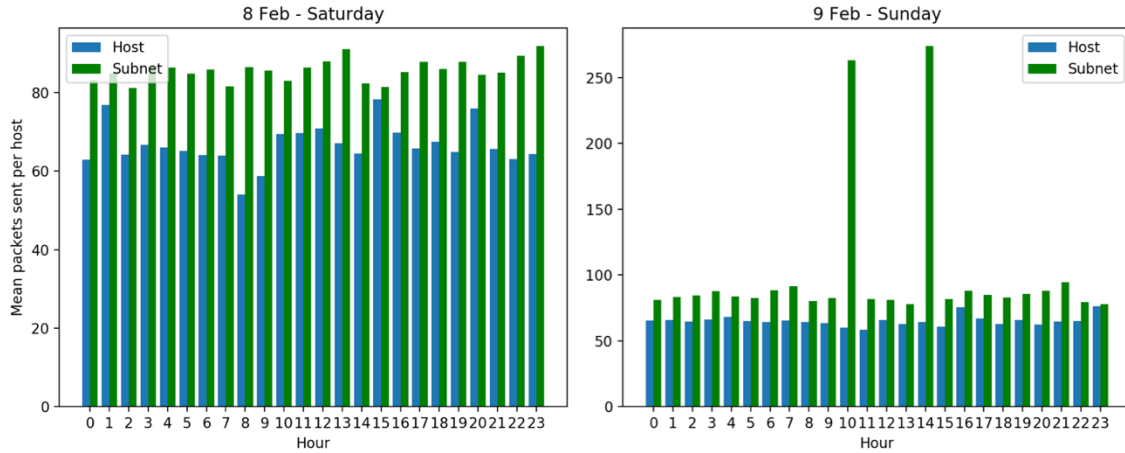


Figure 41. Host B's per-hour activity compared across days

Like Host A, we have shown that Host B's behavior is significantly different from the population of network hosts during the observation period. Comparing Host B's behavior with the behavior of other hosts in its cluster also explains its low mean silhouette score and why it was flagged by the system. Host B's unique identity on the network as an energy management controller explains its anomalous behavior relative to the population and its cluster. We identified several behavior differences between Host B and its subnet. First, Host B consistently demonstrates less per-day and per-hour activity; second, Host B communicates at different rates, notably with many more large packet time intervals. In the processes of examining Host B we found an anomalous spike in activity from members of Host B's subnet on the 9th—activity spikes on a day (Sunday) where we would expect less activity driven by humans on the network.

Host B is indeed behaving anonymously relative to the population and perhaps relative to its subnet. Without additional operational context and information, we cannot conclude Host B is a strong candidate to investigate further as a possible insider threat. Given Host B unique role in the network, comparisons with the population are likely less meaningful. A host-relative approach for identifying behavioral anomalies would likely be more effective for Host B.

3. Country Feature Vectors – Host C

Host C was flagged by the system for having a low Markov chain probability. As shown in Figure 42, Host C occupied 3 of the 5 total clusters and hopped 3 times. Only approximately 3% of the hosts we evaluated (not wireless and active for all 6 days) hopped 3 or more times, and 0.2% occupied 3 or more unique clusters. Host C also has a below average mean silhouette score. These measurements alone suggest that Host C is an outlier.

8 – 13 Feb Clusters	Hops	Unique	Days Active	Mean SS	Markov Prob.	Type
0 2 4 0 0 0	3	3	6	0.7995	1.07×10^{-5}	DHCP Wired
Transition Probs.	0.0343	0.0037	0.0909	0.9636	0.9636	

Figure 42. Host C cluster analysis

We immediately notice in Figure 43 that while Host C communicates more often with destination addresses in the United States, it sends a disproportionate amount of data to destination addresses in Canada: Host C sends an average of approximately 2,121 bytes per packet to destination addresses in Canada and it sends approximately 148 bytes per packet to addresses located in the United States. Table 6 compares bytes/packet ratios per destination country between Host C, the population, and Host C's subnet across the entire observation period (total bytes / total packets from 8–13 Feb). Host C sends significantly more data to destination addresses in Canada than the subnet and population mean.

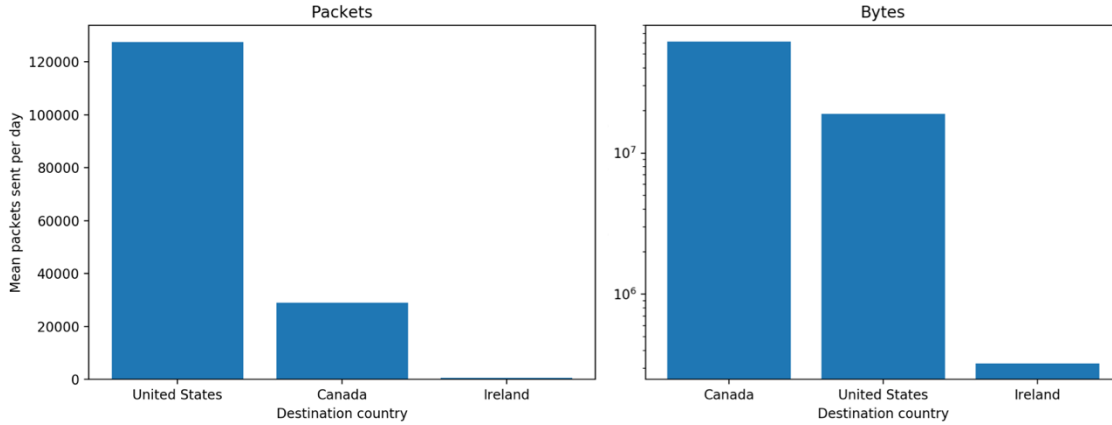


Figure 43. Host C destination countries

Table 6. Destination bytes/packet ratios

	United States	Canada	Ireland	UK	Hong Kong	Germany	Japan
Host C	148	2,120	537	824	632	43	602
Population	701	323	523	477	791	84	748
Subnet	793	93	690	567	619	44	587

Upon closer inspection, we noticed that Host C actually sends less data to destination addresses in Canada than the subnet and population mean on every day in the observation period except for the 10th and 13th. Figure 44 shows a clear spike on the 10th.

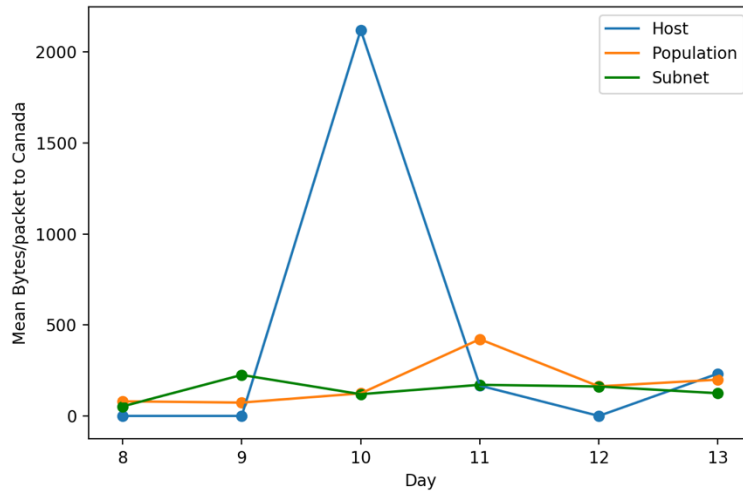


Figure 44. Host C bytes/packet ratio to Canada

While not as severe, Host C demonstrates anomalous activity concerning data sent to destination addresses in the United States. In Figure 45 we see Host C's mean bytes per packet to the United States fall well below population and subnet-relative norms.

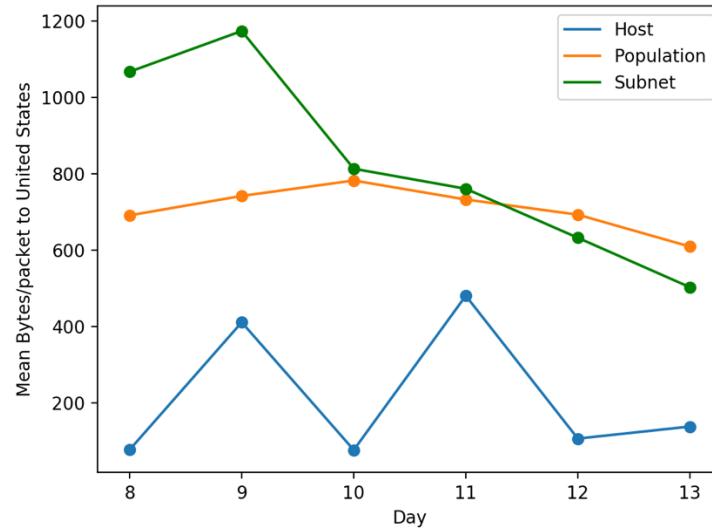


Figure 45. Host C byte/packet ratio to the United States

Figure 46 shows the total amount of data (bytes) Host C sends per day compared to the average host from both Host C's subnet and the population. The spike on the 10th in Figure 44 is clearly reflected in the overall total and surpasses average subnet and population hosts.

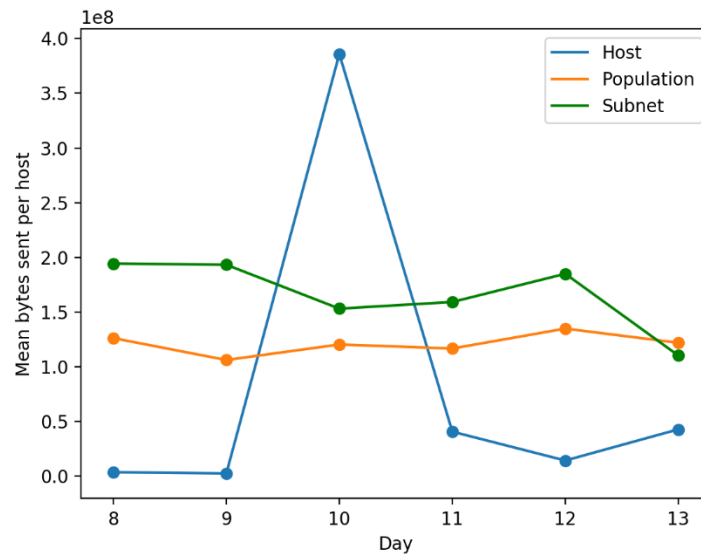


Figure 46. Mean bytes sent per day

This spike is likely the reason for Host C’s hop to cluster 4 from the 9th to the 10th (Figure 42). This transition is unlikely—0.37% probability. However, according to our cluster analysis, hosts in cluster 4 are foremost identified based on the amount of data they send to destination addresses in Canada. Despite only 0.39% hosts occupying cluster 4 on at least one day during the observation period, there is a noticeable network phenomenon that is causing 1 of only 5 clusters to be formed based on the amount of data sent to destination addresses in Canada. However, upon closer inspection of the other 27 hosts that occupy cluster 4 on at least one day, they do not appear to share this defining feature. Figure 47 shows the average host in cluster 4 does send more data to destination addresses in Canada on the 11th, but not nearly the amount Host C sends on the 10th. This possibly indicates that due to the similarity of the country feature vector hosts—as a result of the sparseness of the dataset—cluster 4 was formed to accommodate hosts that do not broadly share the common similarities of the other hosts in clusters 0-3. Again, this is likely a result of K-means forming a fixed number of clusters and grouping dissimilar samples.

