



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**BIFROST: A STATISTICAL ANALYSIS FRAMEWORK
FOR DETECTING INSIDER THREAT ACTIVITIES ON
CYBER SYSTEMS**

by

Scott E. Findley

June 2019

Co-Advisors:

Alan B. Shaffer
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 June 2019	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE BIFROST: A STATISTICAL ANALYSIS FRAMEWORK FOR DETECTING INSIDER THREAT ACTIVITIES ON CYBER SYSTEMS			5. FUNDING NUMBERS	
6. AUTHOR(S) Scott E. Findley				
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) The purpose of this research is to investigate, design and implement a statistical analysis-based insider threat detection product deployable to resource-disadvantaged systems and provide organizations with a method for baselining the network profiles and host activities unique to their operational environments. Our system design seeks to alert the system and its operators to invest greater monitoring resources against hosts who exhibit threat characteristics of insider activity and prevent such activities from inflicting harm on the system and/or causing an information-loss event for the organization. This system provides an initial starting point for future work, implementing one means of detecting insider threat activities; this implementation results in best- and worst-case detection rates of ~74% and ~68.2%, respectively, against our test data. We believe our framework provides a reasonable starting point for future work and improvement.				
14. SUBJECT TERMS insider threat, baseline development, cyber-security, information security			15. NUMBER OF PAGES 107	
			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.

**BIFROST: A STATISTICAL ANALYSIS FRAMEWORK FOR DETECTING
INSIDER THREAT ACTIVITIES ON CYBER SYSTEMS**

Scott E. Findley
Lieutenant, United States Navy
BS, Georgia Southern University, 2012

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
June 2019**

Approved by: Alan B. Shaffer
Co-Advisor

Gurminder Singh
Co-Advisor

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The purpose of this research is to investigate, design and implement a statistical analysis-based insider threat detection product deployable to resource-disadvantaged systems and provide organizations with a method for baselining the network profiles and host activities unique to their operational environments. Our system design seeks to alert the system and its operators to invest greater monitoring resources against hosts who exhibit threat characteristics of insider activity and prevent such activities from inflicting harm on the system and/or causing an information-loss event for the organization. This system provides an initial starting point for future work, implementing one means of detecting insider threat activities; this implementation results in best- and worst-case detection rates of $\sim 74\%$ and $\sim 68.2\%$, respectively, against our test data. We believe our framework provides a reasonable starting point for future work and improvement.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	OBJECTIVE	1
B.	MOTIVATION	1
C.	CHAPTER OUTLINE.....	2
II.	BACKGROUND AND PREVIOUS WORK.....	3
A.	INTRODUCTION—THE INSIDER THREAT	3
B.	DEFINING INDICATORS AND DETECTING OUTLIERS.....	4
1.	Anomalous Network Activity.....	4
2.	Host-Based Indicators	8
3.	Baselining Operations.....	10
4.	Model Interoperability	10
C.	COMPOSITE SYSTEM DEVELOPMENT	11
1.	Identifying Design Shortcomings.....	12
2.	Synthesis.....	13
D.	SUMMARY	13
III.	BASELINE METHODS AND TARGET IDENTIFICATION	15
A.	INTRODUCTION—ESTABLISHING THE NEED FOR SYSTEM ACTIVITY BASELINES.....	15
B.	PLAN DESCRIPTION.....	16
1.	Profiling User and Network Data.....	16
2.	Generating Baselines	20
3.	Identifying Inherently Anomalous Behaviors	21
4.	Detecting Anomalies	22
5.	OCO Operational Means to Subvert System Operations	24
C.	PROPOSED SYSTEM	26
D.	SUMMARY	28
IV.	EXPERIMENT DESIGN AND INITIAL RESULTS	29
A.	INTRODUCTION.....	29
B.	EXPERIMENT DESIGN AND SETUP.....	29
C.	INITIAL RESULTS	32
D.	ANALYSIS	37
1.	System Detection Accuracy.....	37
2.	Baseline Generation Performance.....	38
3.	System Interface.....	38

4.	Observed System Shortcomings	39
E.	SUMMARY	39
V.	CONCLUSIONS AND RECOMMENDATIONS.....	41
A.	CONCLUSIONS	41
B.	FUTURE WORK	42
1.	Host Activity Granularity and Malicious Activity Interception	42
2.	Network Observation.....	43
3.	Administrative Interface	43
4.	System Efficiency	44
5.	OCO Considerations.....	44
	APPENDIX. HEIMDALL.....	47
	LIST OF REFERENCES.....	83
	INITIAL DISTRIBUTION LIST	89

LIST OF FIGURES

Figure 1.	Proposed BIFROST System	26
Figure 2.	NPS Glasgow Network Traffic	30
Figure 3.	NPS Glasgow Network Traffic Application Distribution.....	30
Figure 4.	NPS Glasgow Weekday Network Traffic Hourly Application Graph.....	31

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Anomaly Metrics and Sources	17
Table 2.	Detection Statistics for BIFROST Using Mixed Anomalous and Benign Data under Accurate Detection Assumption	34
Table 3.	Detection Statistics for BIFROST Using Known Benign Data	35
Table 4.	Theoretical Detection Statistics for BIFROST Using Mixed Anomalous and Benign Data under Extrapolated Detection Rate Assumptions.....	36

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ACL	Access Control List
BYOD	Bring Your Own Device
DCO	Defensive Cyber Operations
DoD	Department of Defense
DODIN	Department of Defense Information Network
DISA	Defense Information Systems Agency
ESS	Endpoint Security Solutions
HBSS	Host-Based Security System
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
IS	Information System
ITACS	Information Technology and Communication Services
NFS	Networked File System
NPS	Naval Postgraduate School
NTBA	Network Threat Behavior Analysis
OCO	Offensive Cyber Operations
POR	Program of Record
SDN	Software Defined Network
UBA	User Behavior Analytics
USG	United States Government

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

To my spouse, Paige: Thanks for putting up with the long hours and sleepless nights while I finished this up. While I wasn't home every night and don't communicate it well, your support and understanding were critical in this process, and I couldn't have done it without you.

To my friends at NPS: While I can't say you maintained my focus, thank you for helping me break away and not letting me sleep in the office.

To my advisors: Thank you for your support and insight during this challenge; this process wouldn't have been nearly as smooth or effective without your understanding and guidance.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. OBJECTIVE

Systems responsible for detecting and responding to information system (IS) threats, especially those serving the United States Department of Defense (DoD), must be constantly updated and improved to remain abreast of advances in offensive methods and strategies. Of great concern, however, are the insider threats, those that do not necessarily meet malicious discovery thresholds and instead blend in with the normal activities of their respective organizations. In this thesis, we attempt to identify potentially malicious insider activity by developing a baseline of organizational activity and using this information to develop threat anomaly alerts for insider activity. Our ultimate goal is to utilize these collected activity attributes to develop a novel detection system framework and provide the basis for validation of its implementation and effectiveness against threat data.

B. MOTIVATION

Significant effort and expense have been spent protecting information systems from external malicious threats, but relatively little has been done to evaluate the actions of legitimate users to prevent them from engaging in malicious or otherwise damaging activity. As seen in the recent DoD leaks by Edward Snowden, Bradley Manning, and Reality Winner, the insider threat possesses significant destructive potential against national security, international relations, and global commerce interests [1]–[3]. While external threat protection systems, like antivirus software, are established to be viable threat prevention means and are ubiquitous on modern information technology (IT) systems, detection and prevention systems focused on insider threats and are far less common [4].

Previous work to improve insider threat detection and prevention includes efforts to apply existing frameworks and techniques to malicious activity detection, but they are disparate in their methods of classifying and assigning threat potential to the various user activity indicators. These differing means of describing malicious indicators and responding to threats are also often not applicable to the DoD due to the restrictions

imposed by network authorization/validation processes and concerns unique to national security systems. We seek to alleviate this issue by producing a systems-agnostic solution capable of detecting insider activity within any organization's network without the need for integration or communications with any other external system.

C. CHAPTER OUTLINE

In this thesis, we discuss the background and existing models for insider threat detection, and the process for designing and implementing a detection system on commodity platforms to support disadvantaged networks. In Chapter II, we present the history and motivation for our study, to include insider threat definition and impact, suspect threat identifiers and previous efforts to span the insider threat detection gap in both academia and the commercial realms. Our design methodology, as shown in Chapter III, details our method of profiling system operations from available network and host data and the characteristics identified for inclusion in our detection system. In Chapter IV, we present our proof-of-concept detection system and discuss initial results against available threat data. Finally, in Chapter V we present our research conclusions and provide recommendations for future work.

II. BACKGROUND AND PREVIOUS WORK

A. INTRODUCTION—THE INSIDER THREAT

Insider threat activities are “an insider’s action(s) that puts at risk an organization’s data, processes or resources in a disruptive or unwelcome way” [5]. These actions can take the form of physical thefts, electronic exfiltration of data, the installation and control of malicious logic, or a host of other actions. The key component of any insider threat is the misuse of privilege or access by a trusted member of an organization [6].

Insider threats are critical threats against any business type, including the commercial sphere where 60% of fraudulent activities and 21% of all cyber-attacks against organizations are perpetrated by insiders [6]. More insidiously, the 2018 Insider Threat Report stated that 90% of the organizations represented in their study were vulnerable to insider activity due to the large number of unsecured/untracked mobile devices attached to their networks and the large body of users with privileges far above those required for their job functions [7]. These threats are not limited to commercial targets, as the actions of Edward Snowden, Bradley Manning and Reality Winner clearly showed [1]–[3]. With the significance of insider threat clearly established, it follows that attempts to categorize, detect and defeat insider threat activity has been researched and financed similarly to conventional efforts to disrupt the actions of external actors. As such, the DoD has established the “DoD Insider Threat Program” in an attempt to “develop and maintain an insider threat program to comply with the requirements and minimum standards to prevent, deter, detect, and mitigate the threat insiders may pose to DoD and U.S. Government (USG) installations, facilities, personnel, missions, or resources” [8].

However, as explained by Cole and Ring, many organizations do not clearly understand or consider inherently dangerous the threat posed by insiders, and that there are significant repercussions associated with identifying oneself as the victim of insider misconduct [4]. Additionally, the inherent nature of the insider threat creates great difficulty in identifying and isolating their actions from those of a benign, legitimate user. In fact, there is an increasing trend in unintentional insider threat activities, due to user

ignorance, poor training and adherence to policies, and the pervasive intrusion of Information Systems (IS) into previously unconnected or sensitive realms [9]. These difficulties create an extreme barrier in developing systems to detect and prevent insider activities.

Ultimately, anyone with trusted, privileged access to information within an organization is a vector for insider threat activity, whether intentional or not [6], [10]. As defined by the Carnegie Mellon University Computer Emergency Readiness Team (CMU-CERT), insiders can be intentional or unintentional; while the unintentional threat is not in malice, its action nonetheless “substantially increases the probability of ... serious harm to the ... organization’s resources or assets” [11]. With that fact established, when investigating the indicators and data associated with insider threat detection, operators must consider the behavior of users both in isolation and against the whole; with any member of an organization a potential threat vector, all activity is suspect. As opposed to the signature-based detection methodology used by many Intrusion Detection Systems (IDS), there are no true “signatures” for insider threats due to their unique status. This refers to the fact that insiders may appear benign in their activities, their typical activities are likely within the bounds of an employee’s expected normal behavior and their actions are differentiated only by the maliciousness of their intent or specific actions that traditional detection systems may disregard. To counter this target profile, any effective detection system for insider activity must employ a method of inductive analysis to establish a baseline for an organization’s inherent operational attributes, in order to identify disparate activities of their organizational populace and discover potential threats.

B. DEFINING INDICATORS AND DETECTING OUTLIERS

1. Anomalous Network Activity

There has been significant investment in exploring the network activity aspect of identifying and combatting the insider threat. With these efforts, the underlying assumption is that there exists some explicit indication of wrongdoing or improper activity. Some suspected indicators of insider activity include web traffic with questionable destinations, email gauges such as recipients and attachments, network traffic measurements and

network functionality interactions such as remote file system queries, inter-domain interactions, and administrative function use [4]–[6], [8]. Importantly, previous works have placed significant emphasis on identifying quantitative measurements of the monitored actions or activities themselves, including concepts such as volumetric and temporal characteristics of the activity [12]–[14]. These indicators, when taken together, serve as the basis for identifying outliers or anomalies within an organization, which can be used to signal malicious activities by an outside threat or by insider activities.

Maloof and Stephens, of Georgetown University and the MITRE Corporation, respectively, conducted research into detecting violations of user need-to-know and developed the ELICIT system as a means to monitor for and prevent insider misuse of resources for malicious activities, with specific emphasis on network activities [12]. Their patented method provided “methods, systems, and computer products for insider threat detection,” specifically to “detect network activity...to detect volumetric anomalies, suspicious and/or evasive behavior” [13]. The network attributes collected include the protocol in use, IP addresses, packet time and size metrics, and information about the user and resources in use [12]. Their method provides a central directory that stores and associates attributions for each user, comparing collected information against known patterns of misuse, as well as a user’s “past information-use activity”; these comparisons feed a probability distribution function that provides an alert when a user’s behavior falls below the existing threshold [13]. The system then processes this information through a Bayesian network to assign a threat score for each user, ranking all users on the network from most-to-least threatening [13].

The network sensors suggested by Maloof and Stephens “transform the packets [collected] into information-use events” by classifying events into eight discrete actions based on observed network activity: file listings, file reads, file writes, file deletes, sending emails, using search engines, printing, and moving files or directories [12]. In addition, their sensor network constructed social network maps from the information to provide greater granularity for use-case analysis; however, the sensors only monitor the local network; they do not collect information on packets traversing to/from the Internet due to privacy concerns [12]. The system attempts to assign each event to a user, but many events

(such as unauthenticated system shares) may not contain or require explicitly attributable information. For these unattributed events, the system uses a “nearest-neighbor method” based on active time, system/network locality and other weighted methods to link attributable events to the unattributed event based on surrounding actions [12], [13].

This collected information is compared against the existing body of data for both the individual user and the network as a whole to determine if it is anomalous against alert definitions for “volumetric anomalies, suspicious behavior, and evasive behavior”; this is determined using the times, locations and characteristics of the suspicious information-use event. These use-event alerts include blacklisted search terms, such as “proprietary” or “sensitive,” activities such as printing to non-local printers, and “anomalous browsing activity.” Each of these alert definitions feeds into the system’s Bayesian ranking network, which develop a risk score of the event and user based on the event’s severity/association with insider activity and how anomalous it is in context [12], [13].

In other work, the Splunk User Behavior Analytics (UBA) tool provides an integrated information ingest methodology when paired with the Splunk Core product [14]. UBA is designed to work with the Core as a primary ingest point, but can be configured to use a wide array of other sources, including host operating system logs, network traffic indicators and volumetric/temporal analysis [15]. Anomalous network activity, as compared to the baseline or current model, is used to develop alerts or adjust the observation model. This tool digests data in realtime, both testing it against existing baselines and developing/adjusting baselines; it operates with a 30/60/90-day comparison scope, enabling trends analysis beyond its current baseline. This method of updating baselines and models as a real-time function, i.e. maintaining running models beyond the “established” baseline and retaining previous versions for historic comparison, results in a very CPU-intensive product, limiting the product’s placement on network infrastructure nodes vice user devices. Additionally, UBA produces and maintains multiple models for each use-case, which seeks to increase the likelihood of detecting potentially malicious/anomalous behavior, however, this modeling method drastically increases the processing and memory requirements of the product [14].

The system proposed by Legg, Buckley, Goldsmith and Creese in 2015 represents “a synthetic approach for insider threat detection and analysis based on ... anomaly detection” by describing a method of defining normal operations using activity log data from an organization’s network [16]. The system examines historic “computer-based access logs, email and web records and physical building access records,” from which they generate a “daily observed profile” for each user [16]. Using this data, they create a series of “anomaly assessment scores” to identify groups of suspicious activities based on deviation from a user’s normal profile and that of others in their assigned job role [16]. Based on their previous work on a conceptual model of the insider threat, they used the elements discovered to define the targets of their system, a threat impact score for anomalous behavior, and a means to classify individual user threats based on deviation from the user’s own baseline and job role [6], [16]. This system provides a high level of granularity and continually updates its knowledge of user baseline activities, preventing a portion of false-positives over time due to the large body of data; however, its use of role-based data could potentially result in skewed results [16]. An alternative method of describing network data is suggested by Dean in his “Systematic Assessment of the Impact of User Roles on Network Flow Patterns” in 2018. In his work, he established that a user’s network activity can be better monitored for anomalous behavior by comparing it to all near-peers, regardless of job or user role, as roles and network activity have been shown to be relatively unrelated [17].

Shaghaghi et al. presented another model for managing risk created by the insider threat with their GARGOYLE system, in which they integrated their monitoring and response modules into a Software Defined Network (SDN) construct. Their methodology includes restricting data object and network access via functionally based access control rules, effectively locking resources against exposure and misuse [18]. Their system accomplishes this by dynamically inspecting and segregating network traffic based on membership to various access groups and current network/user activity. Their system accounts for the increasing propensity for bring-your-own-device (BYOD) policies by requiring all such devices to manage requests for network resources through an additional management level. This management entity then monitors the device’s location, activity

and user presence, restricting access to resources depending on the risk posed by the device's situation as defined by current physical or logical operating environment, the policies in place for that environment and requested resources [18]. This system seeks to be comprehensive in its coverage of network activity and file access requests, but does not maintain and utilize a robust means of detecting insider threat activity as a unique occurrence; rather, the system seeks to protect data and segregate user activity into approved zones instead of searching only for malicious activity. While it does gather data useful in identifying potential insider threats, the system uses these attributes to restrict access and enforce network/host policies to prevent misuse and requires an extensive repository of network/user rules for proper enforcement [18]. This system also requires integration into an existing SDN configuration and a potentially comprehensive re-engineering of network flow and user activity paths, possibly causing at least significant short-term disruption in operations.

2. Host-Based Indicators

Along with network-based indicators, research has also focused on host-based indicators for insider threat activities. Some system activities that could be identified as prospective threats include file system use, process executions, and system call activity. A study by Liu developed a method to track system calls specifically related to these indicators, which granted them the ability to monitor all activity conducted by the insider's host [19]. This effort used existing methods for detecting anomalies associated with external malicious threats and attempted to apply them to insider threats; overall, they determined that "feature representations ... for the external threat do not perform as well for insider [threats]" [19]. However, they did determine that some existing features from external threats had potential to be developed into representations of insider activity detectable via their system call method [19]. Building upon this work, Kanaskar developed a monitoring system based on employee "system usage profile[s]" defining "system usage patterns ... in terms of system calls" [20]. This method used Dynamical System Theory to develop a model of system user activities based on a deterministic set of discrete states. By studying the application environment of a user, their method develops an expected model of system activity based on the various applications and activities conducted by the user in

their normal operation [20]. This model uses an “approximate entropy” measure to evaluate the various state transition likelihoods between the user’s various application states across the system, allowing for development of expected or normal models of how a user or application should normally progress through states of operation, creating another means of anomalous behavior detection [20]. This system, while developmental, was successful in detecting “abnormal system dynamics” and indicating that anomalous behavior was occurring, which might integrate with insider threat detection systems; however, while useful for insider detection, this method is not restricted to the insider threat, as it theoretically can detect anomalous behavior regardless of source [20].

Columbia University researchers proposed a means of detecting insider threat host activities using a three-tiered system that deceives malicious actors and monitors potential insider activities at the network and host levels [21]. Their method relies on an extensive system architecture to develop, generate and monitor the use of various types of decoy documents, and integrate host-based sensors. Most interestingly, their host-based sensors perform user host activity baselining and work with network monitors to “combine multiple views of the same event” in an attempt to reduce false-positive rates [21]. At the host level, they developed a suite of host sensors known as “Are You You (RUU)” that profiles user activities to “isolate behavior differences over time,” providing a means to detect and compare anomalous behavior in host actions [21]. When a decoy document is activated, the system begins to report its use to the monitoring network, identifying the target host of interest, and logging information about the user and the interactions happening at the host level. Their deployment model relies on the idea that not all insiders pose the same risk as others, noting that based on the “levels of insider sophistication and capability,” different users will be more or less inclined to be fooled by the decoys, bypassing part of their detection system and acting outside the system’s purview [21]. However, their system is somewhat limited in its scope of detection in that only 10 seconds of user activity is used to form a baseline, and the system only monitors user search activity for anomalous behavior.