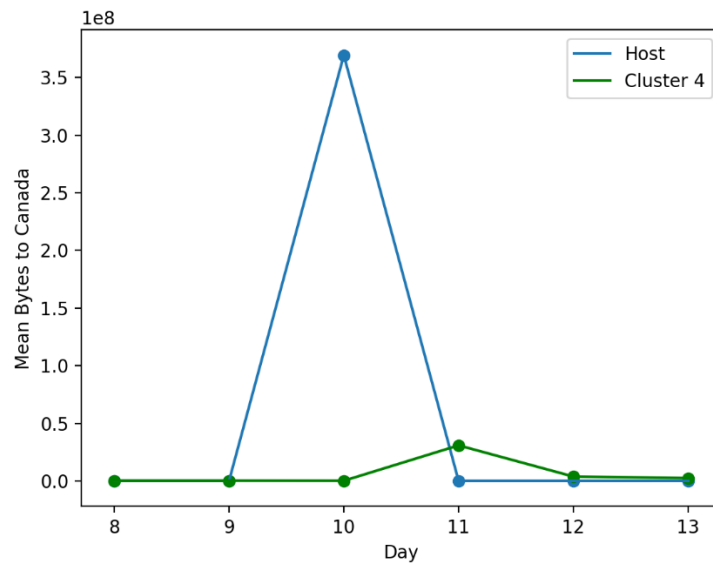


Figure 47. Host C bytes sent to Canada compared to cluster 4 hosts

It appears that Host C is behaving anonymously compared to the average population, subnet, and cluster 4 host. We uncovered this anonymous behavior by examining the amount of data sent to destination addresses in Canada during the

observation period. Interestingly, the country feature vectors (possibly due to their sparsity) were clustered in such a way that a small number of abnormal hosts were isolated in a small abnormal cluster (cluster 4). While our cluster analysis methodology indicates data sent to Canada is a defining characteristic of cluster 4, none of the 28 hosts — except Host C — seem to have this characteristic. This possibly indicates that the feature vector’s sparsity — in this case — is an advantage for anomaly detection. Due to the vast majority of hosts demonstrating similar behavior in terms of activity directed toward destinations in the United States, K-means appears to have isolated a few anomalous hosts that do not demonstrate this prevalent behavior.

C. DISCUSSION

In this chapter, we first provided an overview of how each dataset (time-of-day, time interval, and country feature vectors) was clustered. Second, we surveyed the population of network hosts in order to establish behavioral norms within the network. Third, we chose an outlier host for each feature vector type and conducted a case study on each to determine in what ways the host’s behavior diverged from the population and its subnet during the observation period.

As demonstrated in the case studies, each feature vector type described the hosts’ behavior in a different way (as designed), and as a result, we were able to recognize different behavioral anomalies based on each feature vector type. In the case studies, we found that outliers based on one feature type are also outliers based on the others, suggesting the feature vector types are complementary. Practically speaking, this means it is generally useful to apply the insights and clues all three feature types rather than just one or two. Having said this, under our current design each type provides different advantages and disadvantages as they relate to behavioral anomaly detection for the purpose of insider threat detection.

The time-of-day feature vectors were more comprehensive and easier to understand than the other feature vector types. As shown in Figure 23, the time-of-day feature vectors tolerated the selection of a higher k . Resultantly, we were able to segregate hosts into smaller, more defined groups. This allowed us to characterize host behavior (in an

understandable way) on a given day based on its cluster membership. Furthermore, engineering the features based on hourly activity allowed for interpretable results. Preliminary results suggest that further enumerating ports grouped in the “other” column would have increased the anomaly detection power of these feature vector types. Our results also indicate that the time-of-day feature vectors will likely miss host behavioral anomalies caused by automated processes that do not follow hourly routines (like humans do). As discussed in Chapter III, this was an intentional design and is further evidence that the feature vector types are complementary.

The time interval feature vectors were designed for the detection of device (automated) behavioral anomalies at a granular level. Our initial observations indicate that the time interval vectors could be immensely powerful for behavioral anomaly and insider threat detection if the right analytical framework is developed and applied. Such a framework should focus on exploiting the time interval feature’s discriminatory power and understanding the granular detail they offer for behavioral anomaly detection. A disadvantage to the time interval feature vectors is that without the right framework, they are difficult to interpret. Ideally in insider threat detection we want to be able to point out why a host is behaving anomalously, and anomalies based on time interval behavior are generally not as interpretable as time-of-day behavior. Given the mix of human and automated activity on many hosts (making it difficult to separate the two), we believe this feature vector type would be beneficial for host fingerprinting and the subsequent development of host-relative baselines, which this research largely did not address.

The country feature vectors were designed to characterize human-to-device behavior. As discussed in Host C’s case study, the skewed activity directed toward destinations in the United States surprisingly helped us identify a small group of misfit hosts who were behaving anomalously. However, this skewed distribution of activity would typically be undesirable for machine learning, and in our case did not allow widespread and fine-grained behavior characterization. This means while we were able to identify the most anomalous hosts, there were likely false negatives our system did not flag. On the other hand, one could argue this is desirable in an operational context—only flag the most anomalous hosts so human analysts are not overwhelmed with false positives.

A design improvement to how the system enumerates destinations (e.g., using destination cities or counties instead of countries) could have reduced sparseness and increased its detection effectiveness.

The case studies revealed an interesting relationship between the Markov chain probabilities and silhouette scores: the hosts flagged for having the lowest silhouette scores had very little cluster movement—most occupied 1 unique cluster and had 0 hops. Additionally, they usually were consistently clustered day-to-day on the periphery of a given cluster. This explains why (as was the case for Host B), the hosts flagged for their silhouette scores were unique relative to the population. This is favorable for behavioral anomaly detection. However, we suspect these hosts demonstrated consistent abnormal behavior because of their unique roles on the network, and not because they were insider threats. Fully enumerating the advantages and disadvantages of each metric would likely reveal further insights on their applicability to insider threat detection. Likewise, host-relative metrics could be used to evaluate these hosts for insider threat activity.

While mean silhouette scores appeared to be more effective in detecting repeated abnormal population-relative behavior, the Markov chain probabilities appeared to be effective in detecting behavioral anomalies as a result of dynamic or new emerging behavior. As seen with Host A and Host C, the Markov chain was useful in detecting changes in behavior that were different from the population (i.e., “spikes”). If we presume behavioral anomalies as a result of behavior change are indicative of insider threats, the Markov chain probabilities are likely more useful for this purpose.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION AND FUTURE WORK

This chapter summarizes the research conducted, draws conclusions from this work, identifies limitations, and discusses opportunities for future work.

A. SUMMARY

This thesis described a system that extracts host and organizational behavior from network traffic and detects behavioral anomalies using machine learning techniques for the purpose of insider threat detection. We developed and tested the system by using real-world network traffic collected on NPS's operational network over a contiguous 6-day period.

The system we developed extracts host and organizational behavior from TCP and UDP network traffic in the form of feature vectors using temporal measurements, volumetric measurements, and destination examination. We developed and engineered three different feature types; time-of-day features for describing human behavior, time interval features for describing device behavior, and destination country features which characterize human-to-device behavior. The system leverages the parallelism offered by the HSC at NPS to process several terabytes of real-world network traffic to create per-host feature vectors of each type.

The system's anomaly detection module uses the K-Means unsupervised machine learning algorithm to cluster network hosts and examine cluster membership over the contiguous time period. The system specifically accounts for hosts with statically assigned IP addresses and wired hosts with DHCP-leased IP addresses. Wireless hosts on the network are not accounted in our analysis. The anomaly detection module uses silhouette scores and a Markov chain to examine host conformity to network trends as defined by the population of active network hosts. The system flags the hosts with the lowest mean silhouette scores and Markov chain probabilities calculated over the observation period.

We selected three hosts flagged as anomalous by the system and conducted case studies to determine if indeed they behaved anomalously during the observation period. Furthermore, we examined the advantages and disadvantages of the three different types of feature vectors in order to determine which was better suited for behavioral anomaly

detection. Lastly, we examined the advantages and disadvantages of mean silhouette scores and Markov chain probabilities as population-relative metrics.

B. CONCLUSIONS

The system flagged the lowest scoring 1% of hosts based on their Markov chain probability and mean silhouette score for each feature vector type. An exhaustive approach to determine if all flagged hosts were indeed behaving anomalously would have resulted in over 200 individual case studies and ground truth data which was not available. Although we did not produce a complete confusion matrix to determine the system's effectiveness, all three case studies suggest that the subject hosts did behave anomalously during the observation period relative to the population. It is unknown whether this anomalous behavior was the result of benign or insider threat activity.

Given that each feature vector type accounts for different kinds of activity, we assess their use is complementary and their success is codependent for effective behavioral anomaly detection. However, we have pointed out areas for improvement: the system could be more effective if it enumerates more destination ports for the time-of-day or time interval feature vectors; and the country vectors are rather sparse and heavily skewed, resulting in inadequate host characterization. Furthermore, the power of the time interval feature vectors to detect behavioral anomalies based on device automation remains untapped without more robust analysis methods. Considering analytical comprehensiveness and ease of interpretation, we consider the time-of-day feature vectors to be the most effective host characterization method.

The case studies suggest the two population-relative metrics our system uses for measuring conformity are also complementary. We assess silhouette scores are best suited to detect static behavioral anomalies—hosts that consistently demonstrate population-relative anomalous behavior. Only 1 of the 30 hosts with the lowest mean silhouette scores occupied more than 1 cluster during the observation period. Host B consistently behaved anomalously relative to the population likely due to its unique identity and functionality in the energy management controller subnet. The Markov chain probabilities are best suited

for detecting emerging and perhaps longer-term behavioral anomalies which cause a host to move between clusters.

Host B serves as an example why silhouette scores and Markov chain probabilities are complementary for the purpose of insider threat detection. Consider the following hypothetical scenario. Host B behaves anomalously due to insider threat activity at some point during a 20-day observation period. Perhaps it hops clusters 1 time due to the insider threat activity. If our detector used Markov chain probabilities in isolation, its probability would likely be too high to be flagged by our system. In this case, it would be a false negative. If the system used silhouette scores in isolation, it would be flagged but it would likely be disregarded as a false positive given proper operational context. With operational context and a detector that uses both metrics in a complementary fashion, an analyst (or host-relative metric) would hopefully realize that a cluster hop resulting in a higher Markov chain probability is unlikely specifically for Host B.

C. FUTURE WORK

Our system has several limitations that, if implemented in an operational insider threat detection system, would have to be contended with. Fortunately, we believe these limitations can be practically and reasonably resolved.

1. System Efficiency

While we did not precisely measure system performance as part of this research, it is apparent our system's behavior extraction module is insufficient for keeping pace with the time requirements needed from an operational insider threat detector. In an operational context, insider threats must be urgently identified and addressed to minimize organizational damage. With the current resource and scheduling restrictions on the HSC, our system took approximately 12 hours to extract host behavior in the manner described in chapter IV from the input PCAP files. Several potential solutions exist to increase the efficiency of the behavior extraction module.

Remaining consistent with our current offline machine learning system design which extracts host behavior from batches of input PCAP files, one could attempt to

leverage a distributed storage and processing framework to speed-up the behavior extraction module. Apache's open-source Hadoop software is such a framework [44]. While more complex than our system design, an online machine learning model operating on real-time data streams would provide even greater performance increases. Apache Spark could be a computing solution for implementing a comparable system with an online machine learning capability [45].

An alternate methodology for increasing system efficiency and providing a better real-time insider threat detector could be to examine different network activity timeframes. As currently implemented, our system processes and analyzes 24 hours worth of PCAP files at a time. Even with our current non-distributed offline model, we would likely observe significant speed-ups from operating on 1 or 2 hours worth of PCAP files at a time. However, such a solution would have to account for changes in feature vectors and potential changes in detection performance.

2. Network Activity Timeframes

Examining smaller network activity timeframes offers another opportunity to achieve smaller detection timelines, potentially without sacrificing detection accuracy. In this research we examined 6 days of contiguous network activity. While machine learning models often perform better when presented with more training data, this should be verified and possibly sacrificed for the sake of smaller detection times. For example, while 6 days of training data may provide marginally better detection results, perhaps another system design using 2 or 3 days of training data provides comparable detection results with significantly smaller detection times. In essence, our research did not address how much training data is required to reliably detect behavioral anomalies and insider threats. Quantifying the tradeoff between the length of observation period and detection accuracy would be beneficial from an operational standpoint.