3. Baseline Operations

Whether for network or host-based monitoring, there is a clear requirement to properly baseline system activities; without a baseline for comparison, no anomalous detection method can identify activities that are outside the norm. For a composite detection system, the designer must integrate both network and host data to develop a usable target baseline for each user/host threat possibility, whether it be a low-level user, a system administrator or some other power user. For network baselining, a system must gather historic data on the network's activity overall and the specific attributes for each user. As with intrusion detection, these attributes must be properly categorized based on proximity to similar factors and classified into discrete labeled qualities [22]. Effective baselining must collect significant data from network traffic, including user roles, destinations, volumes, and temporal data points to “detect volumetric anomalies, suspicious and/or evasive behavior” [12], [16].

In the Bowen study, researchers described a system that profiles user activities based on host data from “file accesses, windows registry accesses, dynamic library loading, and window access events” to develop unique file system search patterns for each user that can be used in anomaly detection. They stated that “wider and deeper and in a less targeted manner,” search activity could be indicative of threat behavior [21]. In addition to search activity, the Bowen monitor collected data on various system attributes such as process creation patterns and timing; if the Dynamical System method used by Kanaskar were integrated with the data gathered by the Bowen monitor, a viable means of creating a baseline of system activity by system call analytics could be generated [20], [21].

4. Model Interoperability

Also important are the methods used to describe and define the various attributes and indicators of insider threat activity. Each detection methodology covered in the preceding sections described a unique means to measure and detect insider activity, but they do not share a common set of representations or measurement criteria. Therefore, the various models are not inherently interchangeable due to their individual methods of categorizing and representing the various data points used in their work.

As Magklaras and Furnell noted, there exist within the intrusion detection field clear lexicons for threat characteristics; these include the Common Intrusion Specification Language (CISL) and the Panoptis language [23]. These standards exist to provide a common means to classify and report on potential threats, creating frameworks for interoperability between IDS products and the operating systems and networks on which they reside; they “bridge the gap between language semantics and operating system/IDS product implementation” [23]. They define in their work the process of threat estimation which takes as inputs “Insider Threat Qualification Attributes” (ITQA), which are designer-specified user characteristics that delineate the characteristics of normal users across various detection levels and user types (network activity, host utility, process execution, group membership) that hold differing threat weights that feed their model. Using various weights and these inputs, their “Estimated Potential Threat” (EPT) function develops a quantifiable classification category based on likelihood of maliciousness. Their work on EPT also produced a matrix describing the various ITQAs, their weights and the potential meanings of the entries. They used this matrix to develop a clear language for insider threat detection activities, similar to those available within the IDS community. Their intent with the “Insider Threat Prediction Specification Language” (ITPSL) was to provide a means to standardize the various threat characteristics and be a link between different threat models and disparate detection vectors. Development of signatures using ITPSL definitions, which could then be used across various systems or activities, enables clearer integration of various methodologies and models for combatting the insider threat [23].

C. COMPOSITE SYSTEM DEVELOPMENT

While various methods and models for detecting and responding to insider threat activities exist in the academic, commercial and governmental realms, each has its own shortcomings and strengths. If the beneficial attributes of these disparate detection systems were combined into a single detection system, the resulting composite system could bridge their individual detection gaps and provide a more comprehensive means to monitor the potential indicators of insider threat activity.

1. Identifying Design Shortcomings

While many of the aforementioned detection methods provide extensive means for monitoring insider threat indicators, they often require considerable computing resources due to the extent of their monitoring. For example, the Splunk UBA product requires significant processing and memory resource allocation for its operation; this is due to UBA's maintenance of multiple threat models for each potential threat use-case and its real-time ingestion of network data while conducting comparisons and model manipulation using this data [14]. Similarly, GARGOYLE necessitates the use of SDN architecture, directing network flow and user activity paths across the monitored control plane infrastructure for monitoring and interaction; this potentially requires the complete redesign of existing infrastructure systems, significantly affecting operations [18].

Detection systems are inherently sensitive to target awareness of monitoring efforts; if the observed population knows the means by which an organization or monitoring system is tracking their activity, there exists the potential to subvert such efforts. Bowen acknowledges this when defining the various levels of sophistication within the insider threat continuum [21]. In their definitions, insiders vary from levels of low sophistication to highly privileged users; while low category users likely will not possess the levels of understanding or access to perceive the monitoring methodology in use, those on the other end of the spectrum likely will [21].

Furthermore, when dealing with anomaly detection systems, there must be adequate data for the creation of representative baselines. In the Bowen design, by limiting the baseline data to 10 seconds of monitoring after system initialization without clear historic analysis of past baselines, a sophisticated insider aware of these limitations could bypass the safeguards and proceed unimpeded [21]. In order to avoid these types of weaknesses regarding baseline data, an extended snapshot of activity must be gathered, but the line between too much and too little can be difficult to find.

An additional key shortcoming in many of these commercial and academic detection/prevention designs is that they require significant alteration to an organization's existing network and security design, and may require a complete redesign of how the

network manages its day-to-day operations regarding how resources are used. The DoD does not make such changes without a long review process comprised of rigorous testing and validations, potentially spanning several years of additional proving activities. With the sensitivity and criticality of these systems, it is understandable that the organization may be hesitant to adopt new technologies without extensive effort to ensure both effectiveness and security [24], [25]. As such, most of these designs are not viable for DoD systems; however, by augmenting the systems present on DoD assets to fill gaps in approved Programs of Record (POR), some of the most promising features from these various designs could be integrated into existing security architectures.

2. Synthesis

Although each previous methodology carries potential shortcomings and weaknesses, they all provide significant strengths that are prime for synthesis into a new product, which delivers a more comprehensive method of detecting insider threats while avoiding the design limitations of its parents. If taken individually as transitional tools or architectural design primitives, a developer using these separate design methodologies could effectively bridge the various gaps and synthesize a new method of adapting to and prosecuting the insider threat, which we will propose and describe in our later Chapters.

D. SUMMARY

This review introduced existing research regarding insider threat detection efforts, both in the commercial sphere and in academics. We described current literature concerning insider threat characteristics analysis and proposed detection schemes. This chapter further identified potential shortcomings and difficulties encountered with these various methodologies, including concerns with the current DoD security environment, both in developing an effective security system and overcoming the administrative hurdles for validation and program acceptance. For this work, we proposed integrating various existing detection methodologies to develop a new system for detecting and preventing insider threat activities, which we further explain and present in the following chapters.

THIS PAGE INTENTIONALLY LEFT BLANK

III. BASELINE METHODS AND TARGET IDENTIFICATION

A. INTRODUCTION—ESTABLISHING THE NEED FOR SYSTEM ACTIVITY BASELINES

As stated in Chapter II, the insider threat is a pervasive reality in both the commercial and governmental spheres. When considering its potential effects, one cannot ignore some of the most recent insider events in U.S. Government (USG) operations, specifically by members of the U.S. Intelligence Community including Edward Snowden, Bradley Manning, Reality Winner, and Harold Martin [1]–[3], [26]. The USG entrusted each of these individuals with incredibly sensitive information and accesses, which they used to inflict grave damage to both the international image of the United States and the operations of the United States Intelligence Community [1]–[3], [26]. While there are considerable oversight issues with these individuals, beyond systems access and use, each used their Information System (IS) access to find and steal information from their respective organizations [1]–[3], [26].

In the commercial sector, while there may not be individuals with the same household infamy, the impact of the insider threat is still significant [7]. The 2018 Insider Threat Report by Cybersecurity Insiders indicates that 90% of administrators consider their organizations susceptible to the insider threat, and 51% estimate that an insider breach would result in a greater-than \$100,000 loss [7]. The 2018 Ponemon report on insider threat costs found that the organizations in the study suffered between \$280,000 and \$650,000 in direct damage from these incidents, depending on the type of threat profile, defined as employee negligence (~\$280,000 average loss), criminal or malicious insider (~\$600,000 average loss) and credential theft (~\$650,000 average loss) [27]. This loss consists of direct monetary damage to the IS and intellectual property (IP) theft, but second-order damages caused by loss of brand prestige, contractual or liability concerns, and reduced trust from customers cannot be easily measured and would arguably be comparable in severity to the direct monetary loss [7], [9], [27]. Additionally, while the difference in motivation between a witting and unwitting insider is important, administrators should not separate their policies based on this distinction, as the unwitting insider's impact can be as great as or

greater than that of the witting, deliberate threat [6], [10]. By definition, “any person who has some legitimate privileged access to internal digital resources, [or] anyone who is allowed to see or change the organization’s computer settings, data, or programs in a way that arbitrary members of the public may not” must be considered a candidate for insider threat [28].

There is a need for consolidated detection systems triggered by anomalous or malicious user and network activity across all industry areas. Within such a system, we previously referred to two areas within an IS that have potential to provide insider threat alerts: host-based and network-based activity indicators, specifically those that prove anomalous to the established IS baselines. Network-based indicators refer to those network attributes of a network considered to have merit in developing a model of normal behavior, and that are tied to variance in a user’s normal operations; an example of this is network volume metrics, which provide insight to the activity levels of a user on the organization’s infrastructure [10], [12], [18]. Host-based indicators, as the name implies, are those related to a user’s actions and effects on his/her host system, relative to the operational requirements of their position; examples include application use, file system activity, and interaction with hardware resources [16], [19], [21]. Unfortunately, in order for these indicators to be useful in a detection system, there must first exist an effective and comprehensive IS-specific baseline for use in the anomalous characteristic comparison. This baseline is difficult to establish and validate across disparate organizational systems due to the significant differences and nuances present in different IS environments. However, we believe that implementing a generalized, systems-agnostic approach could successfully bridge this gap. To prove this concept, we suggest several elements below that we later integrate into our example detection system.

B. PLAN DESCRIPTION

1. Profiling User and Network Data

The core concept of our detection system is to develop dynamic profiles of the “normal” activities of an organization’s networks and users. Without these profiles, our system will have no activity baseline for comparison, preventing meaningful analysis of

emerging trends or potentially anomalous actions. Based on categorical distinctions between host and network-based data, we outline several metrics below that are effective for building these activity baselines, namely, network data metrics, host activity metrics, and user role analytics, as seen in Table 1.

Table 1. Anomaly Metrics and Sources

Metric Type	Measured Attributes	Example Metric	Metric Source
Network	Traffic time, volume, type and destination measurements	Average daily flow size for a target host to a given destination type	Collection nodes resident on network infrastructure
Host	Application use, system call activity and time measurements	List of applications commonly used on a target host	Collection nodes resident on each network host
User Role	Correlation or application use and network activity to user group norms	Average daily flow size for a host in a given user role group	Organizational user role assignment and K-Means++ cluster generation

Regarding network data, we considered the following attributes significant for this requirement: temporal characteristics, volume metrics, traffic destination inspection, and protocol analysis. Of note, considering the potentially critical resource requirements of monitoring the contents of each packet within an organization’s network, we believe that the use of a NetFlow-style record system will provide our design with the baseline utility and detail required to detect anomalous behavior while significantly reducing the processing and memory requirements [29]. We acknowledge that some fine-grained network details such as data content and specific external site details will be lost; however, this information is not directly attributed to our system at this time. Using this flow record method, we obtain patterns of life for the various hosts on the network, particularly related to work cycles and periods of peak activity. These temporal characteristics give direct insight to the normal operations of the organization’s network, providing an indicator

source for anomalous activity based on the previous observed active times for target hosts. Similarly, volume metrics are relatively simple to measure and account for with this flow method, and our system’s presence on infrastructure within the organization’s network makes access readily available. This volume metric is used to identify statistically significant anomalous behavior in terms of the volume of data movement within the network, which could be indicative of exfiltration behavior or the positioning of information for malicious use at a more opportune time [12].

Similar to our rationale with regard to volume metrics granularity, when inspecting destination characteristics of network activity, we are concerned with whether traffic is going to an internal or external destination and, if an internal destination, its category and sensitivity. As such, our system does not baseline traffic against external destinations by site, as the internal destination is more significant for monitoring resource use within the network; however, the system does check against an operator-defined “dirty-list” of external sites, providing direct indication of insider wrongdoing.

To obtain user activity metrics, we investigate the suitability of two attributes to feed our system’s baseline process: temporal characteristics and host activity analysis. Similar to network temporal characteristics above, examining the activity levels of a user over time has the potential to provide insights into a user’s “normal” behavior concerning both types of activities and general system use. An example of such a behavior abnormality is the case of a 9AM-5PM user conducting network activities late at night, potentially indicating an attempt to hide malicious activity; by including these time bounds on activity analysis, we characterize the user’s routine, adding a level of fidelity that may be lost by focusing solely on the activities themselves. Additionally, our method includes attributes of the user’s direct host-based actions across two main lines of effort: the use of network resources and host application use.

To monitor such host-based actions, our system receives and characterizes host system functions via inspection of the various system calls made on the target node; of particular interest are the calls to open/modify filesystem resources, search activities, networking operations, and process creation. These particular activities are notable due to their direct potential for misuse during an insider event, providing the insider access to

materials or information targeted for theft or manipulation and the capability to traverse the organization's network or initiate potentially malicious logic on their hardware [19]. This system call monitoring is accomplished through interaction with established auditing procedures on the target host, such as the Linux Audit system [30]. Reported events include the call made, the target of the call, and relevant arguments to specific calls. In order to limit the number of calls monitored, the specific system calls of concern are currently limited to file operations (open, read, write, chown, chmod, mkdir, etc.), network operations (socket, connect, send, receive, bind, listen), process manipulation (fork, execve, kill), and external hardware operations (printers, removable media, etc.). For example, if a user modifies a file, the write call and the targeted file would be reported to the system, along with the size of the write being made. These reports form each user's host activity profile, providing the system with the basis for comparing changes in a network operator's behavior, which gives insight into potential data manipulation or excessive/unbounded search activity.

Regarding user roles, Dean's 2017 study found that network activity comparisons between members with the same static user role (administrators, human resources, etc.) were not useful in discerning the user role identity of a given target profile [17]. As such, this type of user membership provides no benefit to profile generation in our system. However, by using a method similar to that described by Dean to create new "behavior-defined-user groups," we seek to operationalize that study's findings, specifically the use of K-means++ to "cluster ... feature vectors" and create representative user groups based on these similarities [17]. To accomplish this user group creation, we develop clusters of user data based on network time and volume data and the number of system calls per hour, generating distinct groupings [17]. These user groupings provide another statistical measurement for outlying data, similar to the Neighborhood Outlier Factor from the 2015 study by Jabez, giving an indication of the behavior's deviation from the user's established closest neighbor grouping [17], [31].

User roles do, however, help in determining whether the specific activities of a user are suspicious or inherently anomalous. Specifically, the types of applications used and internal destinations visited are indicators of group membership; for example, the use of

system administrator applications or commands by a standard user can indicate malicious activity if that user’s work role does not require such tools [32]. By comparing a user’s given work role to their application usage, the system can immediately flag activities that are questionable, and alert for administrative action. Given that each organization will have idiosyncratic requirements for their various user roles, these inherently anomalous behaviors must be configurable to match the organization’s needs throughout the system’s lifecycle.

Having described relevant classification criteria for both network and user data to form our target profiles, we develop representative baselines for use in our anomaly detection engine; by expanding beyond traditional categories and including additional data points, we believe that anomalous behavior will be captured more faithfully while minimalizing false results.

2. Generating Baselines

To generate useful baselines for the disparate data collected, we split the data into two bins: host-based data, and role-based data. Each host is designated its own baseline for network and host-based activities, providing a direct representation of that individual host for any data type collected by our system. We identify the data points within these two host baseline categories by day and hour, to order them chronologically. Lastly, we generate generic user activity groups, via the method used in Dean’s work to discern user group clusters with similar network activity using K-Means++ clustering, for use with collected user data to provide another comparison against expected behavior [17].

For these three baseline types (network activity, host-based activity, and cluster activity comparison) we construct an array of “neighborhood clusters” as described in Jabez’s 2015 study [31]. These clusters are generated by determining the “distance” between data points; for example, a neighborhood could be 10,000 packets whose sizes are within 1B of each other [31], [33]. Using these clusters, we discard outlying data, whose distance from the baseline’s clusters is greater than the distance between members of any other cluster. This data sanitization pre-processes the baselines for use, assuming that these outliers are results of irregular network conditions irrelevant to the overall model and may

otherwise introduce unintended results. Additionally, we find the data distribution's density bounds, indicating various values that fall within different quantile ranges of the distribution (i.e., those that encompass 90% of the distribution, then 50% of the distribution, and so forth) [34]. These findings form the primary foundation for our anomaly detection engine, as described later in this Chapter.

Generation of the baselines must occur against a body of normal system activity in order to provide an accurate representation of the target system [16], [21]. However, too large a period could result in anomalous activity being represented as “normal” in the operational baseline, reducing the system's ability to respond to threats. Conversely, too short a period will likely result in a greater false-positive rate, due to the lack of comparative information available to the system. We suggest that an initial baseline period of two weeks provides an ideal representation of network operations; we believe that the two-week baseline is ideal, as it provides multiple data points for each measurement and will help prevent potential false-positive results by smoothing the data ingested by the baseline. By starting with a two-week sample covering 24 hours of operations each day, the system begins with a complete representation of the organization's activities, assuming the observed weeks are considered “normal” for the organization. After generating this baseline, we update the data every week, weighting the existing baseline's values at 85%, as compared with 15% for any new data.

3. Identifying Inherently Anomalous Behaviors

Beyond system call activity level, there are several specific system calls that are inherently dangerous, and potentially indicative of unusual user activity. Among these are calls used for system manipulation or administrative actions, such as `sysctl`, `chown` or `mmap`, system operations monitoring such as `strace` or `ptrace`, and interaction with security modules such as `seccomp` or `selinux` modification. Without a valid reason for these various system requests, any of these requests could be indicative of a malicious user or potentially a system compromised by some other malicious entity. As such, these requests are innately suspicious and will not be considered “normal” activity for any user without intervention from a system administrator or membership in the administrator group [5], [6], [19].

In addition to the host-based malicious indicators, network indicators also exist that are inherently suspicious. For this system, we are concerned with indicators that correlate to malicious tool activity, such as DNS queries to external sources, IRC activity, etc., or use of dangerous protocols, such as SSH or telnet outside of established administrative norms, file transfers to external destinations, etc. To protect against these activities, we define a new threat set that includes these various actions and network indicators, and similar to the dangerous host activities, any time an instance of this threat set is present on the network, our system always considers such activity anomalous, barring the approval of a legitimate administrator [12], [20].

4. Detecting Anomalies

In 2014, the Bhuyan study stated that an effective network anomaly detection system must have a “model of normal behavior of the system and ... a proximity measure that allows one to compute, given an activity record, the degree of deviation that such activities have with regard to the model” [22]. As previously stated in this Chapter, in our system this proximity measure is a statistical analysis of data with respect to our established baselines (our model of “normal” behavior); statistically, anomalies are observations not directly related to the model and assumed potentially irrelevant to the model or data being investigated [35]. For our work, these anomalous data points provide the basis for detecting insider activities or other potentially dangerous activities.

With our assumption that activities related to insiders are inherently different from benign behavior, insider activities must appear as outliers to a “normal” behavior model [1], [2], [5], [6], [9]–[11]. Borrowing from Chhabra’s 2008 study, we rank network outliers on their deviation within modified “Generalized Quantile Sets (GQS)” of our traffic model’s data distribution [32]. These GQSs are multi-leveled areas “where the mass of [a distribution or model] is most concentrated,” with the level occupied by a data point indicating the level of anomalousness the data point possesses [32]. However, as Chhabra describes, one must account for the statistical “multiple testing problem” in that “observations in isolation” are not representative of behavior overall and should be adjusted to avoid high false-positive rates [32]. To avoid this issue, we create a threat

indicator, which is the average of a target's anomaly indicator value from its GQS level and the number of anomalies detected for the target. For this, we use an initial anomaly threshold value of GQS level 3 (representing 70% of the distribution) in our system to provide some latitude for variation in the target's activity levels before the system initiates a response. Our rationale for this threshold selection is that a host's activity will not be static over time, but if a host's activity varies more than 30% from their established norm, such deviation warrants additional scrutiny and consideration for observation. If a target's average activity is or becomes higher than our initial response threshold, a context anomaly conditional alert will occur and the target's threat indicator value is increased. Additionally, if a target's threat indicator ever exceeds our threat response threshold or its activity ever exceeds GQS level 4 (representing 60% of the distribution), a point anomaly conditional alert will occur, requiring system response and administrator intervention [22], [32].

Regarding host activity anomalies, a similar method as the one described for network detection is used. The number of system calls for a given day and time is compared with the established norms for the targeted user, and a comparison is made using the event's GQS-based anomaly indicator. Similarly as with network anomalies, if the average of a host's events is higher than our given GQS threshold of 2.5 (representing 75% of the distribution), a context anomaly conditional alert will occur; a point anomaly conditional alert will trigger if the indicator ever falls below GQS level 3 [22], [32]. We use a lower threshold for host activities due to the greater concern regarding significant variations in a user's direct actions; while network activity (specifically volume) may vary in the course of normal operations due to network conditions, we consider a user's normal host activities to be more static and thereby more sensitive to variations [36].

Similarly, the host's registered user's established user role and generated representative activity groups are used to further detect anomalous behavior. For each of these categories, a GQS is generated for the various indicators (volume, time, syscall, etc.), against which each event is further checked for anomalousness. If the GQS average of a host's event is higher than our given GQS threshold (GQS level 3 in this case), a context anomaly conditional alert will occur; a point anomaly conditional alert will trigger if the indicator ever falls below GQS level 4 [22], [32]. Additionally, as indicated previously,

our system must immediately respond to any indication of known malicious or dangerous activity. Any event matching an established signature of these events will trigger a point anomaly conditional alert.

5. OCO Operational Means to Subvert System Operations

Although we believe our system methodology can detect anomalous behavior in most cases, there exists real potential for developing countermeasures for use in Offensive Cyber Operations (OCO). These operations by necessity must seek to minimize exposure to reporting and alerting mechanisms, as their presence on a system can have significant and far-reaching impacts on military operations and diplomatic ventures; efforts to remain within the realm of a target's "normal" operations are by definition desirable, as they provide the greatest concealment [37]. However, an effective insider detection system could capture indications of OCO operations, potentially endangering their mission and leading to exposure; due to this potential threat of exposure, OCO operators must develop means to circumvent such systems. Specifically in our design, if the malicious actor were to intercept the features collected or study the system's data ingest and reporting process, there is potential for three primary vectors of circumvention: manipulation, deletion, and subversion/stealth.

In the case of manipulation, if an attacker knew of and could access messages between the infrastructure nodes and the central hub, they could intercept the messages related to the activities generated from their infected host and change the contents to appear less abnormal. By altering the time, volume or activity information, the attacker could prevent the system from ingesting data related to the attacker's malicious activity. While difficult to perform wholesale due to the positioning of consolidation nodes and requirement for extensive infection in the target system, such a method would provide the attacker virtually unimpeded freedom to operate if no other detection system were in place.

Similarly, given an intermediary position within the reporting network, the attacker could simply delete damning messages related to their activity, effectively removing any indication of their presence. The current framework of this system does not have a robust means of checking for deletions, instead relying on a simple acknowledgment from the

central server, which an attacker could subvert relatively simply. Without some form of metrics report check—an additional message digest comparison, for example—this subversive threat vector would be difficult to correct and rectify without a supplementary security system searching for such activities.

Additionally, if an actor were to study and learn the target network’s “normal” message traffic from the resident detection system, the attacker may change his activity to remain below the target threshold. For example, if an attacker inspected the proposed system’s cleartext update messages from host and network nodes, the attacker could discern the metrics being reported and avoid any activities that may appear anomalous or any that are specifically targeted by the system. While this method would require time to review system information and re-acclimate to the target space, it is the least intrusive of the subversion methods against such a system and likely would be the easiest to implement, if no other detection system or protections were in place.

The attacker could potentially use such system knowledge and subversion capability as an information warfare weapon, causing significant network congestion, detection system inefficiency, or organizational mistrust and dissent. By generating false reports or directly altering system configurations, they could cause the detection system to react to false positives and significantly skew baseline information across the network. This manipulation would severely affect the system’s reliability and functionality, and would introduce a great deal of additional resource load on the system beyond its normal operating bounds. These false reports could also result in arbitrary false-positive responses by either the system or the administration, severely affecting efficiency of the organization’s members and potentially leading to mistrust or resentment within the organization. This environment of mistrust could result in members leaving the organization due to dissatisfaction, provide rationale for members to become insiders in an attempt to respond to this perceived mistreatment, or could give adversaries an opportunity to influence and recruit the member for malicious purposes.

C. PROPOSED SYSTEM

Our system, referred to as BIFROST, ingests, categorizes, and responds to data from network infrastructure nodes and individual hosts to detect and defeat insider threat activities. BIFROST provides insider threat detection for organizational networks that do not possess this capability, due to prohibitive cost or resource utilization. BIFROST develops and maintains user and network baseline data using a composite statistical analysis approach with established threat criteria, which reduces required overhead resource requirements.

Figure 1 shows the proposed BIFROST system's component diagram and the flow of information within the architecture, key points of which are identified by the numbered sub-elements.

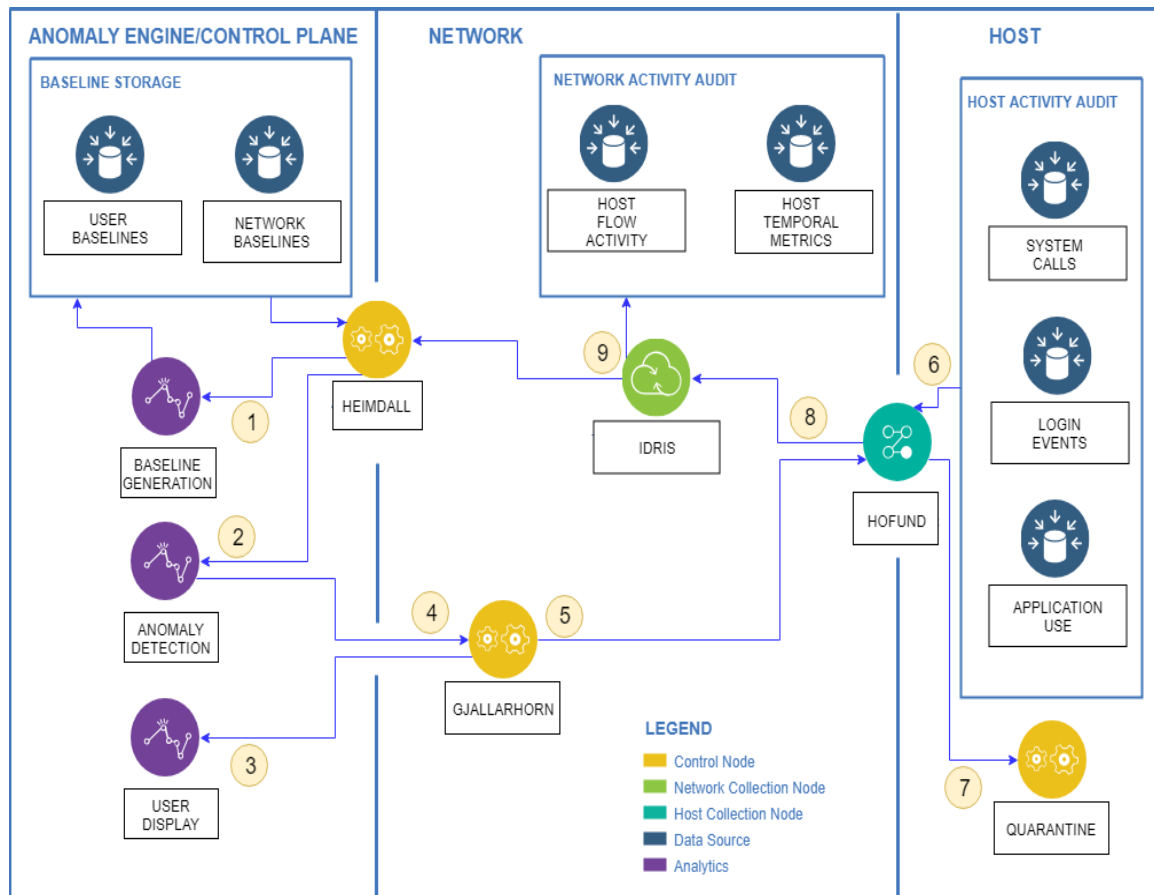


Figure 1. Proposed BIFROST System

HEIMDALL is the central reception hub for all reports from other BIFROST nodes, serving as the baseline generator for all network entities and the maintenance/update entity for the same, to integrate current and historic data and ensure a relevant metric, as seen in Figure 1, element 1. HEIMDALL also houses the anomaly detection engine for the system, seen in Figure 1, element 2, alerting against established threat response criteria and encountered anomalies as determined by its detection engine. This point-and-context anomaly analysis provides a full-spectrum representation of network and host activity, enabling the system to detect activities that a monolithic analysis method may miss. Depending on the severity of the anomaly as determined by deviation from the distribution (a deviation correlating to a GQS level of 7, for example, would be much more significant than one correlating to GQS level 3), or identification as a known-malicious activity, HEIMDALL sends either critical or informational alert to the BIFROST report reception node.

The GJALLARHORN node acts as the administrative interface for the BIFROST system, and is the executor of HEIMDALL alerts as seen in Figure 1, elements 3 and 4 respectively. GJALLARHORN provides the administrator a snapshot of current operations within the network, current alerts from HEIMDALL, and historic information about the network. From this interface, the administrator can also review a host's current anomaly index, based on deviation from historic data and their system activities. Upon receipt of an alert from HEIMDALL, GJALLARHORN displays an alert to the operator for response; if the alert is critical, the system independently responds with an isolate response. The operator can acknowledge a non-critical report with either an ignore or isolate command. The isolate command, received from either an operator or the system, sends a quarantine message to the offending host node, as seen in Figure 1, element 5, which results in the prevention of any network connections to/from the node and restricts the user to a limited set of system operations until an administrator lifts the quarantine.

Host nodes in BIFROST are known as HOFUND and serve as monitoring platforms for activities originating from the target device, seen in Figure 1, element 6. HOFUND logs and reports on the actions taken by the user and any utility in use, to include program execution, host or networked file system (NFS) interaction, network operations and

hardware use; the information reported varies by event type, but minimum information collected includes the system call intercepted and the principle arguments for the call. For example, a new program execution report might include the system call for `execve`, the target program executable of the call and the arguments for the new program. In addition to its role as a reporting node, HOFUND also functions as the executor of GJALLARHORN's quarantine commands. Upon receipt of a quarantine message, HOFUND will block the execution of any program not included in a list of acceptable utilities and will prevent the host from making network connections, as seen in Figure 1, element 7. By doing so, HOFUND will prevent the suspected insider from performing any potentially dangerous actions until a review by administrative personnel is completed.

Finally, nodes residing on organizational network infrastructure are labeled IDRIS. These nodes monitor network metrics and relay messages from HOFUND nodes in a given network region traversing to the HEIMDALL node as seen in Figure 1, element 8; to accomplish its monitor function, IDRIS must reside on the gateway device for each subnet, ensuring that it has access to all network activity for accurate metrics. IDRIS uses a simple network accounting methodology to maintain network information, reporting information about each subnet host's connections to various target services and endpoints as seen in Figure 1, element 9; IDRIS does not perform general packet or payload inspection, as such action requires a greater resource allocation, likely exceeding the host device's capacity.

D. SUMMARY

This chapter described our methodology for generating and maintaining target user and network profiles, identifying outlying anomalies, and detecting inherently malicious insider actions. We further described the general framework for our monitoring and reporting system, BIFROST, to include its intended use and prevention capabilities. We analyzed potential subversion shortcomings for such a system, giving insight into considerations for system-hardening requirements for DCO and OCO planning purposes. Our experiment environment setup and system integration steps were shown, leading to our initial results and required optimization steps, which we describe in the following chapter.

IV. EXPERIMENT DESIGN AND INITIAL RESULTS

A. INTRODUCTION

In this chapter, we present our proof-of-concept system design and implementation, and initial testing results. Specifically, we examine our system’s performance on time and memory requirements as well as its accuracy on detection of anomalous or malicious activities. We provide an overview of success rates of the system against simulated benign and anomalous data, as well as an investigation of our system’s theoretical success rates using extrapolation of our initial results.

B. EXPERIMENT DESIGN AND SETUP

To effectively test the BIFROST framework on data from a real-world network, we coordinated with the Naval Postgraduate School (NPS) Information Technology and Communication Services (ITACS) center to establish a network tap to internal network traffic. This tap was a 10Gbps connection created by a virtual network profiler system, which passed all traffic that initiated or terminated within the network. Using this data, we observed the traffic application and size distributions shown in Figure 2, denoting the distribution of Glasgow traffic, which shows the traffic application breakdown of a typical weekday for our target network. Of note, ~44% of the observed traffic was encapsulated wireless traffic propagating over UDP port 23233 using a Generic Routing Encapsulation (GRE) protocol proprietary to Ruckus Wireless as shown in Figure 3. However, using a modified version of a dissector utility created by ITACS, we were able to extract these packets and account for them within the network model for our experiment [38]. Notably, the GRE traffic was primarily encapsulating web traffic (~85%), with the remainder being mostly composed of mail, print and SSH activity. Using data collected over a typical workday, we created a representation of the network’s normal operational composition, segregating data by application and clock hour to find representative distributions usable in generating simulations of applications use, as seen in Figure 4.

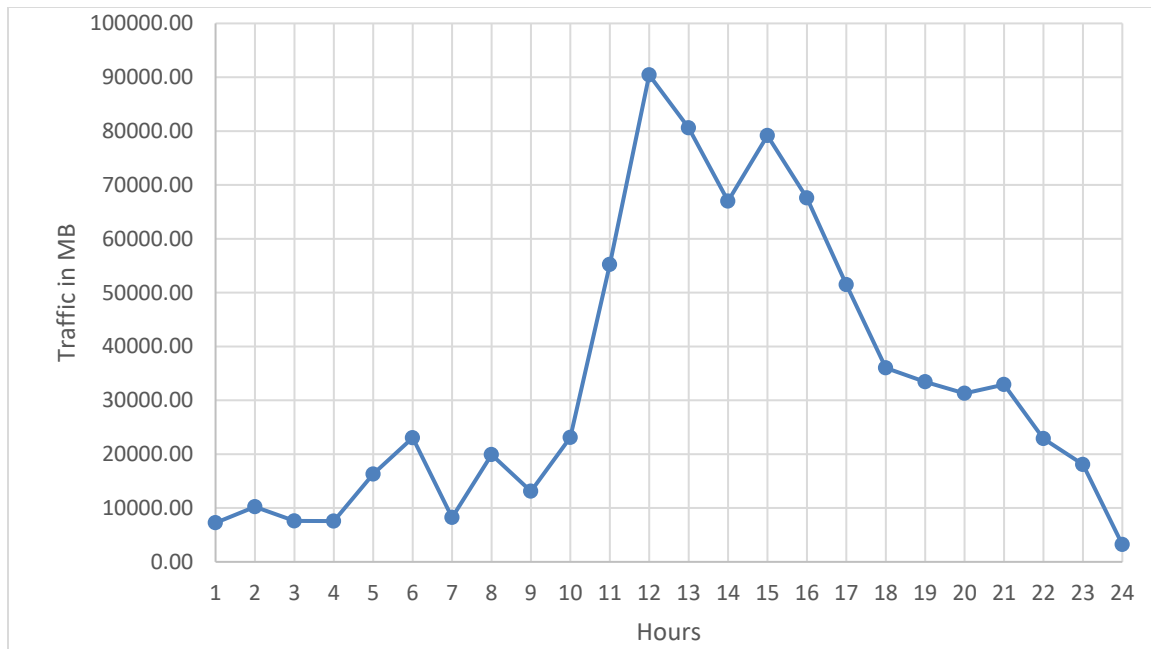


Figure 2. NPS Glasgow Network Traffic

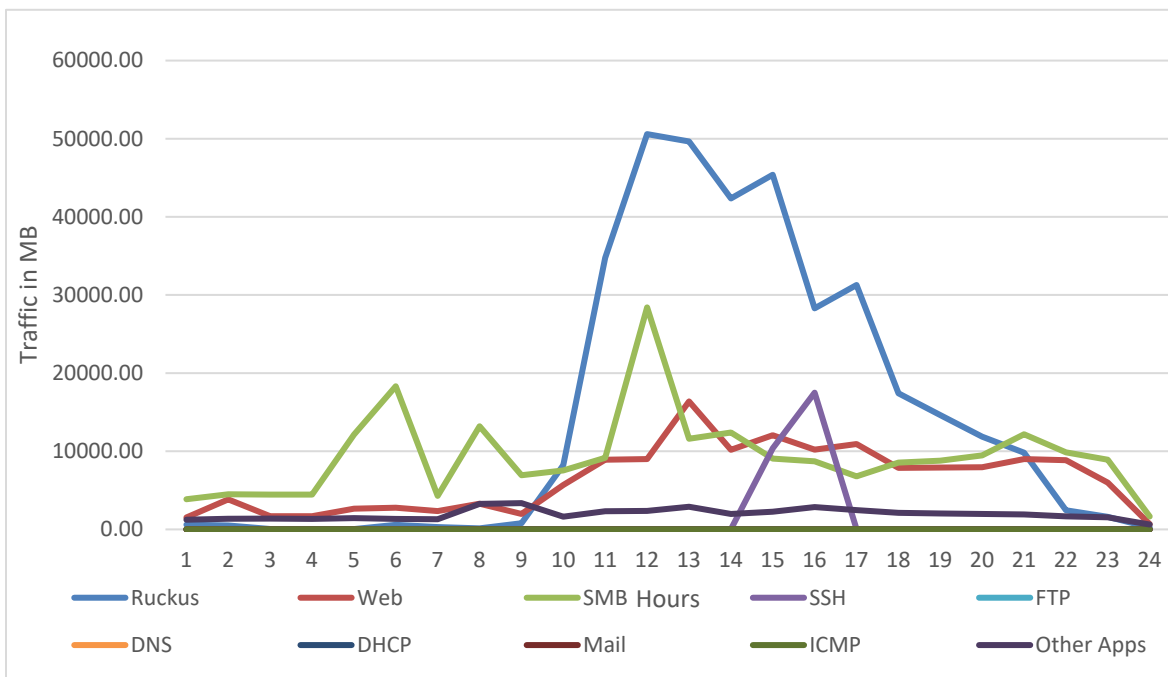


Figure 3. NPS Glasgow Network Traffic Application Distribution

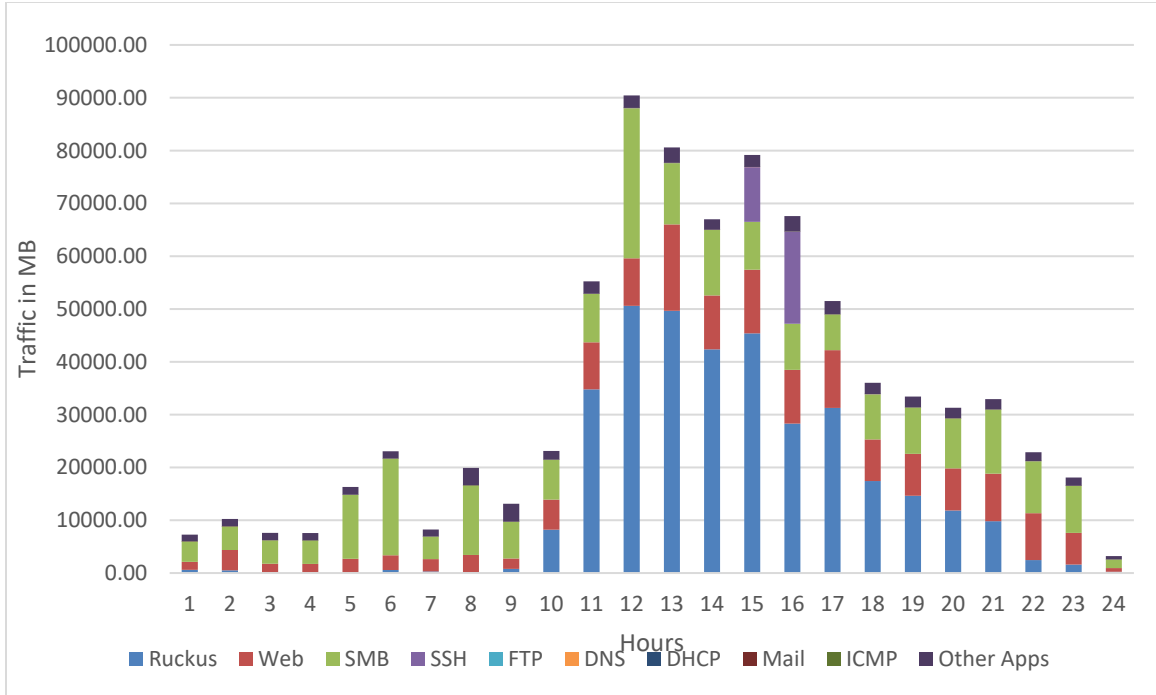


Figure 4. NPS Glasgow Weekday Network Traffic Hourly Application Graph

The experimental system environment was contained in a virtual network consisting of one CentOS 7 server housing the HEIMDALL and GJALLARHORN nodes, three VyOS machines providing routing services and housing the IDRIS nodes, and eight Lubuntu desktops and two Ubuntu servers for the HOFUND nodes. These devices were initialized with “normal” utilities and services enabled, and with standard security policies in place such as Access Control Lists (ACLs) and firewall rules. All basic services being hosted within the NPS network (email, domain name services [DNS], secure shell [SSH], etc.) were replicated within the test network, with the exclusion of the Ruckus GRE tunnels, ITACS secure network management utilities, and DoD-specific network restrictions.