3. Population Activity Patterns

We examined how hosts behaved within the population of network hosts to detect behavioral anomalies. This research does not address how the population behaves relative

to the population on a recurrent basis. Examining population behavioral anomalies across a broad timeframe could provide useful and important insights for insider threat detection.

To address population-relative metrics, we used a Markov chain to examine the behavior of the population of network hosts on a given day, and how individual hosts behaved relative to the population. However, our methodology does not capture population behavior baselines or deviations from such a baseline. An understanding of how the population is expected to behave could help explain instances where hosts follow historical population activity patterns whereas the population reacts to new stimuli. We assess our current behavior extraction methodology lends itself to such an endeavor.

Markovian analysis could also help measure and document population behavior over an extended timeframe. Furthermore, our methodology for examining distinct clusters and their unique features could help one understand which clusters they expect will form on the network on a given day.

4. Host Activity Patterns

This research did not address host-relative behavioral anomalies. An insider threat detection system that considers population-relative and host-relative metrics could be powerful. For example, host-relative metrics could be used to build detailed host baselines which profile a hosts' hourly and interevent time patterns. As identified in this thesis, we foresee DHCP and data availability being the two primary challenges which would need to be addressed.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. VECTORIZE() COMPONENT

"""

VECTORIZE() COMPONENT

Written by: Brett Rajchel

Purpose: Create time-of-day, time interval and country feature vectors

Input: A single CSV or directory of CSV files in the following format

Epoch_time, source_ip, destination_ip, protocol, source_port, destination-port, length

1219005803,172.2.2.2,172.2.2.3,6,80,22,1500

Output: 3 CSV files. time-of-day, time interval, and country vectors

Notes: Can be used to process single CSVs in parallel on a HPC by calling `parallelize_gen_vectors(path)`.

"""

```
import geoip2.database
import pandas as pd
import numpy as np
import time
import os
import sys
from multiprocessing import Process
```

Constants

```
PORT_DIST = 0.6
HPC_READ_PATH = str(sys.argv[1])
HPC_WRITE_PATH = str(sys.argv[2])
SLURM_I = int(sys.argv[3])
```

Classes

```
class smart_dict(dict):
    def __missing__(self, key):
        return key
```

```
#####
##### GENERAL/UTILITY FUNCTIONS #####
#####
```

```
def get_top_ports(df, percentage=PORT_DIST, return_port_list=False):
    ''' Returns a pd.DataFrame which excludes packets with dst_ports not in
    top x percent of all dst_ports. This should only be used with time-of-day and
    time-interval vector creation. if 'return_port_list' parameter is True, only
    returns a list of ports, and not the filtered pd.DataFrame. '''
```

```

dist = df['dst_port'].value_counts(normalize=True)

counter = 0
wl = set()
for port, value in dist.items():
    if counter < percentage:
        wl.add(port)
        counter += value
    else:
        break

if return_port_list:
    return wl
else:
    result_df = df.query('dst_port in @wl')
    return result_df

def filter_top_ports(df, top_ports_set):
    ''' Inputs are pd.DataFrame and a set of ports desired ports. Returns a
    filtered version of the input df with packets to ports in top_ports_set and
    packets to 'other' ports. It adds synthetic packets from src_ip "x.x.x.x" to
    ports in top_ports_set that do not appear as a dst_port in the input df. '''

    others_dict = smart_dict()
    others = set(df['dst_port']) - top_ports_set
    for other_port in others:
        others_dict[other_port] = 'other'
    df['dst_port'] = df['dst_port'].map(others_dict)

    missing_top_ports = top_ports_set - set(df['dst_port'])
    for port in missing_top_ports:
        df = df.append({'time': 0.0, 'src_ip': 'x.x.x.x', 'dst_ip': 'x.x.x.x',
        'proto': 6, 'src_port': 4, 'dst_port': port, 'len': 10}, ignore_index=True)

    return df

def filter_top_countries(df, top_countries_set):
    ''' Inputs are pd.DataFrame and a set of desired destination countries (as
    resolved by mmdb). Returns a filtered version of the input df with packets to
    countries in top_countries_set and packets to 'other' countries. It adds
    synthetic packets from src_ip "x.x.x.x" to ports in top_ports_set that do not
    appear as a dst_port in the input df. '''

    others_dict = smart_dict()
    others = set(df['dst_country']) - top_countries_set
    for other_country in others:
        others_dict[other_country] = 'other'
    df['dst_country'] = df['dst_country'].map(others_dict)

```

```

missing_top_countries = top_countries_set - set(df['dst_country'])
for country in missing_top_countries:
    df = df.append({'time': 0.0, 'src_ip': 'x.x.x.x', 'dst_ip': 'x.x.x.x',
'proto': 6, 'src_port': 4, 'dst_port': 4, 'dst_country': country, 'len': 10},
ignore_index=True)

return df

def gen_vectors_by_file(file_path, filter_bool=False, ports_path='',
countries_path=''):
    ''' Call this function to get time-of-day, time-interval, and country
vectors in csv format from a single CSV denoted by file_path. Setting
filter_bool to True will ensure vectors are created uniformly from pre-
defined sets of countries and ports. If filter_bool is set to True, a ports_path
and countries_path must be provided to .txt files containing the desired ports
and destination countries. This function can be called by user or
parallelize_gen_vectors. '''

    fname = os.path.basename(file_path)
    name = fname.split('.')[0]

    df = pd.read_csv(file_path)
    df = df.replace([np.inf, -np.inf], np.nan).dropna(axis=0)
    if not filter_bool:
        top_ports_df = get_top_ports(df, PORT_DIST)
    else:
        ports_set = read_txt_to_set(ports_path, True)
        countries_set = read_txt_to_set(countries_path, False)

    # time-of-day vectors
    if not filter_bool:
        tod = gen_tod_vectors(top_ports_df)
    else:
        tod = gen_tod_vectors(df, ports_set)
    dst_filename = HPC_WRITE_PATH + name + '_tod.csv'
    tod.to_csv(dst_filename)

    # time-interval vectors
    if not filter_bool:
        ti = gen_ti_vectors(top_ports_df)
    else:
        ti = gen_ti_vectors(df, ports_set)
    dst_filename = HPC_WRITE_PATH + name + '_ti.csv'
    ti.to_csv(dst_filename)

    # country vectors
    if not filter_bool:
        cv = gen_country_vectors(df)
    else:
        cv = gen_country_vectors(df, filter_set=countries_set)

```

```

dst_filename = HPC_WRITE_PATH + name + '_cv.csv'
cv.to_csv(dst_filename)

def parallelize_gen_vectors(path):
    ''' Used to generate ti, tod, and cv vectors in parallel from a set of
    CSVs, all in one directory. For use on Hamming HPC. '''

    file_list = os.listdir(path)
    os.chdir(path)

    ports_path =
    '/home/brett.rajchel/work/gitlab/thesis_code/feature_engineering/top_ports_12
    feb.txt'
    countries_path =
    '/home/brett.rajchel/work/gitlab/thesis_code/feature_engineering/top_countrie
    s_12feb.txt'
    gen_vectors_by_file(file_list[SLURM_I], True, ports_path, countries_path)

def read_txt_to_set(txt_file_name, need_int):
    ''' Reads a text file and returns set of items from the text file. With
    need_int parameter, user can define the type (int or str) or set members. '''

    return_set = set()
    with open(txt_file_name, 'r') as filehandle:
        filecontents = filehandle.readlines()
        for line in filecontents:
            # remove linebreak which is the last character of the string
            port = line[:-1]
            # add item to the set
            if need_int:
                return_set.add(int(port))
            else:
                return_set.add(port)
    return return_set

#####
##### TIME OF DAY #####
#####

def __bin_ports(df):
    ''' Helper function for gen_tod_vectors. Bins number of packets sent during
    from src_ip during 0-23 hours. '''
    temp = ((df['time'] / 3600) % 24).astype(int)
    temp2 = np.bincount(temp, minlength=24)
    return pd.Series(temp2)

def __rename_add_cols(df):
    ''' Helper function for gen_tod_vectors. Creates and names columns needed
    for final df. '''

```

```

port = '_' + str(df.iloc[0]['dst_port'])
df.columns = ['src_ip', 'dst_port'] + [str(i) + port for i in range(24)]

return df

def gen_tod_vectors(df, filter_set=None):
    ''' Called to generate time-of-day feature vectors. If a filter_set is
    provided, it will only create column headers for the ports in the filter_set.
    '''

    if filter_set is not None:
        df = filter_top_ports(df, filter_set)

    df = df.groupby(['src_ip', 'dst_port'], sort=False).apply(__bin_ports)
    df = df.reset_index() # get rid of MultiIndex so we can directly reference
columns
    df.columns = df.columns.astype(str) # make sure all column names are
strings

    df = df.groupby(['src_ip', 'dst_port'],
sort=False).apply(__rename_add_cols) # rename and add all nessecary columns

    df = df.set_index('src_ip') # remove src_ip column and make it an index
column
    df = df.drop(['dst_port'], axis=1) # drop dst_port columns
    df = df.fillna(0) # make sure all nans are 0 so we can add them accross
groups

    df = df.groupby('src_ip', sort=False).aggregate(np.sum) # combine rows
with same src_ip by adding them

    # remove dummy src_ip
    if filter_set is not None:
        df = df.query('src_ip != "x.x.x.x"')

    return df

#####
##### TIME INTERVAL #####
#####

def __gen_bins(start=0.000001, stop=900, num_bins=200):
    '''Returns a list of logarithmically spaced numbers between start and stop
    (inclusive)'''

    bins = np.logspace(np.log10(start), np.log10(stop), num=num_bins)
    bins = bins.tolist()
    bins.pop(0)
    bins.insert(0, 0)
    return bins

```

```

def __bin_times(df):
    ''' Calculates interevent times between packets and bins them into
    logarithmically separated bins. '''

    bins = __gen_bins()
    diff_lst = df['time'].sort_values(ascending=True).diff().tolist()
    diff_lst.pop(0)
    s = pd.cut(diff_lst, bins=bins).value_counts()
    return s

def __mult_columns(df):
    ''' Defines column names as bin ranges for interevent times. '''

    port = '_' + str(df.iloc[0]['dst_port'])
    new_cols = []
    cop = df.columns[2:]
    for col in cop:
        temp = col + port
        new_cols.append(temp)
    new_cols = ['src_ip'] + ['dst_port'] + new_cols
    df.columns = new_cols
    return df

def gen_ti_vectors(df, filter_set=None):
    ''' Main function to generate time-interval vectors. If a filter_set is
    provided, it will only create column headers for the ports in the filter_set.'''

    if filter_set is not None: #filter the df to only have the dst_port in
filter_set
        df = filter_top_ports(df, filter_set)

    df = df.groupby(['src_ip', 'dst_port'], sort=False).apply(__bin_times)
    df.columns = df.columns.astype(str)
    df = df.reset_index()

    df = df.groupby(['src_ip', 'dst_port'], sort=False).apply(__mult_columns)

    df = df.set_index('src_ip') # remove src_ip column and make it an index
column
    df = df.drop(['dst_port'], axis=1) # drop dst_port columns

    df = df.fillna(0) # make sure all nans are 0 so we can add them accross
groups

    df = df.groupby('src_ip', sort=False).aggregate(np.sum) # combine rows
with same src_ip by adding them

    # remove dummy src_ip
    if filter_set is not None:

```

```

df = df.query('src_ip != "x.x.x.x"')

return df

#####
##### COUNTRY #####
#####

def __resolve_ips(df, del_not_resolved, return_set):
    ''' This function returns the df with the dst_country column. Optional to
    remove df rows where dst_country could not be resolved with del_not_resolved
    set to True. Can also be used to return the set of dst_countries in the df if
    return_set is set to True. IMPORTANT - "reader" variable must include the
    path to a valid mmdb. '''

    hamming_path =
    '/home/brett.rajchel/work/gitlab/thesis_code/mmdb/GeoLite2-
    Country_20200225/GeoLite2-Country.mmdb'
    dgx_path = '/home/brett.rajchel/Documents/thesis_code/mmdb/GeoLite2-
    Country_20200225/GeoLite2-Country.mmdb'
    local_path = '/Users/brettrajchel/Documents/thesis_code/mmdb/GeoLite2-
    Country_20200225/GeoLite2-Country.mmdb'
    reader = geoip2.database.Reader(hamming_path)

    loc_dict = dict() # store ips which have already been looked up
    loc_dict[ip] = country
    num_not_resolved = 0 # number of times an ip can not be resolved
    not_resolved = set() # the ips which could not be resolved
    new_col = [] # this will become the 'dst_country' column

    for ip in df['dst_ip']:
        if ip in not_resolved: # if the ip is a previously seen ip that cannot
            be resolved
            num_not_resolved += 1
            new_col.append('not_resolved')
            continue

        if ip not in loc_dict: # if ip has NOT already been seen and recorded
            - we need to look it up
            try:
                response = reader.country(ip)
            except Exception as e: # first time seeing this ip and it CANNOT
                be resolved
                not_resolved.add(ip)
                num_not_resolved += 1
                new_col.append('not_resolved')
            else: # first time seeing this ip and it CAN be resolved
                area = response.country.name
                loc_dict[ip] = area # add it to the dict
                new_col.append(area)

```

```

        else: # we have seen this ip before - look it up in dict
            new_col.append(loc_dict[ip])

    if return_set:
        result = set(new_col)
        result.discard('not_resolved')
        return result

    df['dst_country'] = new_col
    # remove not 'not_resolved' rows (or not)
    if del_not_resolved:
        result_df = df[df.dst_country != 'not_resolved']
        return result_df
    else:
        return df

def __prepare(df, pkts):
    ''' Enables half of the DF to count pkts, and the other half to counts
    bytes. Names columns accordingly. '''
    df = df.reset_index()

    if pkts: # if processing number of pkts df
        df = df.pivot(index='src_ip', columns='dst_country', values='time')
        df.columns = [col + '_pkts' for col in df.columns]
    else: # if processing number of bytes df
        df = df.pivot(index='src_ip', columns='dst_country', values='len')
        df.columns = [col + '_bytes' for col in df.columns]

    df = df.reset_index()
    df = df.fillna(0)

    return df

def gen_country_vectors(df, del_not_resolved=True, return_set=False,
filter_set=None):
    ''' The function generates country vectors. If a filter_set is provided,
    it will only create column headers for the countries in the filter_set.'''

    df = __resolve_ips(df, del_not_resolved, return_set)

    if filter_set is not None: #filter the df to only have the dst_countries
in filter_set
        df = filter_top_countries(df, filter_set)

    grouped = df.groupby(['src_ip', 'dst_country'], sort=False)

    pkts = grouped.count()
    vol = grouped.sum()

    pkts = __prepare(pkts, True)

```

```

vol = __prepare(vol, False)

df = pkts.merge(vol, on='src_ip')

# remove dummy src_ip
if filter_set is not None:
    df = df.query('src_ip != "x.x.x.x"')

df = df.set_index('src_ip')

return df

#####
##### MAIN #####
#####

if __name__ == "__main__":
    parallelize_gen_vectors(HPC_READ_PATH)

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: CLUSTERING() COMPONENT

"""

CLUSTERING() COMPONENT

Written by: Brett Rajchel

Purpose: Run the clustering pipeline as described in "Machine Learning Techniques for Insider Threat Detection" thesis chapter IV

Input: A single CSV containing time-of-day, time interval, OR country feature vectors across a given timeframe. A "date" column must exist in the CSV if multiple more than one of the same source_ip's exist. vectorize.py does not account for the "date" column.

Output: A single CSV file for each input CSV of the following form:
source_ip, <day1>, <day2>, <dayN>, num_hops, num_clusters, days_active, mean_silhouette_score, address_type, markov_probability

"""

```
import pandas as pd
import numpy as np
import math
import pickle
import time
import sys
import os
import ipaddress
import matplotlib
import glob
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from itertools import product, permutations
from sklearn.preprocessing import normalize
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import DBSCAN
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import SelectKBest, chi2, f_classif, f_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from multiprocessing import Process
```

```
##### CONSTANTS #####
CONTIG_DAYS = ['8', '9', '10', '11', '12', '13']
MASTER_NPS_DATA_LIST =
['/home/brett.rajchel/work/new_data/master_cv_nps_labeled.csv',
'/home/brett.rajchel/work/new_data/master_tod_nps_labeled.csv',
'/home/brett.rajchel/work/new_data/master_ti_nps_labeled.csv']
HPC_VECTORS_WRITE_DIR = '/home/brett.rajchel/work_new/markov_data/'
DGX_VECTORS_WRITE_DIR = '/home/brett.rajchel/Documents/thesis_code/vectors/'
NUM_DAYS_DICT = {'1011': 2, '101112': 3, '10111213': 4, '8910111213': 6,
'891011121315': 7}
IDEAL_K_DICT =
{'/home/brett.rajchel/work/new_data/master_cv_nps_labeled.csv': 5,
'/home/brett.rajchel/work/new_data/master_tod_nps_labeled.csv': 65,
'/home/brett.rajchel/work/new_data/master_ti_nps_labeled.csv': 27}
```

```
#####
##### GENERAL FUNCTIONS #####
#####
```

```
def d_tree(train_data, train_labels, test_data, test_labels, ret_str=True):
    ''' Implements a basic decision tree. Takes as input training and test
    sets with their respective labels. If ret_str is set to True (default), returns
    results in the form of a format string for easy writing. Else, prints results
    to stdout. '''
```

```
    tclf = DecisionTreeClassifier()
    tmodel = tclf.fit(train_data, train_labels)
    tpred = tmodel.predict(test_data)

    taccuracy = round(accuracy_score(test_labels, tpred), 4)
    tprecision = round(precision_score(test_labels, tpred, average='macro'),
4)
    trecall = round(recall_score(test_labels, tpred, average='macro'), 4)
```

```
    # for .txt output
```

```
    if ret_str:
        acc_str = 'DTree accuracy = {}, '.format(taccuracy)
        prec_str = 'DTree precision = {}, '.format(tprecision)
        rec_str = 'DTree recall = {}'.format(treall)
```

```
    ret_str = acc_str + prec_str + rec_str
    return ret_str
```

```
    else: # print results to stdout
        print('DTree accuracy = %f' %taccuracy)
        print('DTree precision = %f' %tprecision)
        print('DTree recall = %f' %treall)
```

```
def univariate_fs(data, cluster_labels, k_num):
```

```
''' Univariate feature selection. Removes all but the k_num highest scoring
features. Returns the dataset with only the top k_num features and a set of
the top five features'''
```

```
old_cols = data.columns
```

```
sel = SelectKBest(chi2, k=k_num).fit(data, cluster_labels) #chi2 is one
way to rank features
```

```
new_data = data[data.columns[sel.get_support(indices=True)]]
```

```
test2 = zip(old_cols[sel.get_support()], sel.scores_[sel.get_support()])
```

```
ordered = sorted(test2, key=lambda x: x[1], reverse=True)
```

```
for m in ordered:
```

```
    ordered2 = [(l, int(c)) for l, c in ordered]
```

```
return (new_data, ordered2)
```

```
def encode_one_vs_all(cluster_labels, one):
```

```
''' Prepares dataset labels for one-vs-all classification analysis. The
"One" labels do not change. The "All" labels are changed to 99. Returns the
encoded labels. '''
```

```
encoded_labels = []
```

```
for label in cluster_labels:
```

```
    if label == one:
```

```
        encoded_labels.append(one)
```

```
    else:
```

```
        encoded_labels.append(99) # is dummy value for "all"
```

```
return encoded_labels
```

```
def normalize_df(df):
```

```
''' Simple function which returns normalized data in a pd.DataFrame. '''
```

```
cols = df.columns
```

```
ind = df.index
```

```
return pd.DataFrame(normalize(df), index=ind, columns=cols)
```

```
def markov_chain(df, day_list, num_clusters, typ, nans=False,
drop_existing=False):
```

```
''' Implements a simple Markov Chain. It calculates the probability of
each possible state transition, and calculates the product of all src_ip
transitions accross a given time period specified by day_list. Probabilities
are calculated using all DHCP wired hosts and applied to all hosts (static,
DHCP leased wired, DHCP leased wireless). Returns a pd.DataFrame with a new
probabilities column with src_ip's who were active for all days specified in
the day_list parameter. '''
```

```
fd = open(HPC_VECTORS_WRITE_DIR + 'markov_trans_{}.txt'.format(typ), 'w')
```

```
# error checking - day_list must be passed as list of strings
```

```

if not all(isinstance(item, str) for item in day_list):
    print('Days must be passed as a list of strings')
    sys.exit(1)

# used to encode cluster id's (states)
map_dict, state_str = __blank_dict(num_clusters)
num_days = len(day_list)

# filters only src_ip's active for all days in day_list
if not nans:
    df = df.query('days_active == @num_days')

wired_df = df.query('src_ip in @MM_SET')
wired_df = wired_df[day_list].fillna(-1)
wired_df = wired_df.astype(int)
wired_df = wired_df.replace(map_dict)
wired_num_rows = wired_df.shape[0]
wired_num_cols = wired_df.shape[1]
wired_list = wired_df.values.tolist()

full_df = df[day_list].fillna(-1)
full_df = full_df.astype(int)
full_df = full_df.replace(map_dict)
full_df_num_rows = full_df.shape[0]
full_df_num_cols = full_df.shape[1]
full_df_list = full_df.values.tolist()

# get all possible transitions and instantiate needed dicts
state_transitions = product(state_str, repeat=2)
state_transitions = [''.join(x) for x in state_transitions]
state_dict = dict((el, 0) for el in state_transitions) # all possible
transitions - {ab: 0, aa: 0}
total_dict = dict((el, 0) for el in state_str) # accounts for the number
of all transitions from a single state {a: 0, b: 0}

# get raw number of transitions
for row in range(wired_num_rows):
    for ele in range(wired_num_cols - 1):
        cur_next = wired_list[row][ele] + wired_list[row][ele + 1]
        cur = wired_list[row][ele]

        state_dict[cur_next] += 1 # {ab: 34, aa: 15}
        total_dict[cur] += 1 # {a: 100, b: 34}

## error checking - makes sure number total transitions is correct - checks
total_dict ##
true_num_transitions = (wired_num_cols - 1) * wired_num_rows
actual_num_transitions = sum(total_dict.values())
if true_num_transitions != actual_num_transitions:
    print('The number of transitions is incorrect - total_dict is wrong')

```

```

        print('Should have {}, but have found {}'.format(true_num_transitions,
actual_num_transitions))
        sys.exit(1)

    ## more error checking - checks state_dict ##
    test_val = sum(state_dict.values())
    if true_num_transitions != test_val:
        print('The number of transitions is incorrect - state_dict is wrong')
        print('Should have {}, but have found {}'.format(true_num_transitions,
test_val))
        sys.exit(1)

    # calculates percentages vice totals in state_dict
    for key, value in state_dict.items():
        first = key[:1]
        total = total_dict[first]
        try:
            state_dict[key] = value / total
        except ZeroDivisionError: # this will apply if no users were clustered
in a given cluster
            state_dict[key] = 0
            print('No evaluated users were clustered in {}'.format(first))
            fd.write('No    evaluated    users    were    clustered    in
{}\n'.format(first))