From the generated network activity model, a random selection method was used to select users for simulation in our experiment. In order to ensure the widest representation of user activity, only users who were active within at least one standard deviation of the mean number of flows and packets within the distribution were selected for simulation. With our target users selected, their traffic patterns were simulated using a modified version of the open-source tool PyTgen, used to script the various connection events

observed by our experiment [39]. To represent normal user host activity, shell scripts were written with a variety of user commands, using the TWOS and CERT datasets as a basis along with insights from Liu’s 2018 study to align activity patterns examined with simulated user schedules [40]–[42]. To represent malicious behavior, additional scripts were written to implement known malicious operations, as defined in Chapter III, sections B-2 and B-3; additionally, scripts composed of random program executions and network connections provided “anomalous” activity injections. Use of these scripts was accomplished via random selection, with “normal” scripts set to be selected at a rate significantly higher than the “abnormal” activities (~85%); further, the “abnormal” events only began after the system baseline had been established.

The BIFROST network was installed and initialized on each device within the network, and the HEIMDALL node began initial baselining operations immediately upon initialization. To enable quick adjustment to the system and multiple trials, the HEIMDALL node was initialized using a one-day baseline as the initial setting, and an initial weight for updates of 85/15 as previously described in Chapter III, Section B-2. This setting may have reduced the fidelity of our baseline construction, as described in Chapter III, Section B-2, but we believe that the low volume of input data is not severely impacted by this change; future trials should examine the efficacy of changing to a longer initial baseline period., e.g., two weeks or one month.

C. INITIAL RESULTS

Two separate trials of the system were conducted, one composed of mixed anomalous and benign events to simulate an active target environment, and a controlled trial using only benign “normal” events; in each trial, three separate hosts were targeted. During the experiment set trial, Host 1 produced 6,812 benign and 1,845 anomalous events; Host 2 produced 7,023 benign and 1703 anomalous events; and Host 3 produced 5,222 benign and 2,409 anomalous target events. The anomalous events within these datasets were mixed between context anomalies (data size, temporal characteristics, irregular file/application use, etc.) and point anomalies (known bad destinations, dangerous application use, etc.). Specifically, Host 1 produced 1,508 context and 212 point anomaly events;

Host 2 produced 1322 context and 187 point anomaly events; and Host 3 produced 2128 context and 179 point anomaly events.

The system detection results for this trial appeared numerically impressive, detecting 1,620 anomalous events for Host 1, 1,599 anomalous events for Host 2 and 422 anomalous events for Host 3. These detection events break down to 1,445 context anomaly and 175 point anomaly events for Host 1; 1,454 context anomaly and 154 point anomaly events for Host 2; and 337 context anomaly and 85 point anomaly events for Host 3. At first glance, it appears that the system accounted for 87.8%, 93.9% and 17.5% of the anomalous events for Hosts 1, 2 and 3, respectively; the rate for Host 3 was significantly lower than the other Hosts, and is investigated further in section D-1. Unfortunately, the system was not setup to output the exact event that triggers a detection, rendering further analysis of this trial problematic. Assuming that all recorded detection events were accurate, the detection rates for the three Hosts and the initial trial overall are shown in Table 2.

Table 2. Detection Statistics for BIFROST Using Mixed Anomalous and Benign Data under Accurate Detection Assumption

Data Source	P	N	TP	TN	FP	FN	PREC	REC	FNR	FPR	ACC	F1
Host 1	1620	7037	1620	6812	0	225	100%	87.8%	12.2%	0%	97.4%	93.5%
Host 2	1599	7127	1599	7023	0	104	100%	93.9%	6.1%	0%	98.8%	96.8%
Host 3	422	7209	422	5222	0	1987	100%	17.5%	82.5%	0%	73.9%	29.8%
Overall	3641	21373	3641	19057	0	2316	100%	61.1%	38.9%	0%	90.7%	75.9%

P – Number of events alerted on; N – Number of host events not alerted on; TP – Number of true positive alert events; TN – Number of true negative alert events; FP – Number of false positive alert events; FN – Number of false negative alert events; PREC – Precision, or positive predictive value, of test; REC – Recall, or true positive rate, of test; FNR – False negative rate of test; FPR – False positive rate of test; ACC – Accuracy of test (comparison of result and true value); F1 – F-score of test

During the controlled trial, Host 1 produced 1,993 events, Host 2 produced 1,822 events, and Host 3 produced 3,967 events. During this trial, the system detected 274 context anomaly events for Host 1, 317 context anomaly events for Host 2, and 168 anomaly events for Host 3. Considering that there were no intentionally anomalous events, the detection rates for this controlled trial are shown in Table 3.

Table 3. Detection Statistics for BIFROST Using Known Benign Data

Data Source	P	N	TP	TN	FP	FN	FPR	ACC
Host 1	274	1929	0	1929	274	0	12.4%	87.6%
Host 2	317	1745	0	1745	317	0	15.4%	84.6%
Host 3	168	3909	0	3909	168	0	4.1%	95.9%
Overall	759	7583	0	7583	759	0	9.1%	91%

P – Number of events alerted on; N – Number of host events not alerted on; TP – Number of true positive alert events; TN – Number of true negative alert events; FP – Number of false positive alert events; FN – Number of false negative alert events; FPR – False positive rate of test; ACC – Accuracy of test (comparison of result and true value)

Based on the collected data from both trials, as shown above, we extrapolated a false positive rate for the system between 4.1% and 15.4%; however, this is only an assumption based on available data and additional testing is required to verify it. Using this range as the best-case and worst-case false positive rates, respectively, Table 4 shows the potential composite performance of the proof-of-concept system against the experiment dataset.

Table 4. Theoretical Detection Statistics for BIFROST Using Mixed Anomalous and Benign Data under Extrapolated Detection Rate Assumptions

Data Source	P	N	TP	TN	FP	FN	PREC	REC	FNR	FPR	ACC	F1
Best-Case Scenario	3641	21373	3492	19057	149	2316	95.9%	60%	40%	4.1%	90.1%	74%
Worst-Case Scenario	3641	21373	3080	19057	561	2316	84.6%	57.1%	42.9%	15.4%	88.5%	68.2%

P – Number of events alerted on; N – Number of host events not alerted on; TP – Number of true positive alert events; TN – Number of true negative alert events; FP – Number of false positive alert events; FN – Number of false negative alert events; PREC – Precision, or positive predictive value, of test; REC – Recall, or true positive rate, of test; FNR – False negative rate of test; FPR – False positive rate of test; ACC – Accuracy of test (comparison of result and true value); F1 – F-score of test

D. ANALYSIS

1. System Detection Accuracy

Notably, there was significant deviation between the results for Hosts 1 and 2 and the results for Host 3. Inspecting the raw detection numbers (P) for the first trial, Host 3's baseline appeared to be 73.9% and 73.61% less-effective than those of Hosts 1 and 2 in detecting anomalous behavior. Additionally, inspecting the statistical analysis of the trial's results shows that the detection events targeting Host 3 were far less-accurate than that of the other two Hosts; specifically, its false negative rate was 576.2% higher than that of Host 1 and 1252.5%% higher than that of Host 2. However, the controlled trial showed a different view of these results, as Host 3's inputs triggered less alerts than Hosts 1 and 2 did against benign inputs, which ultimately caused its false positive rate to be 66.9% lower than Host 1 and 73.4% lower than Host 2. These results seem to indicate that the detection threshold for Host 3 was more stringent than that of Hosts 1 and 2, reducing the number of alerts triggered. Strangely, there were no indicators of bad data within the baseline output of Host 3, and its logged activities did not appear to be excessively anomalous compared to the baseline; further, no threshold value changes were conducted during any trial. Additional testing with a greater number of target hosts is required to determine if this was an issue with input from that target or if the deviation is an indication of additional system instability or inaccuracy and whether the system's threshold values are prone to corruption and change as the experiment progresses.

Using only the initial results of the system's performance against the trial data it is not possible to gather an accurate assessment of the system's accuracy due to the way in which BIFROST generates its threat assessment. Specifically, the system does not produce an alert or message each time an anomaly is detected due to the threshold requirements for threat alerts and the volume of data ingested by the system. To counteract this, the controlled trial used only known benign data, which provided an environment to test the system and determine what potential false positive rates may be present. Using this information, the results in Table 4 were generated to extrapolate potential best and worst-case detection rates. From this data, the system's current development state may provide

some detection capability against insider threat activities, but it potentially misses 40+% of all anomalous events. This, paired with the potential worst-case false positive rate of 15.4%, results in a system that is currently quite inaccurate and that would provide minimal protection against the insider threat.

However, these proof-of-concept trials were conducted using a small target sample size with an artificially low event population. With a greater body of data to perform baseline construction and comparison against, there is potential for this system's accuracy rates to increase and provide greater fidelity against insider threat activities. As a proof-of-concept demonstration, the detection rates (less the disappointing false negative rates) observed were fairly impressive and provide a starting point of comparison for further development. Specifically, the thresholds used to determine anomalousness were arbitrary in nature; significant additional research is required to develop and implement relevant threshold and threat score values to increase system capability. Additionally, the future inclusion of machine learning should be explored to dynamically adjust the threshold and threat score values to account for variations between target organizations.

2. Baseline Generation Performance

Our initial system baseline construction design performed reasonably well within our test environment, producing a set of initial baseline comparison data for the three target hosts in less than 6 seconds using the available simulated inputs; this increased to ~10 seconds when conducting the baseline update process. As currently implemented, the system will ignore context anomalies until the established threshold threat value has been met, after which point it identifies any subsequent anomalies with reasonable sensitivity. These initial below-threshold anomalies provide significant opportunity for subversion if a malicious actor is aware of the "resetting" that occurs during each baseline period.

3. System Interface

The current system does not provide for significant administrator interaction, limiting the commands available to simple baseline print capabilities, alert test and detection threshold manipulations. These limitations will severely degrade the system's ease-of-use, impede analysis by users unfamiliar with the system, and limit the utility it

provides for operators on live systems. In order to make changes to the system or adjust most values, the current implementation requires that the modified subsystem be completely halted and restarted after making modifications to the source code. Production environments cannot support such requirements, necessitating the introduction of a live alteration method with networked change promulgation capability.

4. Observed System Shortcomings

We observed some shortfalls in our initial proof-of-concept system that can be improved upon in future implementations. For example, our design choice to store target network and host data in system memory rather than on the disk resulted in excessive memory usage, although it did not cause perceptible system degradation or throttling. Still, it is likely that this would result in resource scarcity if the number of monitored hosts or the volume of monitored activities increased greatly. Additionally, while our algorithm performed well in generating comparison data for anomaly detection, its asymptotic complexity resulted in peak CPU utilization of nearly 100%. The algorithm would likely produce significant delays in generating data when the input magnitude increases, and should be made more efficient.

Due to the data restrictions in our initial experimentation, we did not intermix malicious and benign activities within the initial baseline data, which does not reflect real-world network traffic or host activity. If the data had been intermixed, there would be the real potential for some malicious activities to be overlooked, due to their inclusion in the baseline. We believe that the anomalous nature of insider activities would overcome this and still provide ample means for detection, however, further study and testing are required to confirm these expectations.

E. SUMMARY

In this chapter, we presented initial BIFROST detection system operational results utilizing simulated organizational network traffic and host program activity. Our experiment design was described to include the methodology used to simulate a target organization's traffic model and the activities of various hosts within the network to provide a more realistic test environment for our concept system. Finally, we examined the

initial results and identified the system's detection accuracy and sensitivity using initial proof-of-concept design configurations, and identified potential system shortcomings with the current design's limitations. In our next chapter, we discuss the implications of these results and provide suggestions for future extension of this work.

V. CONCLUSIONS AND RECOMMENDATIONS

In this work, we identified shortcomings in existing methods and systems for insider threat detection, in both academic and commercial settings. In answer to these shortcomings we proposed the BIFROST system, an insider threat detection framework that utilized concepts from other established detection frameworks. To test our system framework, we developed a simulated organization using network traffic data collected from the Naval Postgraduate School and host activity data from the TWOS and CERT datasets. Finally, we analyzed and presented our initial results, which showed that the current system design has the potential to be developed into a robust detection platform, given additional research.

A. CONCLUSIONS

As stated in the previous Chapter, during our trial we observed an F-Score of ~78% regarding system accuracy against simulated target activity and a false positive rate of ~4-15% using a controlled trial. From these initial results, we extrapolated that our proof-of-concept system implementation resulted in a best-case theoretical true positive rate of ~60%, a false positive rate of ~4%, a false negative rate of ~40% and an F-Score of ~74%. Conversely, its worst-case true positive rate was ~57%, its false positive rate was ~15%, its false negative rate was ~43% and its F-Score was ~68%. Using these extrapolated detection rates, the system would miss ~40-43% of threat conditions and produce a significant number of false alarms. We believe that our initial results were skewed by the small sample size used for target data and the manner in which the system performed anomaly detection and the arbitrary threshold and threat values used. Comparatively, the system proposed by Shaghaghi, et al., had a reported successful detection rate of ~30% in its initial result, although their test results were skewed negatively by the large volume of events in their trial that were not monitored by their limited initial system implementation. Notably, the ELICIT system created by Maloof and Stephens reported an 84% detection rate and a false-positive rate of 1.5% against structured scripted simulations and red team activities.

Also of note, relatively high system utilization was observed within the system during both baseline and detection operations with regards to memory and CPU use. This did not cause issues with system performance during our experiment due to the limited volume of data examined, but could impact real-world results as input magnitude greatly increases. Future system implementations must define more meaningful threshold values, and improve efficiency of the system’s algorithm for baseline generation and detection operations, to handle greater data sample sizes from live targets on a production network.

We believe that this initial framework has the potential to be developed into a system capable of mitigating portions of the insider threat, specifically for organizations that cannot field a commercial solution due to prohibitive costs and resource requirements. This will require further refinements and system improvements, as described in Chapter IV, and additional future capabilities outlined below.

B. FUTURE WORK

1. Host Activity Granularity and Malicious Activity Interception

Our system’s ability to characterize and profile host-level actions is reliant on the capability to intercept and inspect the activity of the host’s active user. We chose to implement through the use of the Linux System Audit feature, but this method is far from ideal. While this does provide for a means to collect many instances of host activity, some functions are not necessarily seen by this method and the information provided by default through the System Audit logs may not provide as much granularity as needed to monitor for insider activities. This method supports the capability to block certain dangerous activities through its ACL-like functionality, but it does not prevent the manipulation of its configuration file, nor monitor such activity.

One potential answer to this issue would be to implement a mechanism similar to the DRAKVUF system developed by Lengyel et al. [43] or the Syswall tool in development by Pugnet [44]. The DRAKVUF system provides a testing environment for malware analysis, acting as the intermediary for system activity between a virtualized guest operating system and the suspected malware. It intercepts and inspects every system call created from the suspect program and conducts analysis of the context, and could

potentially be used as a basis for developing a system to block execution of calls deemed dangerous. Similarly, the Syswall tool provides a firewall-like functionality for Unix systems, inserting itself between system calls and the kernel, effectively isolating all system calls on the machine until they have been approved by the system's validation mechanism. However, the DRAKVUF method is unfortunately reliant on a virtual environment, which is not necessarily a given in many organizations, and the Syswall system is still in early development. These are, however, very interesting works with regards to identifying new methods of system call observation and control, and could be valuable starting points for any future improvements to our system.

2. Network Observation

Our proof-of-concept system ingested a limited range of network packet types in order to scope the initial work. Any further experimentation must include implementation for autonomous collection of non-TCP traffic and non-IPv4 traffic. This will allow for a more complete coverage of target activity and provide a more accurate representation of system operations. Additionally, further expansion of the system's monitoring should include resolution of traffic destinations to provide more granularity to host and network activities.

3. Administrative Interface

Our initial system provides limited capability for administrator interaction, specifically tied to tailored commands via TCP connections. The final proposed system should include a Graphical User Interface (GUI) to provide the administrator the ability to view current network statistics and alert information at a glance. To this end, further work should develop and implement these administrative tools, and include the ability to adjust weights and configurations without necessitating system restart or adjustment to the code itself. Additionally, real-time monitoring and display of current host threat scores, similar to a scoreboard of the hosts considered the most anomalous or potentially malicious in the enterprise, should be available to the administrator, with the option of adjusting the scores based on insight or other data. A GUI interface would allow the operator to visually inspect

patterns and trends, and provide a means to introduce organizational expertise to the process, which may potentially increase system efficacy.

4. System Efficiency

The HEIMDAL node took ~6 seconds to configure an initial baseline for three target hosts and ~10 seconds to update the baselines for each update period. The time required to generate the baseline is a product of the amount of data ingested by our system, the algorithmic inefficiency of our baseline generation, and the number of users. Considering this with the assumption that resource cost will increase drastically with a large host population, we believe that conducting the updates only during off-hours or periods of otherwise low network activity would be optimal for system utility. By limiting the update period to only ostensibly low activity periods, the amount of data compared against a, potentially, outdated baseline will be minimized, increasing system accuracy. Efforts to improve this time were not explored in this work, but would likely be readily achieved by utilizing a non-interpreted language, more efficient data structures, maintaining an on-disk database of baseline information vice maintaining it in memory and/or by making algorithmic improvements.

Additionally, extensive research into thresholds and psychological analysis of insider activity on system architecture must be undertaken to define a more accurate foundation for appropriate threshold values. Along with this, effort should be made to integrate a machine learning component to this system's threshold management and analysis of activity patterns; integrating such a component would provide our system with the means to dynamically adapt and specialize itself for the target network's norms and provide a much more accurate tool.

5. OCO Considerations

We noted potential methods in Chapter III to manipulate or subvert this system; however, defeating such efforts is beyond the scope of this work. An additional detection suite should be implemented to conduct in-depth analysis for malicious actor activities within the target network. This capability must be supported by a dedicated malware/malicious logic detection system. More insidiously, if the attacker were to determine the

message formatting and delivery method for alerts to the HOFUND host-based nodes, the attacker could conceivably introduce a great deal of confusion to the network by initiating quarantines on various hosts without system intervention, requiring system administrators to manually investigate and resolve each incident.