    # save transitions in the .txt
    for k, v in state_dict.items():
        num = ''
        for char in k:
            t = str(ord(char) - 97)
            num += t
        fd.write('{} -- {}\n'.format(num, v))

    ## more error checking - checks to makes sure the calculated percentages
are correct ##
    total_pct = len(state_str)
    actual_pct = sum(state_dict.values())
    diff = abs(total_pct - actual_pct)
    if diff > 0.1:
        print('The calculated percentages are wrong in the state_dict')
        print('Should have ~{}, but found {}'.format(total_pct, actual_pct))

    # calculates the product of all transitions over the given time period
given in day_list
    new_col = []
    for row2 in range(full_df_num_rows):
        score = []
        for ele2 in range(full_df_num_cols - 1):
            cur_next = full_df_list[row2][ele2] + full_df_list[row2][ele2 + 1]
            score.append(state_dict[cur_next])

```

```

        new_col.append(np.prod(score))

    if drop_existing: # used for recalculation
        df = df.drop(['9_prob', '10_prob', '11_prob', '12_prob', '13_prob',
'15_prob', 'mean_prob'], axis=1)

    df['markov_prob'] = new_col

    fd.close()
    return df

def __blank_dict(num_clusters):
    ''' Helper function for markov_chain(df, day_list, num_clusters,
nans=False, drop_existing=False) '''

    chars = 'abcdefghijklmnopqrstuvwxyz`~!@#$%^&*()-=_+[]|}{;:<?,<./'
    state_str = chars[:num_clusters]
    map_dict = dict()
    clusters = [i for i in range(num_clusters)]
    for i in range(len(clusters)):
        map_dict[i] = chars[i]
    return (map_dict, state_str)

def label_source():
    ''' Label source master_nps csvs. For use on the HPC. '''

    index = int(sys.argv[1])
    fd = MASTER_NPS_DATA_LIST[index]

    ### format write name ###
    bname = os.path.basename(fd)
    typ = bname.split('_')[1]
    no_extension = bname.split('.')[0]
    write_name = '{}_labeled.csv'.format(no_extension)
    write_path = HPC_VECTORS_WRITE_DIR + write_name

    data = pd.read_csv(fd)
    data = data.set_index(['src_ip', 'date'])

    # if it is the CV dataset, we need to normalize separately
    if typ == 'cv':
        ind = 0
        for col in data.columns:
            if col.split('_')[1] != 'bytes':
                ind += 1
            else:
                break
        pkts_df = data.iloc[:, :ind]
        bytes_df = data.iloc[:, ind:]

```

```

        norm_pkts = normalize_df(pkts_df)
        norm_bytes = normalize_df(bytes_df)
        norm_df = norm_pkts.join(norm_bytes)
    else:
        norm_df = normalize_df(data)

    k = IDEAL_K_DICT[fd]
    ret = kmeans_model(norm_df, k)
    mod = ret[0]
    data['cluster'] = mod.labels_

    data.to_csv(write_path)

#####
##### CLUSTER ANALYSIS #####
#####

def __analyze_cluster_movement(df, markov_mode, get_each_ss=False):

    df_trad = df[['src_ip', 'date', 'cluster']]
    df_trad = df_trad.groupby(['src_ip'], sort=False).apply(__basic_stats)
    df_trad = df_trad.reset_index(drop=True)

    if markov_mode:
        return df_trad

    else:
        if get_each_ss:
            df_new = df[['src_ip', 'date', 'ss']] # use this for __get_all_ss()
            df_new = df_new.groupby(['src_ip'],
sort=False).apply(__get_all_ss)
        else:
            df_new = df[['src_ip', 'ss']] # use this for __get_mean_ss()
            df_new = df_new.groupby(['src_ip'],
sort=False).apply(__get_mean_ss)
            df_new = df_new.rename(columns={0: 'mean_ss'}) # use this for
__get_mean_ss()
            df_new = df_new.reset_index() #use for __get_mean_ss()

        result = df_trad.merge(df_new, on=['src_ip'])
        return result

def __get_mean_ss(df):
    ''' Returns the mean ss for each src_ip calculated accross all days. '''

    mean_val = df['ss'].mean()
    return pd.Series(mean_val)

def __get_all_ss(df):

```

```

''' Private function called by analyze_cluster_movement (df, write_path).
'''

df = df.pivot(columns='date', values='ss')
df.columns = [str(col) + '_ss' for col in df.columns]
return df

def __basic_stats(df):
    ''' Private function called by analyze_cluster_movement(df, write_path).
    '''

    df = df.sort_values(by='date')
    diff = (df['cluster'].diff().dropna() != 0).sum()
    num_clusters = len(df['cluster'].unique())
    days_active = df['cluster'].count()

    df = df.pivot(index='src_ip', columns='date', values='cluster')
    df = df.reset_index()

    df['num_hops'] = diff
    df['num_clusters'] = num_clusters
    df['days_active'] = days_active
    return df

def get_cluster_stats(df):
    ''' Gets broad cluster stats from the following columns and prints them to
    stdout: "days_active," "num_clusters," and "num_hops." '''

    num_days = df['days_active'].value_counts(normalize=True).round(3)
    for days, counts in num_days.items():
        print('%f percent IPs were active %i days'%(counts, days))

    num = df['num_clusters'].value_counts(normalize=True).round(3)
    for num_clusters, counts in num.items():
        print('%f percent IPs occupied %i different clusters'%(counts,
num_clusters))

    num_hops = df['num_hops'].value_counts(normalize=True).round(3)
    for hops, counts in num_hops.items():
        print('%f percent IPs hopped clusters %i times'%(counts, hops))

#####
##### KMEANS #####
#####

def kmeans_model(data, num_clusters):
    ''' Helper function which implements basic K-Means with a constant
    random_state parameter for K-Means model. Returns the model and the cluster
    distribution. '''

```

```

model = KMeans(n_clusters=num_clusters, random_state=10).fit(data)

unique, counts = np.unique(model.labels_, return_counts=True)
test2 = dict(zip(unique, counts))
return (model, test2)

#####
##### PIPELINES #####
#####

def __min_ss(data, desired_score=0.19, min_clusters=5):
    ''' Helper function for the clustering_data_pipeline which finds the ideal
    number of clusters with a desired silhouette score of atleast desired_score.
    Returns the highest scoring k_num with its corresponding score. '''

    silhouette_avg = 0
    while silhouette_avg < desired_score:
        clusterer = KMeans(n_clusters=min_clusters, random_state=10)
        cluster_labels = clusterer.fit_predict(data)
        silhouette_avg = silhouette_score(data, cluster_labels)

        min_clusters += 1

    return (min_clusters, silhouette_avg)

def __top_ss(data, max_clusters, min_clusters=5):

    score_dict = dict()
    for n_clust in range(min_clusters, max_clusters + 1):
        clusterer = KMeans(n_clusters=n_clust, random_state=10)
        cluster_labels = clusterer.fit_predict(data)
        silhouette_avg = silhouette_score(data, cluster_labels)

        score_dict[n_clust] = silhouette_avg

    score_list = sorted(score_dict.items(), key=lambda x: x[1], reverse=True)
    return score_list[0]

def cluster_data_pipeline(master_csv_path, write_path, markov_knum=5,
days_list=ALL_DAYS, max_clusters=70, find_knum=None, label_orig=False):
    ''' Overarching function for the pipeline. '''

    file_name = os.path.basename(master_csv_path)
    file_no_extension = file_name.split('.')[0]
    day_string = ''.join(days_list)
    stat_fd = '{}_{}.out'.format(file_no_extension, day_string)

    print('Cluster pipeline - %s' %file_name)

```

```

raw_data = pd.read_csv(master_csv_path)
data = raw_data.query('date in @days_list')

data = data.set_index(['src_ip', 'date'])
data = data.drop(['cluster'], axis=1)

# if it is the CV dataset, we need to normalize separately
if file_no_extension.split('_')[1] == 'cv':
    ind = 0
    for col in data.columns:
        if col.split('_')[1] != 'bytes':
            ind += 1
        else:
            break
    pkts_df = data.iloc[:, :ind]
    bytes_df = data.iloc[:, ind:]

    norm_pkts = normalize_df(pkts_df)
    norm_bytes = normalize_df(bytes_df)
    norm_df = norm_pkts.join(norm_bytes)
else:
    norm_df = normalize_df(data)

if find_knum is None:
    # get the ideal number of clusters based on the ss
    num_score = __top_ss(norm_df, max_clusters) # returns a tuple
    (num_clusters, silhouette_score)
else:
    num_score = (find_knum, '<not evaluated - find_knum param was None>')
    num_clusters = num_score[0]

mod, clust_dist = kmeans_model(norm_df, num_clusters)
markov_mod, not_used = kmeans_model(norm_df, markov_knum) # kmeans model
for markov_knum

if label_orig:
    data['cluster'] = mod.labels_
    temp_write = DGX_DATA_DIR + file_no_extension + '_labeled.csv'
    data.to_csv(temp_write)

characterize_cluster(norm_df, stat_fd, num_score, mod, clust_dist)

# get average ss, stats
ss_df = create_clusters_df(norm_df, mod)
m_df = create_clusters_df(norm_df, markov_mod, markov_mode=True)

# get markov score
m_df = markov_chain(m_df, days_list, markov_knum,
file_no_extension.split('_')[1], nans=False, drop_existing=False)

```

```

    m_df.to_csv(HPC_VECTORS_WRITE_DIR + file_no_extension.split('_')[1] +
'_mm.csv', index=False)
    m_df = m_df[['src_ip', 'markov_prob']]

    result = ss_df.merge(m_df, how='left', on=['src_ip'])
    result.to_csv(write_path, index=False)

    secondary_analysis(result, stat_fd)

def create_clusters_df(df, mod, markov_mode=False):
    ''' Intended to read a finalized feature vector csv (e.g.
    "master_tod_nps.csv). Returns a df with the cluster id associated with each
    sample/src_ip, the date the sample/ src_ip appeared on, and a "_ss" column
    column for each day showing the samples silhouette score. The type column shows
    if the src_ip is static or leased. '''

    df['cluster'] = mod.labels_ # add the cluster labels to the df

    if not markov_mode:
        ss = silhouette_samples(df, mod.labels_) # get the sample silhouette
scores
        df['ss'] = ss # add silhouette scores to the df

    # trim the df
    df = df.reset_index()

    if not markov_mode:
        df = df[['src_ip', 'date', 'cluster', 'ss']]
    else:
        df = df[['src_ip', 'date', 'cluster']]

    result = __analyze_cluster_movement(df, markov_mode) # worker function

    if not markov_mode:
        # determine if the src_ip is dhcp leased wired or wireless, or static
        result['type'] = result.apply(lambda row: 'static' if row.src_ip in
ALL_STATIC else ('leased_wireless' if row.src_ip in ALL_NPS_WIRELESS else
'leased_wired'), axis=1)
    else:
        result = result.drop(['num_hops', 'num_clusters'], axis=1)

    result.columns = result.columns.astype(str)
    result.columns = [col.strip() for col in result.columns]
    return result

def characterize_cluster(norm_df, txt_name, num_clusters_score, model,
clust_dist):
    ''' characterize_cluster analyzes the the calculated clusters and writes
    results to a file specified by txt_name. This function is currently configured
    for use in the clustering pipeline. It writes the cluster distribution, and

```

the top 5 cluster features and corresponding cluster scores for each cluster. Using one-vs-all analysis, it uses a decision tree to judge the strength of the top 5 scoring cluster features. '''

```

fd = open(HPC_VECTORS_WRITE_DIR + txt_name, 'w')

# write silhouette results
best_num_clusters = num_clusters_score[0]
best_cluster_score = num_clusters_score[1]
fd.write('Num clusters = {}, score = {}\n'.format(best_num_clusters,
best_cluster_score))

fd.write('Cluster distribution = {}\n'.format(clust_dist))
fd.write('\n')

for i in range(best_num_clusters):
    fd.write('##### START of cluster {} stats #####\n'.format(i))
    zero_labels = encode_one_vs_all(model.labels_, i)
    fd.write('- Highest scoring features for cluster {}: \n'.format(i))

    data, top_5_scores = univariate_fs(norm_df, zero_labels, 5)
    fd.write('-- {}\n'.format(top_5_scores))

    train_data, test_data, train_labels, test_labels =
train_test_split(data, zero_labels, test_size=0.2, random_state=0)
    ret3 = d_tree(train_data, train_labels, test_data, test_labels)
    fd.write('- Decision tree metrics for cluster {}\n'.format(i))
    fd.write('-- {}\n'.format(ret3))
    fd.write('##### END of cluster {} stats #####\n'.format(i))
    fd.write('\n')

fd.close()

def secondary_analysis(cluster_data_df, txt_name):
    ''' Called by cluster_data_pipeline. This function appends additional stats
    to the characterize_cluster.txt file. The stats provide broad analysis of the
    computed dataset. Specifically, it examines type, num_hops, num_clusters, the
    sample silhouette scores, and the markov probabilities accross the entire
    dataset. It also provides preliminary clues about which hosts could be
    anomalous. '''

    fd = open(HPC_VECTORS_WRITE_DIR + txt_name, 'a')

    # examine cluster stats - the full dataset
    fd.write('### Full dataset (not active all days) - {}
###\n'.format(cluster_data_df.shape))
    num_days =
cluster_data_df['days_active'].value_counts(normalize=True).round(3)

```

```

    for days, counts in num_days.items():
        fd.write('{} percent IPs were active {} days\n'.format(counts, days))

    num_clusters = cluster_data_df['num_clusters'].value_counts(normalize=True).round(3)
    for num_clusters, counts2 in num_clusters.items():
        fd.write('{} percent IPs occupied {} different clusters\n'.format(counts2, num_clusters))

    num_hops = cluster_data_df['num_hops'].value_counts(normalize=True).round(3)
    for hops, counts3 in num_hops.items():
        fd.write('{} percent IPs hopped clusters {} times\n'.format(counts3, hops))
    fd.write('\n')

    cluster_data_df = cluster_data_df.set_index('src_ip')

    # examine wired, wireless, static
    typ = cluster_data_df['type'].value_counts().round(3)
    for type_type, counts4 in typ.items():
        fd.write('{} IPs were {}\n'.format(counts4, type_type))
    fd.write('\n')

    wireless = 'leased_wireless'
    cluster_data_df = cluster_data_df.query('type != @wireless')
    fd.write('### Without wireless IPs ###\n')
    fd.write('\n')

    # more stats
    mean_hops = cluster_data_df['num_hops'].mean()
    fd.write('The mean num_hops == {}\n'.format(mean_hops))
    mean_unique = cluster_data_df['num_clusters'].mean()
    fd.write('The mean number of unique occupied clusters == {} \n'.format(mean_unique))
    fd.write('\n')

    # examine silhouette scores
    ss = cluster_data_df['mean_ss']
    fd.write('Silhouette score analysis:\n')
    fd.write('- The median is {}\n'.format(ss.median()))
    fd.write('- The mean is {}\n'.format(ss.mean()))
    fd.write('- {} has the MIN score of {}\n'.format(ss.idxmin(), ss.min()))
    fd.write('- {} has the MAX score of {}\n'.format(ss.idxmax(), ss.max()))

    pct_1 = math.floor(len(ss) * 0.01)
    ss_low = ss.sort_values().iloc[:pct_1]
    fd.write('- The lowest 1 percent scores (IP, score):\n')
    for ip, value in ss_low.items():
        fd.write('-- {}, {}\n'.format(ip, value))

```

```

fd.write('\n')
ss_low_ips = set(ss_low.index)

# examine markov probs
mp = cluster_data_df['markov_prob']
fd.write('Markov probability analysis:\n')
fd.write('- The median is {}\n'.format(mp.median()))
fd.write('- The mean is {}\n'.format(mp.mean()))
fd.write('- {} has the MIN probability of {}\n'.format(mp.idxmin(),
mp.min()))
fd.write('- {} has the MAX probability of {}\n'.format(mp.idxmax(),
mp.max()))

pct_1 = math.floor(len(mp) * 0.01)
mp_low = mp.sort_values().iloc[:pct_1]
fd.write('- The lowest 1 percent probabilities (IP, score):\n')
for ip, prob in mp_low.items():
    fd.write('-- {}, {}\n'.format(ip, prob))
fd.write('\n')
mp_low_ips = set(mp_low.index)

ip_int = mp_low_ips & ss_low_ips
fd.write('Intersecting IPs between low silhouette scores and low markov
probs: {}\n'.format(ip_int))

fd.close()

def parallelize_type(day_list=CONTIG_DAYS, write_dir=HPC_VECTORS_WRITE_DIR,
spec_clust=False, cluster_labels=False):
    ''' This function calls the clustering pipeline for all master_nps feature
    vectors for one day range (default is ALL_DAYS). For use on the HPC to leverage
    parallelism. '''

    if write_dir[-1:] != '/':
        print('Not a correctly formatted write_dir')
        sys.exit(1)

    index = int(sys.argv[1])
    fd
    =
['/home/brett.rajchel/work/new_data/master_cv_nps_labeled.csv'][index]

    file_name = os.path.basename(fd)
    file_no_extension = file_name.split('.')[0]
    day_string = ''.join(day_list)
    new_file_name = '{}_{}.csv'.format(file_no_extension, day_string)
    write_path = write_dir + new_file_name

    if not spec_clust: # default

```

```

        cluster_data_pipeline(fd, write_path, markov_knum=5,
days_list=day_list, max_clusters=50, find_knum=None,
label_orig=cluster_labels)
    else: # not default
        knum = IDEAL_K_DICT[fd]
        cluster_data_pipeline(fd, write_path, markov_knum=5,
days_list=day_list, max_clusters=50, find_knum=knum,
label_orig=cluster_labels)

#####
##### MAIN #####
#####

if __name__ == "__main__":
    start_time = time.time()

    parallelize_type(day_list=CONTIG_DAYS, write_dir=HPC_VECTORS_WRITE_DIR,
spec_clust=True, cluster_labels=False)

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: TIME-OF-DAY CLUSTER ANALYSIS

Cluster	1_Feature	1_Score	2_Feature	2_Score	3_Feature	3_Score	4_Feature	4_Score	5_Feature	5_Score	Accuracy	Precision	Recall
0	3_other	5009	4_other	4839	2_other	4652	1_other	4411	9_other	3983	0.9863	0.9686	0.9741
1	20_443	8249	20_53	70	10_other	67	11_other	67	13_other	59	0.9933	0.9371	0.9333
2	2_443	7356	3_443	462	1_443	212	6_443	130	10_443	70	0.9986	0.9862	0.9404
3	4_445	2849	4_80	105	12_445	55	3_445	31	8_445	19	0.999	0.8845	0.9542
4	10_other	11702	17_443	68	16_443	65	20_443	60	19_443	60	0.9986	0.99	0.9811
5	13_80	13571	13_443	74	11_80	68	6_7680	22	1_7680	22	0.9996	0.9998	0.9545
6	15_443	8723	15_53	70	15_80	38	10_other	33	11_other	31	0.9953	0.9243	0.9142
7	16_443	8617	16_53	81	10_other	61	11_other	58	14_other	51	0.9905	0.9072	0.8977
8	17_80	10312	17_8530	18	17_443	15	18_80	15	17_8014	11	0.9988	0.9208	0.9441
9	11_other	10752	17_443	59	20_443	55	16_443	54	19_443	52	0.9986	0.9854	0.9801
10	5_22	9214	6_22	9148	3_22	9054	13_22	8948	7_22	8912	0.9995	0.99	0.9965
11	6_443	2208	7_443	2037	5_443	1874	9_443	1860	3_443	1807	0.9841	0.8488	0.873
12	18_443	2196	19_443	1903	17_443	131	20_443	78	10_other	40	0.9952	0.9418	0.9185
13	18_445	5395	19_445	200	17_445	75	20_445	22	21_445	10	0.9983	0.9024	0.9024
14	6_other	5225	20_443	22	21_443	20	19_443	20	22_443	20	0.999	0.9595	0.9747
15	13_other	9993	17_443	47	20_443	47	21_443	43	16_443	43	0.9995	0.9895	0.9963
16	15_443	3172	16_443	946	17_443	208	15_53	52	10_other	32	0.9956	0.9279	0.8927
17	17_53	1021	16_53	1005	18_53	805	21_53	802	23_53	695	0.9958	0.8885	0.8621
18	14_other	9723	17_443	55	20_443	55	16_443	51	21_443	50	0.9995	0.992	0.992
19	23_443	1827	0_443	905	22_443	828	21_443	475	20_443	228	0.9831	0.7975	0.8056
20	15_other	7228	20_443	33	19_443	31	21_443	30	17_443	30	0.9989	0.9657	0.982
21	12_other	7325	17_443	34	20_443	32	16_443	31	19_443	30	0.9986	0.978	0.97

Cluster	1_Feature	1_Score	2_Feature	2_Score	3_Feature	3_Score	4_Feature	4_Score	5_Feature	5_Score	Accuracy	Precision	Recall
22	21_443	7736	22_443	179	21_53	76	21_80	70	10_other	60	0.9945	0.953	0.9265
23	16_445	5465	15_445	4232	17_445	694	18_445	227	23_445	192	0.9939	0.9195	0.9059
24	22_445	5659	21_445	217	23_445	205	19_445	83	20_445	74	0.9966	0.9156	0.867
25	8_other	7339	20_443	31	17_443	30	16_443	29	19_443	28	0.9996	0.9934	0.9871
26	19_80	9940	19_8014	34	19_443	23	18_80	20	0_80	8	0.9995	0.9347	0.976
27	7_other	6403	6_other	28	20_443	28	19_443	25	21_443	25	0.9996	0.9826	0.9998
28	0_443	7910	23_443	276	1_443	76	0_53	52	10_other	43	0.9958	0.9314	0.9514
29	14_53	2407	8_53	2372	9_53	2362	6_53	2356	5_53	2299	0.9937	0.8798	0.8896
30	21_other	2202	22_other	129	20_other	67	19_other	30	23_other	25	0.9976	0.8368	0.9134
31	17_443	9118	17_53	121	10_other	70	11_other	70	14_other	61	0.9921	0.9221	0.9451
32	13_445	3368	9_445	3268	10_445	3074	14_445	3005	12_445	2913	0.9898	0.8844	0.9083
33	22_443	7782	21_443	138	22_53	64	10_other	62	11_other	61	0.9944	0.9278	0.9478
34	0_445	4787	23_445	2402	1_445	614	22_445	240	21_445	162	0.9946	0.8597	0.8961
35	7_7680	3714	6_7680	3651	3_7680	3608	5_7680	3462	4_7680	3205	0.9935	0.8577	0.8854
36	14_443	6746	14_80	554	15_443	32	13_443	19	11_other	11	0.9981	0.9162	0.9321
37	9_other	6733	20_443	29	17_443	28	16_443	27	19_443	26	0.9996	0.9931	0.9866
38	5_other	7046	6_other	34	20_443	33	17_443	31	19_443	30	0.9995	0.9856	0.9856
39	20_80	6152	21_80	3208	20_8530	48	19_80	16	21_8014	15	0.999	0.9631	0.9417
40	20_other	2067	19_other	71	9_8530	26	10_7680	25	21_other	20	0.9975	0.8841	0.9074
41	3_445	1884	2_445	1822	6_445	1790	5_445	1541	7_445	1393	0.9952	0.857	0.8668
42	12_80	470	10_80	442	0_8530	401	1_8530	398	5_80	383	0.9386	0.6307	0.5864
43	19_443	7368	19_53	104	20_443	89	10_other	56	11_other	55	0.9916	0.9147	0.8981
44	21_445	5402	20_445	192	22_445	191	23_445	50	19_445	46	0.9976	0.9128	0.8976
45	16_80	10327	16_8530	28	16_443	27	10_other	7	15_80	7	0.999	0.9568	0.9323
46	11_80	13501	10_7680	675	9_7680	526	8_7680	525	11_7680	269	0.9978	0.9378	0.9488