We believe that protective measures could be developed and integrated into our system without significant impact to operational efficiency. Such an example could be the implementation of an encrypted challenge/reply between the nodes, or a more robust accounting methodology for tracking sent and received messages. Efforts to test manipulation and subversion prevention effectiveness should be of highest priority, given the significance of the threat to the system and the potential to develop or improve existing OCO tradecraft against such systems.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. HEIMDALL

```
#!/usr/bin/python3.6
```

```
"""
```

SINGLE DAY OPTION

BIFROST HEIMDALL MODULE 30AUG2018
Scott Findley

Attempts to consolidate insider threat detection and monitoring functions within five separate objects: HOFUND the host sensor, IDRIS the network sensor, GJALLARHORN the response vector and HEIMDALL the anomaly engine/classifying authority

This system conducts analysis of user/network activity and baselines to create a "known good" state to conduct anomaly deviation analysis. This baseline is updated bi-weekly (daily in the proof-of-concept) with a weight of 85/15 against new information unless administrators confirm new input as more valid (i.e. user's responsibilities/duties have changed permanently).

HEIMDALL functions as the primary agent for the BIFROST system. It performs baselining of the target environment, detects anomalies in network/host activity, and maintains the current state of the enterprise. HEIMDALL conducts analysis of various data points and determines the likely threat classification of input characteristics if they prove anomalous or otherwise suspicious. With these developed classifications and administrator/developer-established thresholds for response, HEIMDALL leverages the GJALLARHORN agent for threat response. [NOTE: HEIMDALL and GJALLARHORN should reside on the same system, IDRIS may reside on the same system if part of small deployments]

30AUG2018 - Scott Findley - Split modules from main code for ease of readability.

30AUG2018 - Scott Findley - Added primitives for connections, crypto and basic flow logic

27NOV2018 - Scott Findley - Added additional planning comments, adjusted flow

04DEC2018 - Scott Findley - Adjusted input parsing

16DEC2018 - Scott Findley - Finalized initial system flow

12JAN2019 - Scott Findley - First trial baseline operation

20JAN2019 - Scott Findley - Identified issue with input data, changed storage/utility format

29JAN2019 - Scott Findley - Adjusted baseline operation

03FEB2019 - Scott Findley - Adjusted baseline operation, reduced comparison types

12FEB2019 - Scott Findley - Implemented detection system

21FEB2019 - Scott Findley - Adjusted detection system, implemented tiered comparison and threshold

13MAR2019 - Scott Findley - Identified issue with baseline, corrupting data after 3+ day input
22MAR2019 - Scott Findley
"""

```
import socket, threading, sys, os, pickle, time
import sqlite3 as SQL
import scipy.stats as stats
import pandas as pd
import numpy as np
import sklearn.cluster
import datetime
from time import time as get_time
import matplotlib.pyplot as plt
#from cryptology.fernet import Fernet
import json

class HEIMDALL: #classification center

    #receive inputs independent of others
    #develop baseline for users/network (24 per day, update bi-weekly)
    (one for weekday and one for weekend, maybe?)
    #each user has own baseline for network and host activity
    #user groups have own baselines for network and host activity
    #update baseline for new users/hosts as they appear
    #update baseline every 15 days, 85/15 weight against new data
    #provide enterprise updates to GJALLARHORN
    #perform anomaly processing against new input based on time/host
    involved
    #administrative events extremely suspicious if not from
    administrative role
    #anomalous volume/time events extremely suspicious (off-hours, huge
    file reads/writes outside norms)
    #perform statistical analysis to determine GSQ/cluster deviation
    #if classified as malicious, create alert to GJALLARHORN and create
    user threat profile entry (logs event type/context, UID, role, threat
    level)

    def __init__(self,
HEIMDALL_port,GJALLARHORN,storage,update,GJALLARHORN_port,IDRIS_list):
        self.GJALLARHORN = GJALLARHORN #default address for GJALLARHORN
is localhost
        self.GJALLARHORN_port=GJALLARHORN_port
        self.IDRIS=IDRIS_list
        self.HEIMDALL_port=HEIMDALL_port
        self.up_time=get_time()
        self.up=time.strftime("%DATE:%H:%M:%S",
time.gmtime(self.up_time)) #time of node start in GMT
        #should update previous day at start of new day
        self.interval=86400 #TESTING USE ONLY ### normal interval is 7
days 604800
        self.start_interval=86400 #TESTING USE ONLY ### normal interval
is 14 days 1209600
        self.first_run=True
```

```

        self.update_time=get_time()+ self.start_interval #14 day
initial update timeframe
        self.connections=[]
        self.baseline_loc="/home/hub/data/baseline"
        self.update_loc="/home/hub/data/update_data"
        self.cluster_loc="/home/hub/data/cluster"
        self.cluster_update_loc="/home/hub/data/cluster_update_data"
        self.setup=get_time()-self.up_time #time to setup
        self.targets=[]
        self.ready_users=[]
        self.new_users=[]
        self.clusters={}

self.days={0:"monday",1:"tuesday",2:"wednesday",3:"thursday",4:"friday"
,5:"saturday",6:"sunday"}
        self.update_factor=2 #2 day comparison (two weeks)
        self.cluster_factor=0

self.BLK_LIST={"209.99.40.222":"badguy.com","82.98.86.163":"mod.ru","81
.28.34.108":"wehaku.cn","14.49.30.92":"nukes.ir"}
        self.MAL_LIST=["seccomp","mmap"]
        self.ADM_LIST=["ssh","telnet","scp","strace","sysctl"]
        self.ALERT_stop=False
        self.up_date=datetime.datetime.now()
        self.threshold=4500
        self.d_alerts=[]
        self.point_alerts=0
        self.context_alerts=0
        self.anomaly_alerts=0
        self.data_size=0
        self.active=False

def run(self):
    os.system("clear")
    print("\n\nHEIMDALL NODE ONLINE AT %d:%d %d-%d-%d\n\n" %
(self.up_date.hour,
self.up_date.minute,self.up_date.day,self.up_date.month,self.up_date.ye
ar))
    with open("terminal.txt","a+") as file:
        file.write("HEIMDALL NODE ONLINE AT %d:%d %d-%d-%d\n\n" %
(self.up_date.hour,self.up_date.minute,self.up_date.day,self.up_date.mo
nth,self.up_date.year))
    #start primary loop
    if self.first_run==True: #check for existing baseline data
        try:
            with open(self.baseline_loc,'r') as baseline_file:
                x=baseline_file.readline()
                if x:
                    self.first_run=False
                    self.listen()
                else:
                    self.first_run=False
                    self.listen()
        except Exception as e:
            print(e)

```

```

        pass
    elif self.first_run==False:
        self.listen()

    def listen(self):
        #receive data from IDRIS, do anomaly detection
        #set up listener
        IDRIS=self.IDRIS
        serverSock=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        serverSock.bind(("0.0.0.0",self.HEIMDALL_port))
        serverSock.listen(50) #50 backlogged connections acceptable
        #spin off thread for each connection
        timer=threading.Thread(target=self.timer)
        timer.start()
        while True:
            sock,client=serverSock.accept()
            if (client[0] not in IDRIS):
                #do authentication here, if unauthenticated,
IDRIS[client]=False
                #IDRIS[client] = True
                #if IDRIS[client] != True:
                #sock.close()
                sock.close()
                sock=0
                print("Unauthorized connection from %s" % client[0])
                client=0
                continue
            if self.first_run==False: #if not first run, test input
against baseline

            client_t=threading.Thread(target=self.detect,args=(sock,client[0]))#(cl
ient))
                self.connections.append(sock)
                client_t.start()
            else: #if frist run, just add to update data

            updater=threading.Thread(target=self.update,args=(sock,self.update_df,s
elf.cluster_update_df))
                self.connections.append(sock)
                updater.start()

    def timer(self):
        newday=False
        while True:
            check=self.update_time-get_time()
            if (check<=0):
                if datetime.datetime.now().hour != 15: ####TESTING ONLY,
NORMAL VALUE IS 23
                    pass
                else:
                    self.update_time=get_time()+self.interval
                    baseliner=threading.Thread(target=self.baseline)
                    baseliner.start()
            now=datetime.datetime.now()
            if (newday==True) and (now.strftime('%H:%M')== "00:00"):

```

```

        newday==False
    if (now.strftime('%H:%M')== "01:00"):
        newday=True

    def update(self, client, data=[], addr=0, role=0):
        #data =
        [[usr_addr,usr_role],{(sport,dport,dest):[pkt,size],},{call:[cnt,params
        ],}]

        if client ==0: #data available from detect
            foun=0
            for x in self.ready_users:
                if x.addr==addr:
                    user=x
                    foun=1
                    break
            for x in self.new_users:
                if x.addr==addr:
                    user=x
                    foun=1
                    break
            if foun != 1:
                user=USER(addr,role)
                self.new_users.append(user)
                hour=datetime.datetime.now().hour
                day=datetime.datetime.today().weekday() #0-6 Monday-Sunday
                user.u_update(data,hour,day)
        else:
            try:
                data=client.recv(2048) #report=[target,(host OR net),
                ((call: count, parameters) | (flow: cnt, size))]
                report=pickle.loads(data)
                target=report[0]
                client.close()
                if report[1] == "host":
                    ca=report[2]
                    for x in ca.keys():
                        call=x
                        dat=ca[x]
                        cnt=dat[0]
                        param=dat[1]
                    data=[0,0,0,call,cnt,param]
                elif report[1] == "net":
                    f=report[2]
                    for x in f.keys():
                        flow=x
                        dat=f[x]
                        cnt=dat[0]
                        size=dat[1]
                    data=[flow,cnt,size,0,0,0]
                elif target[1]=='print':
                    client.close()
                    return
                elif target[1]=='active':
                    self.active=True
                    self.first_run=False

```

```

        client.close()
        dt=datetime.datetime.now()
        print("\nDETECT MODE ENABLED FROM UPD AT %d:%d on
%d-%d-%d\n" % (dt.hour,dt.minute,dt.day,dt.month,dt.year))
        return
    client.close()
    print(target)
    self.connections.remove(client)
    foun=0
    for x in self.ready_users:
        if x.addr==target[0]:
            user=x
            foun=1
            break
    for x in self.new_users:
        if x.addr==addr:
            user=x
            foun=1
            break
    if foun != 1:
        user=USER(target[0],target[1])
        self.new_users.append(user)
        hour=datetime.datetime.now.hour()
        day=datetime.datetime.today.weekday() #0-6 Monday-
Sunday
        user.u_update(data,hour,day)
    except Exception as e:
        print("Error receiving update from IDRIS node")
        print(e)
        return

def baseline(self):
    #get mean, sd, variance for network on each hour
    #find distribution of data
    #find GSQs (LVL 1-5)
    #find k-means++ clusters
    running=[]
    start=get_time()
    x=datetime.datetime.now()
    hourz=x.hour
    dayz=x.weekday()
    day=self.days[x.weekday()]
    fd=open("terminal.txt","a+")
    print("System baseline operation started at %d:%d on %s %d-%d-
%d" % (x.hour,x.minute,day,x.day,x.month,x.year))
    fd.write("Baseline started at %d:%d %d-%d-%d\n" %
(x.hour,x.minute,x.day,x.month,x.year))
    x=0
    users=[]
    try:
        for user in self.ready_users:
            x=user.addr
            users.append(x)
            baser=threading.Thread(target=user.base,
args=(hourz,dayz))

```

```

        running.append(baser)
        baser.start()
    for user in self.new_users:
        x=user.addr
        users.append(x)
        baser=threading.Thread(target=user.base,
args=(hourz,dayz))
        running.append(baser)
        baser.start()
    except Exception as e:
        print("Issue with baseline ",e)
    for runner in running:
        runner.join()
    for user in self.new_users:
        self.ready_users.append(user)
        self.new_users.remove(user)
    x=datetime.datetime.now()
    for user in self.ready_users:
        self.data_size+=user.size
        user.size=0
    if len(self.ready_users)==0:
        total_size=0
    else:
        total_size=self.data_size/len(self.ready_users)
    print("%d bytes in use for baseline data (~%.2f bytes/host)" %
(self.data_size,total_size))
    fd.write("%d bytes for baseline (~%.2f B/host)\n" %
(self.data_size,total_size))
    self.data_size=0
    total=get_time()-start
    if len(self.ready_users)==0:
        total_per_host=0
    else:
        total_per_host=total/len(self.ready_users)
    print("System baseline for operation complete: %f seconds taken
for %d hosts (~%f seconds/host), finished at %d:%d on %s %d-%d-%d\n" %
(total, len(self.ready_users), total_per_host, x.hour, x.minute, day, x.day, x
.month, x.year))
    fd.write("Baseline complete: %f seconds for %d hosts (~%f sec/
host)\n\n" % (total, len(self.ready_users), total_per_host))
    fd.close()
    login_low=[]
    login_high=[]
    logout_low=[]
    logout_high=[]
    flow_pkt_low=[]
    flow_pkt_high=[]
    flow_size_low=[]
    flow_size_high=[]
    call_low=[]
    call_high=[]
    for user in self.ready_users:
        for entry in user.cluster_flow:
            flow_pkt_low.append(entry["cnt"][0])
            flow_pkt_high.append(entry["cnt"][1])

```

```

        flow_size_low.append(entry["size"][0])
        flow_size_high.append(entry["size"][1])
    for entry in user.cluster_call:
        call_low.append(entry[0])
        call_high.append(entry[1])
#####
#####
#self.cluster()

"""def cluster(self)
    #assign clusters to users
    #do cluster stuff"""

def detect(self,client,node):
    #data =
    [[usr_addr,usr_role],type,(({sport,dport,dest):[pkt,size]},,{call:[cnt,
params],})]
    data=[]
    BLK_LIST=self.BLK_LIST
    ADM_LIST=self.ADM_LIST
    MAL_LIST=self.MAL_LIST
    now=datetime.datetime.now()
    days=self.days
    day=(datetime.datetime.today().weekday())
    hour=datetime.datetime.today().hour
    try:
        data=client.recv(2048) #report=[[addr,role],type,{data}]
#flows=(({sport,dport,dest,time):[pkt,size]})
#calls={call:[cnt,[params]]}
        report=pickle.loads(data)
        target=report[0]
        addr=target[0]
        role=target[1]
        client.close()
        if report[1] == "host":
            ca=report[2]
            for x in ca.keys():
                call=x
                dat=ca[x]
                cnt=dat[0]
                param=dat[1]
            data=[0,0,0,call,cnt,param]
            check_flow=0
            check_pkt=0
            check_size=0
            check_call=call
            check_call_cnt=cnt
            check_call_param=param
        elif report[1] == "net":
            f=report[2]
            for x in f.keys():
                flow=x
                dat=f[x]
                cnt=dat[0]
                size=dat[1]

```

```

        data=[flow,cnt,size,0,0,0]
        check_flow=flow
        check_pkt=cnt
        check_size=size
        check_call=0
        check_call_cnt=0
        check_call_param=0
    elif role=="alert test":
        addr=target
        print("\n*****")
        print("SENDING TEST ALERT TO GJALLARHORN")
        print("*****")

test_alert=threading.Thread(target=self.alert,args=(addr[0],"DANGER",date
time.datetime.now()))
    test_alert.start()
    return
elif role=="anomaly reset":
    for user in self.ready_users:
        user.anomalies=0
        user.threat_score=0
        self.point_alerts=0
        self.context_alerts=0
        self.anomaly_alerts=0
    return
elif role=="lift test":
    addr=target
    print("\n*****")
    print("SENDING TEST LIFT TO GJALLARHORN")
    print("*****")

test_lift=threading.Thread(target=self.alert,args=(addr[0],"LIFT",date
time.datetime.now()))
    test_lift.start()
    return
elif role=="thresh":
    thresh=report[1]
    dt=datetime.datetime.now()
    print("\nCHANGED THRESHOLD VALUES FROM %d TO %d AT
%d:%d %d-%d-%d\n" % (dt.hour,dt.minute,dt.day,dt.month,dt.year))
    self.threshold=thresh
    return
elif target[1]=='active':
    self.active=True
    dt=datetime.datetime.now()
    print("\nENTERED DETECTION MODE FROM SEC AT %d:%d %d-
%d-%d\n" % (dt.hour,dt.minute,dt.day,dt.month,dt.year))
    return
elif role=="print":
    with open(self.baseline_loc,'a+') as wr:

print("\n*****")
        print("PRINTING CURRENT TARGET DATA to %s" %
self.baseline_loc)

```

```

print("*****")
    print("POINT ALERTS : %d" % self.point_alerts)
    print("CONTEXT ALERTS: %d" % self.context_alerts)
    print("ANOMALY ALERTS: %d" % self.anomaly_alerts)
    danger_data=self.d_alerts
    buffer="*****\n*****\n"
    wr.write("*****\nDANGER-LEVEL ALERTS:\n")
    for x in danger_data:
        wr.write(x)
        wr.write("\n")
    for user in self.ready_users:
        with open("trial","a+") as fi:
            print(user.addr,file=fi)
            print("Threat: %d" % user.threat_score,
file=fi)
            print("Anomalies: %d" %
user.anomalies,file=fi)
            print("Danger alerts: ",file=fi)
            print(user.danger_alert,file=fi)
            print("\n", file=fi)
            title=("HOST: %s\n" % user.addr)
            wr.write(buffer)
            wr.write(title)
            wr.write("\n")
            wr.write("Threat score: %d\n" %
user.threat_score)
            wr.write("Danger alerts:\n")
            d_data=user.danger_alert
            for data in d_data:
                wr.write(json.dumps(data))
            wr.write("\n")
            wr.write(buffer)
        print("\n*****")
        print("Data written to baseline file")
        print("*****")
        return
    else: #bad input
        print("\nBad input from IDRIS node %s:" % node)
        print(report)
        return
    client_t=threading.Thread(target=self.update,
args=(0,data,addr,role))
    client_t.start()
    self.connections.remove(client)
    found=0
    for x in self.ready_users:
        if x.addr==target[0]:
            user=x
            found=1
            break
    if found == 0:
        return #no user data available, no need to check
further
    if self.active==False:

```

```

        return
    except Exception as e:
        print("\nError receiving update from IDRIS node")
        print(e)
        return
    role=target[1]
    #user_cluster=user.baseline_cluster
    d_input=datetime.datetime.now()
    date_input_call=("%d-%d-%d | %d:%d | %s" %
(d_input.day,d_input.month,d_input.year,d_input.hour,d_input.minute,che
ck_call))
    date_input_flow=("%d-%d-%d | %d:%d | %s" %
(d_input.day,d_input.month,d_input.year,d_input.hour,d_input.minute,che
ck_flow))
    d_alert_input=("%d-%d-%d | %d:%d | %s" %
(d_input.day,d_input.month,d_input.year,d_input.hour,d_input.minute,use
r.addr))
    if check_call!=0:
        if (check_call in MAL_LIST): #known bad
            #instant alert
            print("\nALERT GENERATED FOR USER %s: Known dangerous
activity %s found" % (target[0],check_call))
            user.threat_score+=15
            if date_input_call not in user.danger_alert:
                user.danger_alert.append(date_input_call)
                self.d_alerts.append(d_alert_input)

send_alert=threading.Thread(target=self.alert,args=(user.addr,"DANGER",
datetime.datetime.now()))
    send_alert.start()
    self.point_alerts+=1
    return
    if ((check_call_param in ADM_LIST) or (check_call in
ADM_LIST)) and (role != "admin"): #admin tool without admin
        #instant alert
        print("\nALERT GENERATED FOR USER %s: Non-admin user
launched admin tool %s" % (target[0],check_call))
        user.threat_score+=15
        if date_input_call not in user.danger_alert:
            user.danger_alert.append(date_input_call)
            self.d_alerts.append(d_alert_input)

send_alert=threading.Thread(target=self.alert,args=(user.addr,"DANGER",
datetime.datetime.now()))
    send_alert.start()
    self.point_alerts+=1
    return
    if check_flow!=0:
        if (check_flow[2] in BLK_LIST):
            #instant alert
            print("\nALERT GENERATED FOR USER %s: Unauthorized
traffic to %s" % (target[0],BLK_LIST[check_flow[2]]))
            user.threat_score+=15
            if date_input_flow not in user.danger_alert:
                user.danger_alert.append(date_input_flow)