Cluster	1_Feature	1_Score	2_Feature	2_Score	3_Feature	3_Score	4_Feature	4_Score	5_Feature	5_Score	Accuracy	Precision	Recall
47	16_443	3338	17_443	1395	18_443	183	19_443	77	16_53	64	0.9891	0.8586	0.8771
48	8_443	4670	4_443	1388	12_443	1249	0_443	83	9_443	28	0.998	0.9403	0.9461
49	17_445	6760	18_445	126	16_445	104	19_445	76	20_445	39	0.9978	0.921	0.9052
50	6_9997	8338	5_9997	8100	9_9997	8065	7_9997	8013	3_9997	8003	0.9996	0.9687	0.9838
51	20_445	4910	19_445	3075	21_445	480	18_445	283	22_445	238	0.9952	0.898	0.913
52	2_8530	22418	7_7680	60	3_7680	54	20_8530	39	5_7680	38	0.9996	0.925	0.9998
53	20_443	1837	21_443	1713	22_443	537	19_443	284	18_443	193	0.9889	0.8627	0.8727
54	22_80	8163	23_80	79	21_80	50	22_443	14	11_other	5	0.9993	0.9046	0.9471
55	12_445	1126	11_445	1005	12_137	949	13_137	947	14_137	942	0.9887	0.8944	0.8765
56	23_443	7682	0_443	123	23_53	86	23_80	53	11_other	45	0.9957	0.944	0.9491
57	14_137	1263	9_137	1252	13_137	1228	12_137	1227	10_137	1206	0.9972	0.9195	0.8486
58	18_443	7467	18_80	125	18_53	103	10_other	49	11_other	49	0.9922	0.9113	0.8901
59	1_443	8907	0_443	184	2_443	109	9_443	85	5_443	79	0.9968	0.9445	0.9329
60	9_7680	3067	8_7680	2894	10_7680	2857	11_7680	2851	12_7680	2735	0.9963	0.8719	0.8658
61	8_389	1030	14_389	1029	7_389	997	12_389	940	10_389	938	0.996	0.8692	0.8381
62	23_other	460	19_other	430	17_other	414	0_other	381	16_other	366	0.9788	0.7695	0.7601
63	15_80	11092	14_80	23	15_443	22	15_8530	17	20_443	6	0.9999	0.9844	0.9999
64	13_443	4906	9_443	23	12_443	16	13_80	15	11_443	14	0.9984	0.9301	0.8971

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D: TIME INTERVAL CLUSTER ANALYSIS

Cluster	1_Feature (interval_port)	1 Score	2_Feature (interval_port)	2 Score	3_Feature (interval_port)	3 Score	4_Feature (interval_port)	4 Score	5_Feature (interval_port)	5 Score	Accuracy	Precision	Recall
0	(2.02e-05 - 2.24e-05]_443	1607	(2.48e-05 - 2.75e-05]_443	1517	(2.24e-05 - 2.48e-05]_443	1373	(1.82e-05 - 2.02e-05]_443	1248	(2.75e-05 - 3.05e-05]_443	1142	0.9922	0.9131	0.9192
1	(0.000219 - 0.000243]_other	14748	(0.000197 - 0.000219]_other	8828	(0.00141 - 0.00157]_other	4597	(0.00127 - 0.00141]_other	4390	(0.00157 - 0.00174]_other	3740	0.9962	0.9837	0.9868
2	(5.69e-05 - 6.31e-05]_445	3390	(8.61e-05 - 9.55e-05]_445	1923	(9.55e-05 - 0.000106]_445	1922	(0.000117 - 0.00013]_445	1174	(0.000106 - 0.000117]_445	1031	0.9941	0.8763	0.8394
3	(5.616 - 6.229]_other	23517	(1.82e-05 - 2.02e-05]_other	3684	(1.64e-05 - 1.82e-05]_other	2680	(2.02e-05 - 2.24e-05]_other	2212	(0.000501 - 0.000556]_other	1320	0.9994	0.9972	0.9959
4	(1.46 - 1.62]_137	10285	(0.000178 - 0.000197]_137	5393	(0.00016 - 0.000178]_137	2563	(0.00016 - 0.000178]_445	2100	(0.000178 - 0.000197]_445	1353	0.9853	0.9494	0.9359
5	(0.0 - 1.11e- 06]_80	10672	(1.86e-06 - 2.07e-06]_80	4043	(1.11e-06 - 1.23e-06]_80	2567	(2.07e-06 - 2.29e-06]_80	2528	(2.82e-06 - 3.13e-06]_80	1740	0.9908	0.9252	0.9298
6	(0.000117 - 0.00013]_443	1004	(0.000243 - 0.000269]_443	924	(0.00013 - 0.000144]_443	830	(0.000219 - 0.000243]_443	818	(0.000144 - 0.00016]_443	648	0.9794	0.8014	0.8144
7	(0.0389 - 0.0431]_22	24417	(1.64e-05 - 1.82e-05]_22	12846	(1.82e-05 - 2.02e-05]_22	12676	(0.000501 - 0.000556]_22	11964	(2.02e-05 - 2.24e-05]_22	8198	0.9996	0.9913	0.9998
8	(5.69e-05 - 6.31e-05]_other	6492	(5.13e-05 - 5.69e-05]_other	4635	(6.31e-05 - 7e- 05]_other	3671	(4.62e-05 - 5.13e-05]_other	3207	(7e-05 - 7.76e- 05]_other	1744	0.9859	0.9214	0.9214
9	(0.000407 - 0.000452]_53	3389	(4.565 - 5.063]_53	2703	(0.000452 - 0.000501]_53	2581	(287.928 - 319.36]_53	1594	(0.00237 - 0.00263]_53	1435	0.9969	0.9597	0.9538
10	(0.0 - 1.11e- 06]_443	10045	(1.11e-06 - 1.23e-06]_443	2420	(1.86e-06 - 2.07e-06]_443	1789	(2.07e-06 - 2.29e-06]_443	1113	(2.82e-06 - 3.13e-06]_443	482	0.9854	0.9632	0.9657
11	(0.0209 - 0.0232]_443	1275	(0.035 - 0.0389]_443	1199	(0.0232 - 0.0257]_443	1178	(0.0653 - 0.0724]_443	1170	(0.0389 - 0.0431]_443	1100	0.946	0.8272	0.8268
12	(1.993 - 2.21]_other	2455	(26.569 - 29.469]_other	1382	(8.5 - 9.428]_other	1145	(2.21 - 2.452]_other	1137	(67.505 - 74.874]_other	1134	0.9845	0.8481	0.8646
13	(659.559 - 731.56]_other	1322	(92.114 - 102.17]_other	795	(0.0 - 1.11e- 06]_443	560	(287.928 - 319.36]_137	513	(1.82e-05 - 2.02e-05]_9997	415	0.9036	0.641	0.5898
14	(0.0 - 1.11e- 06]_7680	9029	(1.86e-06 - 2.07e-06]_7680	8175	(4.62e-05 - 5.13e-05]_7680	6912	(4.17e-05 - 4.62e-05]_7680	5888	(5.13e-05 - 5.69e-05]_7680	5542	0.9937	0.945	0.95
15	(1.64e-05 - 1.82e-05]_other	2965	(1.82e-05 - 2.02e-05]_other	2828	(0.0 - 1.11e- 06]_other	2297	(2.02e-05 - 2.24e-05]_other	2223	(1.48e-05 - 1.64e-05]_other	1945	0.9814	0.8954	0.8756
16	(54.871 - 60.861]_other	19942	(113.323 - 125.694]_other	271	(0.0 - 1.11e- 06]_443	45	(171.516 - 190.239]_other	31	(1.86e-06 - 2.07e-06]_443	28	0.9998	0.9899	0.9899

Cluster	1_Feature (interval_port)	1 Score	2_Feature (interval_port)	2 Score	3_Feature (interval_port)	3 Score	4_Feature (interval_port)	4 Score	5_Feature (interval_port)	5 Score	Accuracy	Precision	Recall
17	(0.000452 - 0.000501]_other	8159	(0.000407 - 0.000452]_other	2183	(60.861 - 67.505]_other	1277	(0.000501 - 0.000556]_other	236	(125.694 - 139.415]_other	192	0.9987	0.9865	0.9665
18	(0.0 - 1.11e- 06]_445	13735	(1.86e-06 - 2.07e-06]_445	4276	(1.11e-06 - 1.23e-06]_445	3316	(2.07e-06 - 2.29e-06]_445	2670	(2.82e-06 - 3.13e-06]_445	1308	0.9946	0.9613	0.9531
19	(0.965 - 1.07]_other	10690	(1.993 - 2.21]_other	122	(0.0 - 1.11e- 06]_443	55	(1.86e-06 - 2.07e-06]_443	32	(1.46 - 1.62]_137	21	0.9992	0.9795	0.9732
20	(0.000684 - 0.000758]_other	7980	(0.000616 - 0.000684]_other	28	(0.0 - 1.11e- 06]_443	22	(1.86e-06 - 2.07e-06]_443	13	(1.46 - 1.62]_137	9	1	1	1
21	(19.471 - 21.596]_other	9165	(0.000616 - 0.000684]_other	816	(0.000556 - 0.000616]_other	29	(40.212 - 44.602]_other	27	(0.0 - 1.11e- 06]_443	20	0.9996	0.9599	0.979
22	(1.82e-05 - 2.02e-05]_445	4642	(2.02e-05 - 2.24e-05]_445	3884	(1.64e-05 - 1.82e-05]_445	3711	(2.24e-05 - 2.48e-05]_445	2262	(0.000841 - 0.000933]_445	2236	0.9713	0.8636	0.8636
23	(1.86e-06 - 2.07e-06]_443	2940	(2.82e-06 - 3.13e-06]_443	2173	(2.07e-06 - 2.29e-06]_443	1827	(3.76e-05 - 4.17e-05]_443	960	(0.0 - 1.11e- 06]_443	932	0.9675	0.9107	0.9111
24	(0.0 - 1.11e- 06]_8530	17136	(1.11e-06 - 1.23e-06]_8530	4138	(1.86e-06 - 2.07e-06]_8530	3869	(2.07e-06 - 2.29e-06]_8530	2410	(2.82e-06 - 3.13e-06]_8530	1115	0.9972	0.9387	0.9432
25	(49.471 - 54.871]_other	11079	(2.719 - 3.016]_other	4393	(5.616 - 6.229]_other	1285	(3.016 - 3.345]_other	290	(54.871 - 60.861]_137	105	0.9995	0.9876	0.9651
26	(0.000616 - 0.000684]_other	15268	(0.0 - 1.11e- 06]_443	69	(1.86e-06 - 2.07e-06]_443	43	(1.46 - 1.62]_137	31	(2.07e-06 - 2.29e-06]_443	27	0.9995	0.985	0.985