```

```

        self.d_alerts.append(d_alert_input)

send_alert=threading.Thread(target=self.alert,args=(user.addr,"DANGER",
datetime.datetime.now()))
    send_alert.start()
    self.point_alerts+=1
    return
    maliciousness=user.threat_score #get previous value
    compare=user.quantiles[hour]
    flow_gqs_lvl2=compare[1][0]
    flow_gqs_lvl3=compare[1][1]
    flow_gqs_lvl4=compare[1][2]
    call_gqs_lvl2=compare[2][0]
    call_gqs_lvl3=compare[2][1]
    call_gqs_lvl4=compare[2][2]
    checker=user.baseline[hour]
    login_base=checker[0]
    flow_base=checker[1]
    call_base=checker[2]
    #print("\n")
    #print(user.addr)
    #print(compare)
    #print("\n")
    anomalies=0
    if check_flow==0:
        pass
    else:
        match=0
        flow_2=0
        flow_3=0
        flow_4=0
        flow_check_1=0
        try:
            if ((flow_base==0) or (flow_base==[])): #no data for
this type
                #print("NO FLOW BASELINE FOR THIS HOUR: %s" %
user.addr)
                anomalies+=1
                maliciousness+=5
        except:
            for row in flow_base['flow dest']:
                if check_flow[2]==row:
                    match=1
            if match==0:
                anomalies+=1
                flow_check_1=1
                maliciousness+=1
            #####
            #must add ability to account for low flow counts at
start of the hour
            #####
            try:
                if flow_gqs_lvl2[0]==0:
                    anomalies+=1
                    maliciousness+=3

```

```

        except:
            if ((check_pkt >= flow_gqs_lvl2['cnt'][0]) and
(check_pkt <= flow_gqs_lvl2['cnt'][1])):
                #within 80% of density, potential concern
                if ((check_pkt >= flow_gqs_lvl3['cnt'][0]) and
(check_pkt <= flow_gqs_lvl3['cnt'][1])):
                    #within 70% of density, potential concern
                    if ((check_pkt >= flow_gqs_lvl4['cnt'][0])
and (check_pkt <= flow_gqs_lvl4['cnt'][1])):
                        #within 60% of density, no great
concern

                        pass
                    else:
                        #between 70% and 60% density, concern
                        #maliciousness+=2
                        #pass
                        flow_4=1
                else:
                    #between 80% and 70% density, concern
                    #anomalies+=1
                    #flow_got=1
                    #maliciousness+=1
                    flow_3=1
            else: #concerning
                #print("ALERT GENERATED FOR USER %s: User
packet count outside extreme normal range for %s:%d " %
(target[0],days[day],hour))
                #maliciousness+=5
                #anomalies+=1
                flow_2=1
while True:
    if flow_2==0:
        #within gqs_2
        if flow_3==0:
            #within gqs_3
            if flow_4==0:
                #within gqs_4
                break
            else:
                #outside gqs_4, within gqs_3
                maliciousness+=1
                break
        else:
            #outside gqs_3, within gqs_2
            maliciousness+=2
            break
    else:
        #outside gqs_2
        maliciousness+=5
        if flow_check_1==0:
            anomalies+=1
            flow_check_1=1
            break
        else:
            break

```

```

#####
#must add ability to account for low flow counts at start
of the hour
#####
    flow_4=0
    flow_3=0
    flow_2=0
    if ((check_size >= flow_gqs_lvl2['size'][0]) and
(check_size <= flow_gqs_lvl2['size'][1])):
        #within 80% of density, potential concern
        if ((check_size >= flow_gqs_lvl3['size'][0]) and
(check_size <= flow_gqs_lvl3['size'][1])):
            #within 70% of density, potential concern
            if ((check_size >= flow_gqs_lvl4['size'][0])
and (check_size <= flow_gqs_lvl4['size'][1])):
                #within 60% of density, no great concern
                pass
            else:
                #between 70% and 60% density, concern
                #maliciousness+=2
                flow_4=1
        else:
            flow_3=1
    else: #concerning
        flow_2=1
    while True:
        if flow_2==0:
            if flow_3==0:
                if flow_4==0:
                    break
                else:
                    maliciousness+=1
                    break
            else:
                maliciousness+=2
                break
        else:
            maliciousness+=5
            if flow_check_1==1:
                break
            else:
                anomalies+=1
                break
    if check_call==0:
        pass
    else:
        match=0
        call_check_1=0
        try:
            if ((call_base==0) or (call_base==[])):
                #print("NO CALL BASELINE DATA FOR THIS HOUR: %s" %
user.addr)
                maliciousness+=5
                anomalies+=1
        except:

```

```

call_4=0
call_3=0
call_2=0
for row in call_base['syscall']:
    if check_call==row:
        match=1
    if match==0:
        anomalies+=1
        maliciousness+=3
        call_check_1=1
#####
#must add ability to account for low flow counts at start
of the hour
#####
    if ((check_call_cnt >= call_gqs_lvl2[0]) and
(check_call_cnt <= call_gqs_lvl2[1])):
        #within 80% density
        if ((check_call_cnt >= call_gqs_lvl3[0]) and
(check_call_cnt <= call_gqs_lvl3[1])):
            #within 70% density
            if ((check_call_cnt >= call_gqs_lvl4[0]) and
(check_call_cnt <= call_gqs_lvl4[1])):
                #within 60% density
                pass
            else:
                #between 70% and 60% density, concern
                call_4=1
        else:
            call_3=1
    else: #concerning
        call_2=1
    while True:
        if call_2==0:
            if call_3==0:
                if call_4==0:
                    break
                else:
                    maliciousness+=1
                    break
            else:
                maliciousness+=2
                break
        else:
            maliciousness+=5
            if call_check_1==0:
                anomalies+=1
                break
            else:
                break

check_list=call_base.index[call_base['syscall']==check_call].tolist()
match=0
matcher=0
existing_calls=[]
for x in check_list:

```

```

        try:
            da=call_base.iloc[x]
            existing_calls=da[2]
            for test in check_call_param:
                for answer in existing_calls:
                    if test==answer:
                        match+=1
        except:
            #shenanigans
            pass
        matcher=len(existing_calls)
    if match!=matcher:
        if call_check_1==0:
            anomalies+=1
            maliciousness+=3
        else:
            maliciousness+=3
    #####
    #do cluster comparison
    #####
    user.threat_score+=maliciousness
    user.anomalies+=anomalies
    d_input=datetime.datetime.now()
    date_input=(("%d-%d-%d | %d:%d | Anomaly") %
(d_input.day,d_input.month,d_input.year,d_input.hour,d_input.minute))
    d_alert_input=(("%d-%d-%d | %d:%d | %s") %
(d_input.day,d_input.month,d_input.year,d_input.hour,d_input.minute,user
r.addr))
    if ((maliciousness >= self.threshold) and (maliciousness <
self.threshold*1.5)):

#print("*****")
        #print("CAUTION alert generated for: user - %s | threat
value - %d" % (user.addr,maliciousness))

#print("*****")

    send_alert=threading.Thread(target=self.alert,args=(user.addr,"CAUTION"
,datetime.datetime.now()))
        send_alert.start()
        self.anomaly_alerts+=1
    elif maliciousness >= self.threshold*1.5:

#print("*****")
        #print("DANGER alert generated for: user - %s | threat
value - %d" % (user.addr,maliciousness))

#print("*****")
        if date_input not in user.danger_alert:
            user.danger_alert.append(date_input)
            self.d_alerts.append(d_alert_input)

    send_alert=threading.Thread(target=self.alert,args=(user.addr,"DANGER",
datetime.datetime.now()))
        send_alert.start()

```

```

        self.context_alerts+=1

    def alert(self, HOFUND, category, timestamp):
        #send formatted alert to GJALLARHORN
        #wait for acknowledgement: if not received in 45s, retransmit
        sock=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        try:
            sock.connect((self.GJALLARHORN,self.GJALLARHORN_port))
        except Exception as e:
            #print(e)
            print("\nConnection issue sending alert to GJALLARHORN:
Server may not be listening at %s:%d" %
(self.GJALLARHORN,self.GJALLARHORN_port))
            alert_stop=False
            while alert_stop != True:
                try:
                    msg=[HOFUND,category,timestamp]
                    msg=pickle.dumps(msg)
                    sock.send(msg)
                    alert_stop=self.alert_rcv(sock)
                    if alert_stop!=True:
                        time.sleep(15)
                except (ConnectionResetError):
                    print("\nConnection issue sending alert to GJALLARHORN
against host %s: Server not accepting at port %d" %
(self.GJALLARHORN,HOFUND,self.GJALLARHORN_port))
                    return
            except Exception as e:
                print("\nUnknown Exception connecting to GJALLARHORN
against host %s: " % HOFUND)
                print(e)
                return
        return

    def alert_rcv(self, sock):
        try:
            result=sock.recv(1024).decode()
        except Exception as e:
            print("\nError receiving from GJALLARHORN: " + e)
            stop=False
            return stop
        if "STOP" in result: #GJALLARHORN received message
            stop=True
            sock.close()
            return stop

class USER:
    #data =
    [[usr_addr,usr_role],{(sport,dport,dest):[pkt,size],},{call:[cnt,params
],,}]
    def __init__(self,addr,role):
        self.addr=addr
        self.role=role

```

```

        self.data={}
        #{date:{hour:[{login:[time,],logout:[time,]}, {flow:[cnt,
        size],}, {call:[cnt,[param]],}],}}

self.baseline={0:[],1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[],10:[],
11:[],12:[],13:[],14:[],15:[],16:[],17:[],18:[],19:[],20:[],21:[],22:[],
23:[]}

self.quantiles={0:[],1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[],10:[],
11:[],12:[],13:[],14:[],15:[],16:[],17:[],18:[],19:[],20:[],21:[],22:[],
23:[]}

        self.cluster={}
        self.threat_score=0
        self.cluster_flow=[]
        self.cluster_call=[]
        self.cluster_log=[]
        self.danger_alert=[]
        self.size=0
        self.anomalies=0

def u_update(self,record,hour,day):
    #record =[flow,cnt,size,call,cnt,param]
    flow=record[0] #flow
    call=record[3] #call
    dow=self.data
    dat=datetime.date.today()
    date=("%d-%d-%d" % (dat.day,dat.month,dat.year))
    try:
        if dow[date]:
            pass
        if dow[date][hour]:
            pass
    except KeyError as e:
        try:
            if dow[date]:
                pass
        except:
            dow[date]={}
        try:
            if dow[date][hour]:
                pass
        except:
            dow[date][hour]=[[],[],[]]
            dow[date][hour][0]['login']=[]
            dow[date][hour][0]['logout']=[]
    if call=="login":
        try:
            dow[date][hour][0]['login'].append(record[5])
        except: #key error
            dow[date][hour][0]['login'].append(record[5])
    if call=='logout':
        try:
            dow[date][hour][0]['logout'].append(record[5])
        except: #key error
            dow[date][hour][0]['logout'].append(record[5])

```

```

        if flow!=0:
            try:
                dow[date][hour][1][flow][0]=record[1]
                dow[date][hour][1][flow][1]=record[2]
            except:
                dow[date][hour][1][flow]=[record[1],record[2]]
        if call!=0:
            try:
                dow[date][hour][2][call][0]=record[4]
                dow[date][hour][2][call][1]=record[5]
            except Exception as e:
                dow[date][hour][2][call]=[record[4],record[5]]

    def base(self, hourz, dayz):
        #return a dataframe for each day (check against quantile for
        #detection)
        #consider adding logic to support daily update vice weekly
        (only current day vice for dow in self.days)
        labels=['hour','login','logout','flow dest','flow count','flow
        size','syscall','call count','call params']
        log_labels=['login','logout']
        flow_labels=['flow dest','flow count','flow size']
        call_labels=['syscall','call count','call parameters']

        day_log={0:[],1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[],10:[],11:[],
        12:[],13:[],14:[],15:[],16:[],17:[],18:[],19:[],20:[],21:[],22:[],23:[]
        }

        day_flows={0:[],1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[],10:[],11:[
        ],12:[],13:[],14:[],15:[],16:[],17:[],18:[],19:[],20:[],21:[],22:[],23:[
        ]}

        day_calls={0:[],1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[],10:[],11:[
        ],12:[],13:[],14:[],15:[],16:[],17:[],18:[],19:[],20:[],21:[],22:[],23:[
        ]}

        day=self.data
        for date in day:
            fd=open("Log-%s.txt" % self.addr,'a+')
            fd.write("%s data found\n" % date)

        date_log={0:[],1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[],10:[],11:[
        ],12:[],13:[],14:[],15:[],16:[],17:[],18:[],19:[],20:[],21:[],22:[],23:[
        ]}

        date_flows={0:[],1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[],10:[],11:[
        ],12:[],13:[],14:[],15:[],16:[],17:[],18:[],19:[],20:[],21:[],22:[],23:[
        ]}

        date_calls={0:[],1:[],2:[],3:[],4:[],5:[],6:[],7:[],8:[],9:[],10:[],11:[
        ],12:[],13:[],14:[],15:[],16:[],17:[],18:[],19:[],20:[],21:[],22:[],23:[
        ]}

        for hour in self.data[date]: #0-23
            hour_log=[]
            hour_flows=[]
            hour_calls=[]

```

```

hour_log.append((self.data[date][hour][0]['login'],self.data[date][hour][0]['logout']))
    for flow in self.data[date][hour][1]:

hour_flows.append((flow[2],self.data[date][hour][1][flow][0],self.data[date][hour][1][flow][1])) # dest, cnt, size
    for call in self.data[date][hour][2]:

hour_calls.append((call,self.data[date][hour][2][call][0],self.data[date][hour][2][call][1])) #call, cnt, param

log_df=pd.DataFrame.from_records(hour_log,columns=log_labels)

flow_df=pd.DataFrame.from_records(hour_flows,columns=flow_labels)

call_df=pd.DataFrame.from_records(hour_calls,columns=call_labels)
    if ((hour_log[0][0]!=[]) and (hour_log[0][1]!=[])):
        lq1=log_df.quantile(0.15)
        lq2=log_df.quantile(0.85)
        liqr=lq2-lq1
        log_df_no_out=log_df[~((log_df < (lq1 - 1.5*liqr))
| (log_df > (lq2 + 1.5*liqr))).any(axis=1)]
        date_log[hour]=log_df_no_out
    else:
        date_log[hour]=0
    if (flow_df.empty):
        date_flows[hour]=flow_df #####THIS WAS =0
    else:
        hq1=flow_df.quantile(0.15)
        hq2=flow_df.quantile(0.85)
        hiqr=hq2-hq1
        flow_df_no_out=flow_df[~((flow_df[['flow
count','flow size']] < (hq1 - 1.5*hiqr)) | (flow_df[['flow count',
'flow size']] > (hq2 + 1.5*hiqr))).any(axis=1)]
        date_flows[hour]=flow_df_no_out
    if (call_df.empty):
        date_calls[hour]=call_df #####THIS WAS =0
    else:
        cq1=call_df.quantile(0.15)
        cq2=call_df.quantile(0.85)
        ciqr=cq2-cq1
        call_df_no_out=call_df[~((call_df[['call count']] <
(cq1 - 1.5*ciqr)) | (call_df[['call count']] > (cq2 +
1.5*ciqr))).any(axis=1)]
        date_calls[hour]=call_df_no_out
#####
#CHANGED THIS##### #WAS OUTSIDE THE LINE 843 IF
#####
try:
    if day_log[hour]==[]:
        day_log[hour]=date_log[hour]
    else:
        log_df1=day_log[hour]
        log_df2=date_log[hour]

```

```

        try:
            log_df3=pd.concat([log_df1,log_df2])
            day_log[hour]=log_df3
        except:
            pass
    except:
        day_log[hour]=date_log[hour]
    try:
        if day_flows[hour]==[]:
            day_flows[hour]=date_flows[hour]
        else:
            flow_df1=day_flows[hour]
            flow_df2=date_flows[hour]
            try:
                flow_df3=pd.concat([flow_df1,flow_df2])
                day_flows[hour]=flow_df3
            except Exception as e:
                print("broke data update ",e)
    except:
        day_flows[hour]=date_flows[hour]
    try:
        if day_calls[hour]==[]:
            day_calls[hour]=date_calls[hour]
        else:
            call_df1=day_calls[hour]
            call_df2=date_calls[hour]
            try:
                call_df3=pd.concat([call_df1,call_df2])
                day_calls[hour]=call_df3
            except Exception as e:
                print("broke data update",e)
    except:
        day_calls[hour]=date_calls[hour]
    #####
    #TO HERE#####
    #####
    with open("%s-%s" % (self.addr,date),"a+") as op:
        print(date,file=op)
        print(day_flows,file=op)
        print("\n",file=op)
        print(day_calls,file=op)
        print("\n",file=op)
    for hour in self.baseline:
        y=self.baseline[hour]
        if y == []: #[[[]],[[]],[[]]: #no existing data

self.baseline[hour]=[day_log[hour],day_flows[hour],day_calls[hour]]
        compare=self.baseline[hour]
        login_base=compare[0]
        flow_base=compare[1]
        call_base=compare[2]
        try: #login
            x=login_base['login']
            lnigr2=(login_base['login'].quantile(0.8)-
login_base['login'].quantile(.2))*1.5

```

```

        loiqr2=(login_base['logout'].quantile(0.8)-
login_base['logout'].quantile(.2))*1.5
        login_gqs_lvl2 =
{'login':[login_base['login'].quantile(0.2)-
lniqr2,login_base['login'].quantile(0.8)+lniqr2], 'logout':[login_base['
logout'].quantile(0.2)-
loiqr2,login_base['logout'].quantile(0.8)+loiqr2]}
        lniqr3=(login_base['login'].quantile(0.7)-
login_base['login'].quantile(.3))*1.5
        loiqr3=(login_base['logout'].quantile(0.7)-
login_base['logout'].quantile(.3))*1.5
        login_gqs_lvl3 =
{'login':[login_base['login'].quantile(0.3)-
lniqr3,login_base['login'].quantile(0.7)+lniqr3], 'logout':[login_base['
logout'].quantile(0.3)-
loiqr3,login_base['logout'].quantile(0.7)+loiqr3]}
        lniqr4=(login_base['login'].quantile(0.6)-
login_base['login'].quantile(.4))*1.5
        loiqr4=(login_base['logout'].quantile(0.6)-
login_base['logout'].quantile(.4))*1.5
        login_gqs_lvl4 =
{'login':[login_base['login'].quantile(0.4)-
lniqr4,login_base['login'].quantile(0.6)+lniqr4], 'logout':[login_base['
logout'].quantile(0.4)-
loiqr4,login_base['logout'].quantile(0.6)+loiqr4]}
        if login_gqs_lvl2['login'][0]<=0:
            login_gqs_lvl2['login'][0]=0
        if login_gqs_lvl2['logout'][0]<=0:
            login_gqs_lvl2['logout'][0]=0
        if login_gqs_lvl3['login'][0]<=0:
            login_gqs_lvl3['login'][0]=0
        if login_gqs_lvl3['logout'][0]<=0:
            login_gqs_lvl3['logout'][0]=0
        if login_gqs_lvl4['login'][0]<=0:
            login_gqs_lvl4['login'][0]=0
        if login_gqs_lvl4['logout'][0]<=0:
            login_gqs_lvl4['logout'][0]=0
        self.cluster_log.append(login_gqs_lvl4)
    except:
        login_gqs_lvl2={'login':[0,0], 'logout':[0,0]}
        login_gqs_lvl3={'login':[0,0], 'logout':[0,0]}
        login_gqs_lvl4={'login':[0,0], 'logout':[0,0]}
    try: #flow
        x=flow_base['flow count'] #dies here if no values
        cntiqr2=(flow_base['flow count'].quantile(0.8)-
flow_base['flow count'].quantile(.2))*1.5
        sizeiqr2=(flow_base['flow size'].quantile(0.8)-
flow_base['flow size'].quantile(.2))*1.5
        flow_gqs_lvl2 = {'cnt':[flow_base['flow
count'].quantile(0.2)-cntiqr2,flow_base['flow
count'].quantile(0.8)+cntiqr2], 'size':[flow_base['flow
size'].quantile(0.2)-sizeiqr2,flow_base['flow
size'].quantile(0.8)+sizeiqr2]}
        cntiqr3=(flow_base['flow count'].quantile(0.7)-
flow_base['flow count'].quantile(.3))*1.5