APPENDIX E: COUNTRY CLUSTER ANALYSIS

Cluster	1_Feature	1_Score	2_Feature	2_Score	3_Feature	3_Score	4_Feature	4_Score	5_Feature	5_Score	Accuracy	Precision	Recall
0	United Kingdom_bytes	4324	United Kingdom_pkts	3615	Ireland_bytes	1626	Ireland_pkts	1458	Netherlands_bytes	230	0.9919	0.9629	0.9315
1	United Kingdom_pkts	13656	United Kingdom_bytes	9802	United States_bytes	92	United States_pkts	76	Netherlands_bytes	1.0	1.0	1.0	1.0
2	Ireland_bytes	2429	Ireland_pkts	2160	Netherlands_bytes	310	Japan_bytes	307	Japan_pkts	217	0.9961	0.9598	0.971
3	United Kingdom_bytes	5880	United Kingdom_pkts	2613	Ukraine_bytes	91	Ukraine_pkts	80	United States_bytes	16	0.9999	0.9884	0.9999
4	Canada_bytes	5016	Germany_pkts	4951	Germany_bytes	4885	Canada_pkts	4288	Mongolia_bytes	3035	0.9987	0.9439	0.7499

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] CERT National Insider Threat Center, “Common sense guide to mitigating insider threats, Sixth Edition,” Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-2018-TR-010, Dec. 2018. [Online]. Available: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=540644>.
- [2] Clearswift, “Cyber security incidents: Insider threat falls in UK (to 65%) and Germany (to 75%) post GDPR, but US risk increases (to 80%).” Accessed July 9, 20202. [Online]. Available: <https://www.clearswift.com/about-us/pr/press-releases/cybersecurity-incidents-insider-threat-falls-uk-and-germany-post-gdpr>
- [3] M. Nadeau, “State of cybercrime 2017: Security events decline, but not the impact,” CSO Online, July 28, 2017. [Online]. Available: <https://www.csoonline.com/article/3211491/state-of-cybercrime-2017-security-events-decline-but-not-the-impact.html>
- [4] D. Philipps, “Reality Winner, former N.S.A. translator, gets more than 5 years in leak of Russian hacking report,” *The New York Times*, Aug. 23, 2018.
- [5] Department of Defense, *The DOD Insider Threat Program*, DOD Directive 5205.16. September 30, 2014. [Online]. Available: <https://standards.globalspec.com/std/9873779/dodd-5205-16-d>
- [6] L. Liu, O. De Vel, Q.-L. Han, J. Zhang, and Y. Xiang, “Detecting and preventing cyber insider threats: A survey,” *IEEE Commun. Surv. Tutor.*, vol. 20, no. 2, pp. 1397–1417, 2018. [Online]. doi: 10.1109/COMST.2018.2800740
- [7] M. B. Salem, S. Hershkop, and S. J. Stolfo, “A survey of insider attack detection research,” in *Insider Attack and Cyber Security*, vol. 39, S. J. Stolfo, S. M. Bellovin, A. D. Keromytis, S. Hershkop, S. W. Smith, and S. Sinclair, Eds. Boston, MA: Springer USA, 2008, pp. 69–90.
- [8] R. F. Kelly, “The insider threat to cybersecurity: how group process and ignorance affect analyst accuracy and promptitude,” Ph.D. dissertation, Dept. of Info. Sci., NPS, Monterey, CA, USA, 2017. [Online]. Available: <http://hdl.handle.net/10945/56145>
- [9] S. Zeadally, B. Yu, D. H. Jeong, and L. Liang, “Detecting insider threats: Solutions and trends,” *Inf. Secur. J. Glob. Perspect.*, vol. 21, no. 4, pp. 183–192, 2012. [Online] doi: 10.1080/19393555.2011.654318.
- [10] D. E. Denning, “An Intrusion-Detection Model,” *IEEE Trans. Softw. Eng.*, vol. SE-13, no. 2, pp. 222–232, Feb. 1987. [Online]. doi: 10.1109/TSE.1987.232894.

- [11] 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*, Oakland, CA, USA, 2010. [Online] doi: 10.1109/SP.2010.25
- [12] W. T. Young, A. Memory, H. G. Goldberg, and T. E. Senator, “Detecting unknown insider threat scenarios,” in *2014 IEEE Security and Privacy Workshops*, 2014. [Online] doi: 10.1109/SPW.2014.42
- [13] D. Caputo, M. Maloof, and G. Stephens, “Detecting insider theft of trade secrets,” *IEEE Secur. Priv.*, vol. 7, no. 6, pp. 14–21, Nov. 2009. [Online] doi: 10.1109/MSP.2009.110
- [14] Defense Advanced Research Projects Agency, “Final report for the DARPA ADAMS project,” College Park, MD, USA, 2015. [Online]. Available: <https://apps.dtic.mil/dtic/tr/fulltext/u2/a625184.pdf>
- [15] P. A. Legg, O. Buckley, M. Goldsmith, and S. Creese, “Automated insider threat detection system using user and role-based profile assessment,” *IEEE Syst. J.*, vol. 11, no. 2, pp. 503–512, Jun. 2017. [Online]. doi: 10.1109/JSYST.2015.2438442
- [16] V. Sekar, M. K. Reiter, and H. Zhang, “Revisiting the case for a minimalist approach for network flow monitoring,” in *Proceedings of the 10th annual conference on Internet measurement - IMC '10*, Melbourne, Australia, 2010. [Online]. doi: 10.1145/1879141.1879186
- [17] B. Gonçalves and J. J. Ramasco, “Human dynamics revealed through web analytics,” *Phys. Rev. E*, vol. 78, no. 2, p. 026123, Aug. 2008. [Online]. doi: 10.1103/PhysRevE.78.026123
- [18] F. Radicchi, “Human activity in the web,” *Phys. Rev. E*, vol. 80, no. 2, p. 026118, Aug. 2009. [Online]. doi: 10.1103/PhysRevE.80.026118
- [19] V. Paxson, “Strategies for sound internet measurement,” in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement—IMC '04*, Taormina, Sicily, Italy, 2004. [Online]. doi: 10.1145/1028788.1028824
- [20] M. V. Mahoney and P. K. Chan, “PHAD: packet header anomaly detection for identifying hostile network traffic,” Dept. Com. Sci. FIT, Melbourne, FL, USA, Rep. CS-2001-04, 2001. [Online]. Available: <https://repository.lib.fit.edu/handle/11141/94>
- [21] A. Lakhina, M. Crovella, and C. Diot, “Mining anomalies using traffic feature distributions,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 217–228, 2005. [Online]. doi: 10.1145/1090191.1080118
- [22] Splunk, “UBA—User Behavior Analytics,” Accessed July 13, 2020. [Online]. Available: https://www.splunk.com/en_us/software/user-behavior-analytics.html

- [23] IBM, “IBM QRadar Security Intelligence Platform V7.3.3 documentation,” Accessed December 6, 2019. [Online]. Available: www.ibm.com/support/knowledgecenter/ss42vs_7.3.3/com.ibm.qradar.doc/qradar_ic_welcome.html
- [24] A. S. McGough *et al.*, “Insider threats: Identifying anomalous human behaviour in heterogeneous systems using beneficial intelligent software (Ben-ware),” in *Proceedings of the 7th ACM CCS International Workshop on Managing Insider Security Threats - MIST '15*, Denver, Colorado, USA, 2015. [Online]. doi: 10.1145/2808783.2808785
- [25] J. Glasser and B. Lindauer, “Bridging the gap: A pragmatic approach to generating insider threat data,” in *2013 IEEE Security and Privacy Workshops*, San Francisco, CA, May 2013. [Online]. doi: 10.1109/SPW.2013.37
- [26] D. C. Le and A. N. Zincir-Heywood, “Machine learning based insider threat modelling and detection,” in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Arlington, VA, USA, 2019, p. 6.
- [27] P. Parveen and B. Thuraisingham, “Unsupervised incremental sequence learning for insider threat detection,” in *2012 IEEE International Conference on Intelligence and Security Informatics*, 2012. [Online]. doi: 10.1109/ISI.2012.6284271
- [28] R. Pang, M. Allman, V. Paxson, and J. Lee, “The devil and packet trace anonymization,” *SIGCOMM Comput Commun Rev*, vol. 36, no. 1, pp. 29–38, Jan. 2006. [Online]. doi: 10.1145/1111322.1111330
- [29] A. Harilal, F. Toffalini, J. Castellanos, J. Guarnizo, I. Homoliak, and M. Ochoa, “TWOS: A dataset of malicious insider threat behavior based on a gamified competition,” in *Proceedings of the 2017 International Workshop on Managing Insider Security Threats*, New York, NY, USA, 2017. [Online]. doi: 10.1145/3139923.3139929
- [30] A. Azaria, A. Richardson, S. Kraus, and V. S. Subrahmanian, “Behavioral analysis of insider threat: A survey and bootstrapped prediction in imbalanced data,” *IEEE Trans. Comput. Soc. Syst.*, vol. 1, no. 2, pp. 135–155, Jun. 2014. [Online]. doi: 10.1109/TCSS.2014.2377811
- [31] M. Mayhew, M. Atighetchi, A. Adler, and R. Greenstadt, “Use of machine learning in big data analytics for insider threat detection,” in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, 2015. [Online]. doi: 10.1109/MILCOM.2015.7357562.
- [32] D. M. J. Tax and R. P. W. Duin, “Support vector data description,” *Mach. Learn.*, vol. 54, no. 1, pp. 45–66, Jan. 2004. [Online]. doi: 10.1023/B:MACH.0000008084.60811.49

- [33] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, “Deep learning for unsupervised insider threat detection in structured cybersecurity data streams,” in *Proceedings of AI for Cyber Security Workshop*, 2017. [Online]. Available: http://search.proquest.com/docview/2076555492?rfr_id=info%3Axri%2Fsid%3Aprimo
- [34] B. Böse, B. Avasarala, S. Tirthapura, Y.-Y. Chung, and D. Steiner, “Detecting insider threats using RADISH: A system for real-time anomaly detection in heterogeneous data streams,” *IEEE Syst. J.*, vol. 11, no. 2, pp. 471–482, Jun. 2017. [Online]. doi: 10.1109/JSYST.2016.2558507
- [35] Internet Assigned Numbers Authority, “Service name and transport protocol port number registry,” Accessed April, 7, 2020. [Online]. <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>
- [36] G. Gavai, K. Sricharan, D. Gunning, R. Rolleston, J. Hanley, and M. Singhal, “Detecting insider threat from enterprise social and online activity data,” in *Proceedings of the 7th ACM CCS International Workshop on Managing Insider Security Threats*, Denver, Colorado, USA, 2015. [Online]. doi: 10.1145/2808783.2808784
- [37] D. Arthur and S. Vassilvitskii, “K-means++: the advantages of careful seeding,” in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, New Orleans, Louisiana, 2007, pp. 1027–1035.
- [38] Google, “k-Means advantages and disadvantages | clustering in machine learning.” Accessed May 5, 2020. [Online]. Available: <https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages>
- [39] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in python,” *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: <https://scikit-learn.org/stable/index.html>
- [40] P. J. Rousseeuw, “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,” *J. Comput. Appl. Math.*, vol. 20, pp. 53–65, Nov. 1987. [Online]. doi: 10.1016/0377-0427(87)90125-7
- [41] SchedMD, “Slurm Workload Manager - Documentation.” Accessed April 16, 2020. [Online]. Available: <https://slurm.schedmd.com/>
- [42] Center for Applied Internet Analysis, “Analyzing UDP usage in Internet traffic.” Accessed April 13, 2020. [Online]. Available: <https://www.caida.org/research/traffic-analysis/tcpudpratio/index.xml>

- [43] MaxMind, “IP geolocation and online fraud prevention.” Accessed April 14, 2020. [Online]. Available: <https://www.maxmind.com/en/home>
- [44] The Apache Software Foundation, “Apache Hadoop.” Accessed July 14, 2020. [Online]. Available: <http://hadoop.apache.org/>
- [45] The Apache Software Foundation, “Apache Spark - unified analytics engine for big data.” Accessed July 14, 2020. [Online]. Available: <https://spark.apache.org/>

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California