```

```

        sizeiqr3=(flow_base['flow size'].quantile(0.7)-
flow_base['flow size'].quantile(.3))*1.5
        flow_gqs_lvl3 = {'cnt':[flow_base['flow
count'].quantile(0.3)-cntiqr3,flow_base['flow
count'].quantile(0.7)+cntiqr3], 'size':[flow_base['flow
size'].quantile(0.3)-sizeiqr3,flow_base['flow
size'].quantile(0.7)+sizeiqr3]}
        cntiqr4=(flow_base['flow count'].quantile(0.6)-
flow_base['flow count'].quantile(.4))*1.5
        sizeiqr4=(flow_base['flow size'].quantile(0.6)-
flow_base['flow size'].quantile(.4))*1.5
        flow_gqs_lvl4 = {'cnt':[flow_base['flow
count'].quantile(0.4)-cntiqr4,flow_base['flow
count'].quantile(0.6)+cntiqr4], 'size':[flow_base['flow
size'].quantile(0.4)-sizeiqr4,flow_base['flow
size'].quantile(0.6)+sizeiqr4]}
        try:
            if np.isnan(flow_gqs_lvl4['size'][0]):
                flow_gqs_lvl2={'cnt':[0,0], 'size':[0,0]}
                flow_gqs_lvl3={'cnt':[0,0], 'size':[0,0]}
                flow_gqs_lvl4={'cnt':[0,0], 'size':[0,0]}
        except:
            pass
        if flow_gqs_lvl2['cnt'][0]<0:
            flow_gqs_lvl2['cnt'][0]=0
        if flow_gqs_lvl2['size'][0]<0:
            flow_gqs_lvl2['size'][0]=0
        if flow_gqs_lvl3['cnt'][0]<0:
            flow_gqs_lvl3['cnt'][0]=0
        if flow_gqs_lvl3['size'][0]<0:
            flow_gqs_lvl3['size'][0]=0
        if flow_gqs_lvl4['cnt'][0]<0:
            flow_gqs_lvl4['cnt'][0]=0
        if flow_gqs_lvl4['size'][0]<0:
            flow_gqs_lvl4['size'][0]=0
        self.cluster_flow.append(flow_gqs_lvl4)
    except Exception as e:
        flow_gqs_lvl2={'cnt':[0,0], 'size':[0,0]}
        flow_gqs_lvl3={'cnt':[0,0], 'size':[0,0]}
        flow_gqs_lvl4={'cnt':[0,0], 'size':[0,0]}
    try: #call
        x=call_base['call count']
        calliqr2=(call_base['call count'].quantile(0.8)-
call_base['call count'].quantile(.2))*1.5
        call_gqs_lvl2 = [call_base['call
count'].quantile(0.2)-calliqr2,call_base['call
count'].quantile(0.8)+calliqr2]
        calliqr3=(call_base['call count'].quantile(0.7)-
call_base['call count'].quantile(.3))*1.5
        call_gqs_lvl3 = [call_base['call
count'].quantile(0.3)-calliqr3,call_base['call
count'].quantile(0.7)+calliqr3]
        calliqr4=(call_base['call count'].quantile(0.6)-
call_base['call count'].quantile(.4))*1.5

```

```

        call_gqs_lvl4 = [call_base['call
count'].quantile(0.4)-callqr4,call_base['call
count'].quantile(0.6)+callqr4]
        #gg=tgt_day
        try:
            if np.isnan(call_gqs_lvl4[0]):
                call_gqs_lvl2=[0,0]
                call_gqs_lvl3=[0,0]
                call_gqs_lvl4=[0,0]
        except Exception as e:
            print("broke",e)
        if call_gqs_lvl2[0]<0:
            call_gqs_lvl2[0]=0
        if call_gqs_lvl3[0]<0:
            call_gqs_lvl3[0]=0
        if call_gqs_lvl4[0]<0:
            call_gqs_lvl4[0]=0
        except Exception as e:
            gg=0
            call_gqs_lvl2=[0,0]
            call_gqs_lvl3=[0,0]
            call_gqs_lvl4=[0,0]
        try:

self.quantiles[hour]=[login_gqs_lvl2,login_gqs_lvl3,login_gqs_lvl4],[f
low_gqs_lvl2,flow_gqs_lvl3,flow_gqs_lvl4],[call_gqs_lvl2,call_gqs_lvl3,
call_gqs_lvl4]]

        #diagnostic
        """try:
            if gg:
                if hour==16:
                    print(self.quantiles[hour])
                    print()

        except:
            pass"""
        except Exception as e:
            print("GQS Generation Issue initial")
            print(e)
    else: #existing data
        try: ##### LOGIN/LOGOUT
            if ((y[0] ==[]) or (y[0]==0)):
                if day_log[hour]==[]:
                    day_log[hour][0] #break it here so data
isn't lost

                self.baseline[hour][0]=day_log[hour]
                try:
                    login_base=day_log[hour]
                    x=login_base['login']
                    lnqr2=(login_base['login'].quantile(0.8)-
login_base['login'].quantile(.2))*1.5
                    loiqr2=(login_base['logout'].quantile(0.8)-
login_base['logout'].quantile(.2))*1.5
                    login_gqs_lvl2 =
{'login':[login_base['login'].quantile(0.2)-
lnqr2,login_base['login'].quantile(0.8)+lnqr2],'logout':[login_base['

```

```

logout'].quantile(0.2)-
loiqr2, login_base['logout'].quantile(0.8)+loiqr2]]
        lniqr3=(login_base['login'].quantile(0.7)-
login_base['login'].quantile(.3))*1.5
        loiqr3=(login_base['logout'].quantile(0.7)-
login_base['logout'].quantile(.3))*1.5
        login_gqs_lvl3 =
{'login':[login_base['login'].quantile(0.3)-
lniqr3, login_base['login'].quantile(0.7)+lniqr3], 'logout':[login_base['
logout'].quantile(0.3)-
loiqr3, login_base['logout'].quantile(0.7)+loiqr3]]
        lniqr4=(login_base['login'].quantile(0.6)-
login_base['login'].quantile(.4))*1.5
        loiqr4=(login_base['logout'].quantile(0.6)-
login_base['logout'].quantile(.4))*1.5
        login_gqs_lvl4 =
{'login':[login_base['login'].quantile(0.4)-
lniqr4, login_base['login'].quantile(0.6)+lniqr4], 'logout':[login_base['
logout'].quantile(0.4)-
loiqr4, login_base['logout'].quantile(0.6)+loiqr4]]
        except:

login_gqs_lvl2={'login':[0,0], 'logout':[0,0]}

login_gqs_lvl3={'login':[0,0], 'logout':[0,0]}

login_gqs_lvl4={'login':[0,0], 'logout':[0,0]}
        except:
            #####
            #combine with previous baselines 85/15
            #update baseline for hour
            #####
            try:
                compare=self.baseline[hour]
                login_base=compare[0]
                x=login_base['login']
                lniqr2=(login_base['login'].quantile(0.8)-
login_base['login'].quantile(.2))*1.5
                loiqr2=(login_base['logout'].quantile(0.8)-
login_base['logout'].quantile(.2))*1.5
                login_gqs_lvl2 =
{'login':[login_base['login'].quantile(0.2)-
lniqr2, login_base['login'].quantile(0.8)+lniqr2], 'logout':[login_base['
logout'].quantile(0.2)-
loiqr2, login_base['logout'].quantile(0.8)+loiqr2]]
                lniqr3=(login_base['login'].quantile(0.7)-
login_base['login'].quantile(.3))*1.5
                loiqr3=(login_base['logout'].quantile(0.7)-
login_base['logout'].quantile(.3))*1.5
                login_gqs_lvl3 =
{'login':[login_base['login'].quantile(0.3)-
lniqr3, login_base['login'].quantile(0.7)+lniqr3], 'logout':[login_base['
logout'].quantile(0.3)-
loiqr3, login_base['logout'].quantile(0.7)+loiqr3]]

```

```

        lnqr4=(login_base['login'].quantile(0.6)-
login_base['login'].quantile(.4))*1.5
        loqr4=(login_base['logout'].quantile(0.6)-
login_base['logout'].quantile(.4))*1.5
        login_gqs_lvl4 =
{'login':[login_base['login'].quantile(0.4)-
lnqr4,login_base['login'].quantile(0.6)+lnqr4], 'logout':[login_base['
logout'].quantile(0.4)-
loqr4,login_base['logout'].quantile(0.6)+loqr4]}
        mix_log_df=day_flows[hour]
        mix_lnqr2=(mix_log_df['login'].quantile(0.8)-
mix_log_df['login'].quantile(.2))*1.5
        mix_loqr2=(mix_log_df['logout'].quantile(0.8)-
mix_log_df['logout'].quantile(.2))*1.5
        mix_gqs_lvl2 =
{'login':[mix_log_df['login'].quantile(0.2)-
mix_lnqr2,mix_log_df['login'].quantile(0.8)+mix_lnqr2], 'logout':[mix_
log_df['logout'].quantile(0.2)-
mix_loqr2,mix_log_df['logout'].quantile(0.8)+mix_loqr2]}
        mix_lnqr3=(mix_log_df['login'].quantile(0.7)-
mix_log_df['login'].quantile(.3))*1.5
        mix_loqr3=(mix_log_df['logout'].quantile(0.7)-
mix_log_df['logout'].quantile(.3))*1.5
        mix_gqs_lvl3 =
{'login':[mix_log_df['login'].quantile(0.3)-
mix_lnqr3,mix_log_df['login'].quantile(0.7)+mix_lnqr3], 'logout':[mix_
log_df['logout'].quantile(0.3)-
mix_loqr3,mix_log_df['logout'].quantile(0.7)+mix_loqr3]}
        mix_lnqr4=(mix_log_df['login'].quantile(0.6)-
mix_log_df['login'].quantile(.4))*1.5
        mix_loqr4=(mix_log_df['logout'].quantile(0.6)-
mix_log_df['logout'].quantile(.4))*1.5
        mix_gqs_lvl4 =
{'login':[mix_log_df['login'].quantile(0.4)-
mix_lnqr4,mix_log_df['login'].quantile(0.6)+mix_lnqr4], 'logout':[mix_
log_df['logout'].quantile(0.4)-
mix_loqr4,mix_log_df['logout'].quantile(0.6)+mix_loqr4]}
        index=0
        for event,mix_event in
login_gqs_lvl2,mix_gqs_lvl2:
            index=0
            for x,y in event,mix_event:
                if x==y:
                    index+=1
                pass
            else:
                x=x*.85
                y=y*.15
                z=x+y
                login_gqs_lvl2[event][index]=z
                index+=1
        index=0
        for event,mix_event in
login_gqs_lvl3,mix_gqs_lvl3:
            index=0

```

```

        for x,y in event,mix_event:
            if x==y:
                index+=1
            pass
        else:
            x=x*.85
            y=y*.15
            z=x+y
            login_gqs_lvl3[event][index]=z
            index+=1
    index=0
    for event,mix_event in
login_gqs_lvl4,mix_gqs_lvl4:
        index=0
        for x,y in event,mix_event:
            if x==y:
                index+=1
            pass
        else:
            x=x*.85
            y=y*.15
            z=x+y
            login_gqs_lvl4[event][index]=z
            index+=1
    except Exception as e:
        login_gqs_lvl2={'login':[0,0], 'logout':[0,0]}
        login_gqs_lvl3={'login':[0,0], 'logout':[0,0]}
        login_gqs_lvl4={'login':[0,0], 'logout':[0,0]}
    try: ##### FLOW
        if ((y[1] ==[]) or (y[1]==0)):
            if day_flows[hour]==0:
                day_flows[hour][1]#break it here
            self.baseline[hour][1]=day_flows[hour]
            try:
                compare=self.baseline[hour]
                flow_base=compare[1]
                x=flow_base['flow count'] #dies here if no
values
                cntiqr2=(flow_base['flow
count'].quantile(0.8)-flow_base['flow count'].quantile(.2))*1.5
                sizeiqr2=(flow_base['flow
size'].quantile(0.8)-flow_base['flow size'].quantile(.2))*1.5
                flow_gqs_lvl2 = {'cnt':[flow_base['flow
count'].quantile(0.2)-cntiqr2,flow_base['flow
count'].quantile(0.8)+cntiqr2], 'size':[flow_base['flow
size'].quantile(0.2)-sizeiqr2,flow_base['flow
size'].quantile(0.8)+sizeiqr2]}
                cntiqr3=(flow_base['flow
count'].quantile(0.7)-flow_base['flow count'].quantile(.3))*1.5
                sizeiqr3=(flow_base['flow
size'].quantile(0.7)-flow_base['flow size'].quantile(.3))*1.5
                flow_gqs_lvl3 = {'cnt':[flow_base['flow
count'].quantile(0.3)-cntiqr3,flow_base['flow
count'].quantile(0.7)+cntiqr3], 'size':[flow_base['flow

```

```

size'].quantile(0.3)-sizeiqr3,flow_base['flow
size'].quantile(0.7)+sizeiqr3]]
        cntiqr4=(flow_base['flow
count'].quantile(0.6)-flow_base['flow count'].quantile(.4))*1.5
        sizeiqr4=(flow_base['flow
size'].quantile(0.6)-flow_base['flow size'].quantile(.4))*1.5
        flow_gqs_lvl4 = {'cnt':[flow_base['flow
count'].quantile(0.4)-cntiqr4,flow_base['flow
count'].quantile(0.6)+cntiqr4], 'size':[flow_base['flow
size'].quantile(0.4)-sizeiqr4,flow_base['flow
size'].quantile(0.6)+sizeiqr4]]
        if np.isnan(flow_gqs_lvl4['cnt'][0]):

flow_gqs_lvl2={'cnt':[0,0], 'size':[0,0]}

flow_gqs_lvl3={'cnt':[0,0], 'size':[0,0]}

flow_gqs_lvl4={'cnt':[0,0], 'size':[0,0]}
        except:
            flow_gqs_lvl2={'cnt':[0,0], 'size':[0,0]}
            flow_gqs_lvl3={'cnt':[0,0], 'size':[0,0]}
            flow_gqs_lvl4={'cnt':[0,0], 'size':[0,0]}
    except:
        #####
        #combine with previous baselines 85/15
        #update baseline for hour
        #####
        try:
            try:
                compare=self.quantiles[hour]
                if compare[1]==0:
                    compare=self.baseline[hour]
            except:
                compare=self.quantiles[hour]

x=self.baseline[hour][1].append(day_flows[hour])
        self.baseline[hour][1]=x
        flow_base=compare[1]
        mix_flow_df=day_flows[hour]
        try:
            try:
                mix_flow_df['flow count']
                mix_cntiqr2=(mix_flow_df['flow
count'].quantile(0.8)-mix_flow_df['flow count'].quantile(.2))*1.5
                mix_sizeiqr2=(mix_flow_df['flow
size'].quantile(0.8)-mix_flow_df['flow size'].quantile(.2))*1.5
                mix_gqs_lvl2 =
{'cnt':[mix_flow_df['flow count'].quantile(0.2)-
mix_cntiqr2,mix_flow_df['flow
count'].quantile(0.8)+mix_cntiqr2], 'size':[mix_flow_df['flow
size'].quantile(0.2)-mix_sizeiqr2,mix_flow_df['flow
size'].quantile(0.8)+mix_sizeiqr2]}
                mix_cntiqr3=(mix_flow_df['flow
count'].quantile(0.7)-mix_flow_df['flow count'].quantile(.3))*1.5

```

```

        mix_sizeiqr3=(mix_flow_df['flow
size'].quantile(0.7)-mix_flow_df['flow size'].quantile(.3))*1.5
        mix_gqs_lvl3 =
{'cnt':[mix_flow_df['flow count'].quantile(0.3)-
mix_cntiqr3,mix_flow_df['flow
count'].quantile(0.7)+mix_cntiqr3], 'size':[mix_flow_df['flow
size'].quantile(0.3)-mix_sizeiqr3,mix_flow_df['flow
size'].quantile(0.7)+mix_sizeiqr3]}
        mix_cntiqr4=(mix_flow_df['flow
count'].quantile(0.6)-mix_flow_df['flow count'].quantile(.4))*1.5
        mix_sizeiqr4=(mix_flow_df['flow
size'].quantile(0.6)-mix_flow_df['flow size'].quantile(.4))*1.5
        mix_gqs_lvl4 =
{'cnt':[mix_flow_df['flow count'].quantile(0.4)-
mix_cntiqr4,mix_flow_df['flow
count'].quantile(0.6)+mix_cntiqr4], 'size':[mix_flow_df['flow
size'].quantile(0.4)-mix_sizeiqr4,mix_flow_df['flow
size'].quantile(0.6)+mix_sizeiqr4]}

x=self.baseline[hour][1].append(day_flows[hour])
self.baseline[hour][1]=x
except:
    # empty input
    print("flow mix",e)
    mix_gqs_lvl2={'cnt':[0,0], 'size':[0,0]}
    mix_gqs_lvl3={'cnt':[0,0], 'size':[0,0]}
    mix_gqs_lvl4={'cnt':[0,0], 'size':[0,0]}
    flow_gqs_lvl2=flow_base[0]
    flow_gqs_lvl3=flow_base[1]
    flow_gqs_lvl4=flow_base[2]
    for event in flow_gqs_lvl2:
        for mix_event in mix_gqs_lvl2:
            if event==mix_event:
                gqs_values=flow_gqs_lvl2[event]

mix_values=mix_gqs_lvl2[mix_event]

        low_gqs=gqs_values[0]
        high_gqs=gqs_values[1]
        low_mix=mix_values[0]
        high_mix=mix_values[1]
        if low_gqs==low_mix:
            low=low_gqs
            if low<0:
                low=0
        else:
            low_gqs=low_gqs*.85
            low_mix=low_mix*.15
            low=low_gqs+low_mix
            if low<0:
                low=0
        if high_gqs==high_mix:
            high=high_gqs
        else:
            high_gqs=high_gqs*.85
            high_mix=high_mix*.15

```

```

        high=high_gqs+high_mix
        flow_gqs_lvl2[event]=[low,high]
    else:
        pass
index=0
for event in flow_gqs_lvl3:
    for mix_event in mix_gqs_lvl3:
        if event==mix_event:
            gqs_values=flow_gqs_lvl3[event]

mix_values=mix_gqs_lvl3[mix_event]

    low_gqs=gqs_values[0]
    high_gqs=gqs_values[1]
    low_mix=mix_values[0]
    high_mix=mix_values[1]
    if low_gqs==low_mix:
        low=low_gqs
        if low<0:
            low=0
    else:
        low_gqs=low_gqs*.85
        low_mix=low_mix*.15
        low=low_gqs+low_mix
        if low<0:
            low=0
    if high_gqs==high_mix:
        high=high_gqs
    else:
        high_gqs=high_gqs*.85
        high_mix=high_mix*.15
        high=high_gqs+high_mix
    flow_gqs_lvl3[event]=[low,high]
else:
    pass
index=0
for event in flow_gqs_lvl4:
    for mix_event in mix_gqs_lvl4:
        if event==mix_event:
            gqs_values=flow_gqs_lvl4[event]

mix_values=mix_gqs_lvl4[mix_event]

    low_gqs=gqs_values[0]
    high_gqs=gqs_values[1]
    low_mix=mix_values[0]
    high_mix=mix_values[1]
    if low_gqs==low_mix:
        low=low_gqs
        if low<0:
            low=0
    else:
        low_gqs=low_gqs*.85
        low_mix=low_mix*.15
        low=low_gqs+low_mix
        if low<0:
            low=0

```

```

        if high_gqs==high_mix:
            high=high_gqs
        else:
            high_gqs=high_gqs*.85
            high_mix=high_mix*.15
            high=high_gqs+high_mix
            flow_gqs_lvl4[event]=[low,high]
    else:
        pass

except:
    pass
    if np.isnan(flow_gqs_lvl4['cnt'][0]):
        flow_gqs_lvl2={'cnt':[0,0], 'size':[0,0]}
        flow_gqs_lvl3={'cnt':[0,0], 'size':[0,0]}
        flow_gqs_lvl4={'cnt':[0,0], 'size':[0,0]}
except Exception as e:
    print('flow update',e)
    flow_gqs_lvl2={'cnt':[0,0], 'size':[0,0]}
    flow_gqs_lvl3={'cnt':[0,0], 'size':[0,0]}
    flow_gqs_lvl4={'cnt':[0,0], 'size':[0,0]}
try: ##### CALLS
    #y[2]=self.baseline[b_day][hour] calls
    if ((y[2]==[]) or (y[2]==0)): #nothing there
        if day_calls[hour]==[]: #also nothing
            day_calls[hour][2] #break here and all 0s
        self.baseline[hour][2]=day_calls[hour]
    try:
        compare=self.baseline[hour]
        call_base=compare[2]
        x=call_base['call count']
        callqr2=(call_base['call
count'].quantile(0.8)-call_base['call count'].quantile(.2))*1.5
        call_gqs_lvl2 = [call_base['call
count'].quantile(0.2)-callqr2,call_base['call
count'].quantile(0.8)+callqr2]
        callqr3=(call_base['call
count'].quantile(0.7)-call_base['call count'].quantile(.3))*1.5
        call_gqs_lvl3 = [call_base['call
count'].quantile(0.3)-callqr3,call_base['call
count'].quantile(0.7)+callqr3]
        callqr4=(call_base['call
count'].quantile(0.6)-call_base['call count'].quantile(.4))*1.5
        call_gqs_lvl4 = [call_base['call
count'].quantile(0.4)-callqr4,call_base['call
count'].quantile(0.6)+callqr4]
        if np.isnan(call_gqs_lvl4[0]):
            call_gqs_lvl2=[0,0]
            call_gqs_lvl3=[0,0]
            call_gqs_lvl4=[0,0]
        print(call_gqs_lvl4)
    except:
        print("broken call update",e)
        call_gqs_lvl2=[0,0]
        call_gqs_lvl3=[0,0]
        call_gqs_lvl4=[0,0]

```

```

except:
    #####
    #combine with previous baselines 85/15
    #update baseline for hour
    #####
    try:
        try:
            compare=self.quantiles[hour]
            if compare[2]==0:
                compare=self.baseline[hour]
        except:
            compare=self.quantiles[hour]

x=self.baseline[hour][2].append(day_calls[hour])
    self.baseline[hour][2]=x
    call_base=compare[2]
    mix_call_df=day_calls[hour]
    try:
        try:
            mix_call_df['call count']
            mixiqr2=(mix_call_df['call
count'].quantile(0.8)-mix_call_df['call count'].quantile(.2))*1.5
            mix_gqs_lvl2 = [mix_call_df['call
count'].quantile(0.2)-mixiqr2,mix_call_df['call
count'].quantile(0.8)+mixiqr2]
            mixiqr3=(mix_call_df['call
count'].quantile(0.7)-mix_call_df['call count'].quantile(.3))*1.5
            mix_gqs_lvl3 = [mix_call_df['call
count'].quantile(0.3)-mixiqr3,mix_call_df['call
count'].quantile(0.7)+mixiqr3]
            mixiqr4=(mix_call_df['call
count'].quantile(0.6)-mix_call_df['call count'].quantile(.4))*1.5
            mix_gqs_lvl4 = [mix_call_df['call
count'].quantile(0.4)-mixiqr4,mix_call_df['call
count'].quantile(0.6)+mixiqr4]

x=self.baseline[hour][2].append(day_calls[hour])
    self.baseline[hour][2]=x
    except:
        mix_gqs_lvl2=[0,0]
        mix_gqs_lvl3=[0,0]
        mix_gqs_lvl4=[0,0]
        call_gqs_lvl2=call_base[0]
        call_gqs_lvl3=call_base[1]
        call_gqs_lvl4=call_base[2]
        if call_gqs_lvl2[1]==0:
            if mix_gqs_lvl2[0]==0:
                call_gqs_lvl2=[0,0]
        else:
            call_low=call_gqs_lvl2[0]*.85
            call_high=call_gqs_lvl2[1]*.85
            mix_low=mix_gqs_lvl2[0]*.15
            mix_high=mix_gqs_lvl2[1]*.15
            if call_gqs_lvl2[0]==mix_gqs_lvl2[0]:
                low=call_gqs_lvl2[0]

```

```

else:
    low=call_low+mix_low
    if call_gqs_lvl2[1]==mix_gqs_lvl2[1]:
        high=call_gqs_lvl2[1]
    else:
        high=call_high+mix_high
    if low<0:
        low=0
    call_gqs_lvl2=[low,high]
if call_gqs_lvl3[1]==0:
    if mix_gqs_lvl3[0]==0:
        call_gqs_lvl3=[0,0]
else:
    call_low=call_gqs_lvl3[0]*.85
    call_high=call_gqs_lvl3[1]*.85
    mix_low=mix_gqs_lvl3[0]*.15
    mix_high=mix_gqs_lvl3[1]*.15
    if call_gqs_lvl3[0]==mix_gqs_lvl3[0]:
        low=call_gqs_lvl3[0]
    else:
        low=call_low+mix_low
    if call_gqs_lvl3[1]==mix_gqs_lvl3[1]:
        high=call_gqs_lvl3[1]
    else:
        high=call_high+mix_high
    if low<0:
        low=0
    call_gqs_lvl3=[low,high]
if call_gqs_lvl4[1]==0:
    if mix_gqs_lvl4[0]==0:
        call_gqs_lvl4=[0,0]
else:
    call_low=call_gqs_lvl4[0]*.85
    call_high=call_gqs_lvl4[1]*.85
    mix_low=mix_gqs_lvl4[0]*.15
    mix_high=mix_gqs_lvl4[1]*.15
    if call_gqs_lvl4[0]==mix_gqs_lvl4[0]:
        low=call_gqs_lvl4[0]
    else:
        low=call_low+mix_low
    if call_gqs_lvl4[1]==mix_gqs_lvl4[1]:
        high=call_gqs_lvl4[1]
    else:
        high=call_high+mix_high
    if low<0:
        low=0
    call_gqs_lvl4=[low,high]
if np.isnan(call_gqs_lvl4[0]):
    call_gqs_lvl2=[0,0]
    call_gqs_lvl3=[0,0]
    call_gqs_lvl4=[0,0]
except Exception as e:
    pass
except Exception as e:
    print('call update',e)

```

```

        #something bad happened, preserve old bls?
        call_gqs_lvl2=[0,0]
        call_gqs_lvl3=[0,0]
        call_gqs_lvl4=[0,0]

    try:

self.quantiles[hour]=[[login_gqs_lvl2, login_gqs_lvl3, login_gqs_lvl4], [f
low_gqs_lvl2, flow_gqs_lvl3, flow_gqs_lvl4], [call_gqs_lvl2, call_gqs_lvl3,
call_gqs_lvl4]]

        except Exception as e:
            print("GQS Generation Issue after",e)

            day_log=0
            day_flows=0
            day_calls=0
            date_log=0
            date_flows=0
            date_calls=0
            hour_log=0
            hour_flows=0
            hour_calls=0
            #print(self.quantiles)
            with open("%s" % self.addr,"a+") as fi:
                dt=datetime.datetime.now()
                try:
                    print("\nData for %d-%d-%d\nBaseline: " %
(dt.day,dt.month,dt.year),file=fi)
                    print(self.baseline, file=fi)
                    print("Quantile: ",file=fi)
                    print(self.quantiles, file=fi)
                    print("Days: ",file=fi)
                    print(self.data,file=fi)
                except:
                    print("All fucked %s" % self.addr)
                    pass
            #get size of self
            self.size=0
            try:
                for date in self.data:
                    self.size+=sum(sys.getsizeof(hour) for hour in
self.data[date].keys())
                    self.size+=sum(sys.getsizeof(val) for val in
self.data[date].values())
            except Exception as e:
                print("Issue getting days size:")
                print(e)
            try:
                self.size+=sum(sys.getsizeof(hour) for hour in
self.baseline.keys())
                self.size+=sum(sys.getsizeof(val) for val in
self.baseline.values())
            except Exception as e:
                print("Issue getting baseline size: ")
                print(e)
            try:

```

```

        self.size+=sum(sys.getsizeof(hour) for hour in
self.quantiles.keys())
        self.size+=sum(sys.getsizeof(val) for val in
self.quantiles.values())
        self.size+=sys.getsizeof(self.cluster_flow)
        self.size+=sys.getsizeof(self.cluster_log)
    except Exception as e:
        print("Issue getting quantile size or cluster input size:
")
        print(e)
    self.clear()

def clear(self):
    #self.monday.clear()
    #self.tuesday.clear()
    #self.wednesday.clear()
    #self.thursday.clear()
    #self.friday.clear()
    #self.saturday.clear()
    #self.sunday.clear()
    #self.threat_score=0
    self.data.clear()

```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] E. Pilkington, “Bradley Manning verdict: Cleared of ‘aiding the enemy’ but guilty of other charges,” *The Guardian*, Jul. 30, 2013. [Online]. Available: <https://www.theguardian.com/world/2013/jul/30/bradley-manning-wikileaks-judge-verdict>
- [2] United States District Court for the Southern District of Georgia, “Application for search warrant,” Jun. 3, 2017. [Online]. Available: <https://www.documentcloud.org/documents/3848398-Reality-Winner-Affidavit-for-Application-of.html#document/p2/a356449>
- [3] P. Finn and S. Horwitz, “U.S. charges Snowden with espionage,” *The Washington Post*, Jun. 21, 2013. [Online]. Available: https://www.washingtonpost.com/world/national-security/us-charges-snowden-with-espionage/2013/06/21/507497d8-dab1-11e2-a016-92547bf094cc_story.html?utm_term=.3deb5d3b7d2a
- [4] E. Cole and S. Ring, *Insider Threat: Protecting the Enterprise from Sabotage, Spying and Theft*, Rockland, MA, USA: Syngress Publishing, 2005. [Online]. Available: <https://epdf.tips/queue/insider-threat-protecting-the-enterprise-from-sabotage-spying-and-theft.html>
- [5] S. Pfleeger, J. Predd, J. Hunker and C. Bulford, “Insiders behaving badly: Addressing bad actors and their actions,” *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 1, pp. 169-179, April 2010. [Online]. Available: https://www.researchgate.net/publication/224096411_Insiders_Behaving_Badly_Addressing_Bad_Actors_and_Their_Actions
- [6] P. Legg, N. Moffat, J. Nurse, J. Happa, I. Agraftotis, M. Goldsmith, and S. Creese, “Towards a conceptual model and reasoning structure for insider threat detection,” *Journal of Wireless Networks, Ubiquitous Computing, and Dependable Applications*, vol. 4, no. 4, pp. 20–37, 2013. [Online]. Available: http://www.cems.uwe.ac.uk/~pa-legg/research/resources/my_papers/2013_mist.pdf
- [7] Cybersecurity Insiders, “Insider threat 2018 report,” CA Technologies. Accessed Jun. 20, 2018. [Online]. Available: <https://www.ca.com/content/dam/ca/us/files/ebook/insider-threat-report.pdf>
- [8] *The DoD Insider Threat Program*, DODDIR 5205.16, United States Department of Defense, Washington, DC, USA, 2017. [Online]. Available: https://fas.org/irp/DODdir/DoD/d5205_16.pdf

- [9] M. Collins, M. Theis, R. Trzeciak, J. Strozer, J. Clark, D. Costa, T. Cassidy, M. Albrethsen and A. Moore, *Common Sense Guide to Prevention and Detection of Insider Threats*, 5th ed. Pittsburgh, PA, USA: Carnegie Mellon University, 2016. [Online]. Available: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2016_005_001_484758.pdf
- [10] I. Homoliak, F. Toffalini, J. Guarnizo, Y. Elovici, and M. Ochoa, "Insight into insiders: A survey of insider threat taxonomies, analysis, modeling and countermeasures," *ACM Computing Surveys*, vol. 99, no. 99 pp 1-49, 4 May 2018. [Online]. Available: <https://arxiv.org/pdf/1805.01612.pdf>
- [11] CERT Insider Threat Team, *Unintentional Insider Threats: A Review of Phishing and Malware Incidents by Economic Sector*, Pittsburgh, PA, USA: Carnegie Mellon University, 2014. [Online]. Available: https://resources.sei.cmu.edu/asset_files/TechnicalNote/2014_004_001_297777.pdf
- [12] M. Maloof and G. Stephens, "ELICIT: A system for detecting insiders who violate need-to-know," in *Recent Advances in Intrusion Detection 10th Annual Symposium, RAID 2007, Gold Coast, Australia, September 5-7, 2007*, C. Kruegel, R. Lippmann and A. Clark, Eds. Springer-Verlag Berlin Heidelberg, 2007. pp. 146-166. [Online]. Available: <http://people.cs.georgetown.edu/~maloo/pubs/maloof-raid07.pdf>
- [13] M. Maloof and G. Stephens, "Insider threat detection," United States Patent 8,707,431, April 22, 2014.
- [14] Splunk, "Splunk Enterprise Documentation Portal," Accessed Sep 28, 2018. [Online]. Available: <https://docs.splunk.com/Documentation/Splunk>
- [15] J. Brewer, "Phone interview," 28 September 2018.
- [16] P. Legg, O. Buckley, M. Goldsmith and S. Creese, "Automated insider threat detection system using user and role-based profile assessment," *IEEE Systems Journal*, vol. 11, no. 2, June 2015. [Online]. Available: <https://ieeexplore.ieee.org/document/7126970>
- [17] J. Dean, "Systematic assessment of the impact of user roles on network flow patterns," PhD dissertation, Department of Computer Science, Naval Postgraduate School, Monterey CA, 2017. [Online]. Available: <https://calhoun.nps.edu/handle/10945/56119>
- [18] A. Shaghaghi, S. Kanhere, M. Kaafar, E. Bertino, S. Jha, "Gargoyle: A network-based insider attack resilient framework for organization," The University of New South Wales, Australia, July 7, 2018. [Online]. Available: <https://arxiv.org/pdf/1807.02593.pdf>

- [19] A. Liu, C. Martin, T. Hetherington, and S. Matzner, "A comparison of system call feature representations for insider threat detection," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, West Point, NY, USA, June 15-17 2005. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/1495972>
- [20] N. Kanaskar, J. bian, R. Seker, M. Nijim, and N. Yilmazer, "Dynamical system approach to insider threat detection," in *IEEE International Systems Conference, SysCon 2011, Montreal, Canada, April 3-6 2011*. [Online]. Available: https://www.researchgate.net/publication/252012458_Dynamical_System_approach_to_insider_threat_detection
- [21] B. Bowen, M. Salem, S. Hershkop, A. Keromytis and S. Stolfo, "Designing host and network sensors to mitigate the insider threat," *The IEEE Computer and Reliability Societies*, Dec. 2009. [Online]. Available: <http://www.cs.columbia.edu/~bmbowen/papers/DesigningSensorsInsider.pdf>
- [22] M. Bhuyan, D. Bhattacharyya and J. Kalita, "Network anomaly detection: methods, systems and tools," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, 2014. [Online]. Available: <https://ieeexplore.ieee.org/document/6524462>
- [23] G. Magklaras and S. Furnell, "Insider Threat Prediction Tool: Evaluating the probability of IT misuse," *Computers & Security*, vol. 21, no. 1, pp. 62-73, January 2002. [Online]. Available: https://www.researchgate.net/publication/220614801_Insider_Threat_Prediction_Tool_Evaluating_the_probability_of_IT_misuse [Accessed: June 20, 2018].
- [24] *Standards for Security Categorization of Federal Information and Information Systems*, National Institute of Standards and Technology Federal Information Processing Standard 199, 2004. [Online] Available: <https://csrc.nist.gov/publications/detail/fips/199/final>
- [25] *Guide for Applying the Risk Management Framework to Federal Information Systems: A Security Lifecycle Approach*, National Institute of Standards and Technology Special Publication 800-37, 2010. [Online] Available: <https://csrc.nist.gov/publications/detail/sp/800-37/rev-1/final>
- [26] S. Shane, "Ex-N.S.A. worker accused of stealing trove of secrets offers to plead guilty," *The New York Times*, Jan. 3, 2018. [Online]. Available: <https://www.nytimes.com/2018/01/03/us/politics/harold-martin-nsa-guilty-plea-offer.html>
- [27] Ponemon Institute, "2018 cost of insider threats: Global," Ponemon Institute, LLC. Accessed Sept. 12, 2018. [Online]. Available: <https://153j3ttjub71nfe89mc7r5gb-wpengine.netdna-ssl.com/wp-content/uploads/2018/04/ObserveIT-Insider-Threat-Global-Report-FINAL.pdf>

- [28] S. Sinclair and S. Smith, "Preventative directions for insider threat mitigation via access control," in *Insider Attack and Cyber Security*, S. Salvatore, S. Bellovin, A. Keromytis, S. HersHKop, S. Smith, and S. Sinclair, Eds. Boston: Springer, 2008. [Online]. Available: <https://pdfs.semanticscholar.org/04b4/fa5c57cf7ebf3a4afddc8fd19fbb52de0da9.pdf>
- [29] Hofstede, Rick; Čeleda, Pavel; Trammell, Brian; Drago, Idilio; Sadre, Ramin; Sperotto, Anna; Pras, Aiko, "Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037-2064, October-December 2014. [Online]. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6814316>
- [30] Free Software Foundation, "auditctl system utility manual page," *CentOS Linux (Version 7.6.1810)*, 2018. [Online]. Available: <http://man7.org/linux/man-pages/man8/auditctl.8.html>
- [31] J. Jabez, "Intrusion detection system (IDS): Anomaly detection using outlier detection approach," in *International Conference on Intelligent Computing, Communication & Convergence (ICCC-2015)*, Bhubaneswar, Odisha, India, December 27-28 2015. [Online]. Available: https://www.researchgate.net/publication/277935708_Intrusion_Detection_System_IDS_Anomaly_Detection_Using_Outlier_Detection_Approach
- [32] P. Chhabra, C. Scott, E. Kolaczyk, and M. Crovella, "Distributed spatial anomaly detection," in *IEEE INFOCOM 2008 – The 27th Conference on Computer Communications*, Phoenix, AZ, USA, April 13-18, 2008. [Online]. Available: http://math.bu.edu/people/kolaczyk/pubs/anomDetect_gqs_cameraReady.pdf
- [33] 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, LA, USA, January 7-9, 2007. [Online]. Available: <http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf>
- [34] R. Hyndman and Y. Fan, "Sample quantiles in statistical packages," *The American Statistician*, vol. 50, November 1996. [Online]. Available: https://www.researchgate.net/publication/222105754_Sample_Quantiles_in_Statistical_Packages
- [35] F. J. Anscombe, "Rejection of outliers," *Technometrics*, vol. 2, no. 2, pp. 123–147, 1960. [Online]. Available: <https://amstat.tandfonline.com/doi/pdf/10.1080/00401706.1960.10489888>
- [36] F. Idzikowski, F. Pfeuffer, A. Werner, and L. Chiaraviglio, "Spatial and temporal traffic variation in core networks: Impact on energy saving and devices lifetime," *Energies*, vol. 9, issue 10, Oct. 19, 2016. [Online]. Available: <https://www.mdpi.com/1996-1073/9/10/837/pdf>

- [37] *Cyberspace Operations*, Joint Publication 3-12, United States Department of Defense, Washington, DC, 2017. [Online]. Available: https://www.jcs.mil/Portals/36/Documents/Doctrine/pubs/jp3_12.pdf
- [38] M. Monahan, "Traffic dissector for Ruckus GRE," ITACS, Naval Postgraduate School, Monterey, CA, 2018.
- [39] S. Reissman, "PyTgen," GitHub, Dec. 17, 2012. [Online]. Available: <https://github.com/reissmann/PyTgen>
- [40] 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 *International Workshop on Managing Insider Security Threats (MIST@CCS)*, Dallas, TX, USA, October 30, 2017. [Online]. Available: <https://github.com/ivan-homoliak-sutd/twos>
- [41] CERT Insider Threat Team, "Insider threat test dataset," Carnegie Mellon University, 2016. [Online]. Available: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=508099>
- [42] L. Liu, O. De Vel, Q. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, Second Quarter 2018. [Online]. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8278157>
- [43] T. Lengyel, S. Maresca, B. Payne, G. Webster, S. Vogl, and A. Kiayias, "Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system," in *Proceedings of the 30th Annual Computer Security Applications Conference*, New Orleans, LA, USA, December 8-12, 2014. [Online]. Available: https://dl.acm.org/ft_gateway.cfm?id=2664252&ftid=1516652&dwn=1&CFID=117626601&CFTOKEN=d124ee011cc44616-5B8C08B1-F9EE-5CBB-A4CB6E67B05A1FF8
- [44] S. Pugnet, "Syswall: Work in progress firewall for Linux syscalls," GitHub, Feb. 3, 2019. [Online]. Available: <https://github.com/polaris64/syswall>

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