7-2014

Scalable Automation of Online Network Attack Characterization

Ryan T. Rawlins

Recommended Citation

Rawlins, Ryan T., "Scalable Automation of Online Network Attack Characterization" (2014). Thesis. Rochester Institute of Technology. Accessed from

Scalable Automation of Online Network Attack Characterization

by

Ryan T. Rawlins

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering

Supervised by

Associate Professor and Department Head Dr. Shanchieh Jay Yang
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
July 2014

Approved by:

Dr. Shanchieh Jay Yang, Associate Professor and Department Head
Thesis Advisor, Department of Computer Engineering

Dr. Andres Kwasinski, Associate Professor
Committee Member, Department of Computer Engineering

Dr. Raymond Ptucha, Assistant Professor
Committee Member, Department of Computer Engineering

Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title:

Scalable Automation of Online Network Attack Characterization

I, Ryan T. Rawlins, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Ryan T. Rawlins

Date

Acknowledgments

I would like to thank my thesis advisor and committee members for their guidance and time throughout the research process. I would also like to thank my friends and family for their support.

Abstract

Scalable Automation of Online Network Attack Characterization

Ryan T. Rawlins

Supervising Professor: Dr. Shanchieh Jay Yang

Cyber attacks to enterprise networks and critical infrastructures are becoming more prevalent and diverse. Timely recognition of attack strategies and behaviors will assist analysts or resilient network defense systems in deploying effective means in anticipation of future threats. An attack can be characterized by the sequences of observed events that are relevant to critical assets. Earlier work has developed a semi-supervised learning framework to process large-scale events and extract attack behaviors. While the framework is designed to support online processing, the implementation requires extension and restructuring to support scalable automation of sustainable online network attack characterization.

This work builds upon the semi-supervised Bayesian classification framework, and aims at providing a modular and scalable system that supports a variety of features to describe attacks, ranging from packet level information to metadata produced by sensors, such as Snort and Bro. The system will continuously process data streams, generating newly learned models, as well as record critical information of aged behavior models. These behavior models will reflect the attack strategies that are relevant to the critical assets, enhancing the situational awareness and enabling predictive and resilient network defense. The accuracy of the models is demonstrated through comparisons to network topologies and scenarios provided from the source of the dataset utilized. These scenarios often encapsulate multiple complex network attack behaviors allowing for more realistic representations of network traffic over time and better test cases for experimentation.

Contents

List of Tables

List of Figures

List of Equations

Chapter 1

Introduction

1.1 Background

The ability to identify and characterize cyber attacks helps to provide more efficient and appropriate defenses. This capability provides the backbone for prompt and effective identification of attacks from malware designed by users with malicious intents. An attack can be characterized by the sequences of observed events that are relevant to critical assets. Examples of attack types include worms, viruses and botnets. A worm is a standalone computer program that maliciously exploits security vulnerabilities in a target machine to gain access to a network and spread to new targets. Computer viruses look to replicate themselves and infect other programs to either steal information, occupy bandwidth or corrupt data. A botnet is a collection of computers known as bots that are under the control of a remote user known as a botmaster. This collection of machines is then used in coordinated cyber attacks, such as distributed denial of service (DDoS), without the knowledge or permission of the owners of each compromised machine. Unique characteristics describe each type of attack which can be used to identify when one is encroaching upon a network.

Tools like intrusion detection systems (IDS) or intrusion prevention systems (IPS), also known as networks sensors, are widely used by system administrators to monitor traffic and filter malicious from benign traffic. These tools look for specific patterns in network traffic to identify potential threats such as inconsistent IP addresses and ports. Snort and Bro are specific implementations of IDSs that are incorporated into many network topologies to sift through packet level data. Snort is a rule-based IDS meaning it looks for predefined patterns within packets it observes to see if an alert should be generated. Bro is a framework that allows for passive network traffic analysis and security monitoring. Instead of alerts, Bro produces different log files based on what it observes and the settings that the user has specified. Incorporating the metadata from sensors creates a new dimension to describe observed traffic. Each new description of the traffic allows for higher precision when classifying each packet into a group consisting of similar traits. Earlier work has developed a semi-supervised Bayesian classification based machine learning framework to process large-scale events and extract attack behaviors from packet captures. Combining metadata produced by sensors and data from packet captures, this work builds upon the Bayesian framework and aims at providing a modular and scalable system that can extract and identify malicious behaviors from a variety of attack features. Metadata is information that describes one or more aspects about a specific dataset. The system is scalable in the sense that it can accommodate a work load that varies in size.

A semi-supervised machine learning based framework developed by [20] processes large-scale events and extracts attack behaviors. Bayes's Theorem is utilized to calculate posterior probabilities to determine if an actor is likely to be part of a malicious behavior resolved from packet level features. An actor is an entity like a bot or hacker who acts in a malicious manner. Before incoming traffic can be analyzed for specific traits related to each behavior, it has to be processed in a couple of phases. First, a target of interest within the network topology is chosen so incoming traffic can be related to it. Next, the traffic relevant to the resource being attacked must be separated from other potentially unrelated traffic. Following this, the different behaviors that exist within this sub-set of the capture must be identified for classification. This final step can be difficult to isolate due to the extensive amount of variation in the possible behaviors that can be observed from a cyber attack, such as backscatter. Backscatter occurs when a machine is the target of a DDoS attack utilizing multiple spoofed IP addresses. The machine will not be able to distinguish between legitimate traffic and spoofed traffic from a malicious source. Responses sent from the target machine will go to each spoofed address, provided by the malicious source, generating a lot of background noise.

While the framework in [20] was designed to support online processing, the implementation required extension and restructuring. Packets were the only supported source of data which limited the accuracy and functionality of the application. Support was added to allow metadata to be incorporated as a source of input utilizing the alert strings. Additional features, in conjunction with alternative sources of input data, help to better define behaviors.

In this scenario, training the framework to properly define the behaviors of collaborating actors is difficult as data is received in an unsupervised fashion. Unsupervised learning systems do not have a ground truth associated with the training data so it is difficult to know for sure if data is being properly grouped. Typically, this is related to a clustering problem where groups are automatically generated from a dataset. To model the traffic flow on a typical network, the framework was configured with online learning. Online learning looks at one data point at a time to update the feature distributions of its models. The framework uses online unsupervised techniques to classify packets observed from the network. This creates a hybrid problem in trying to both efficiently cluster the data while properly classifying new samples. Each new sample will change the distributions that represent the models in the system which, in turn, effects how well this sample and those before it fit into these models.

1.2 Motivation

Hackers and users with malicious intent are constantly adapting to advances in cyber security which makes it difficult to maintain a secure network. Cyber security analysts have an array of tools that help them to identify threats on their networks. Each tool is designed to look at different aspects of the network to identify separate types of intrusions and malicious actions. Due to the large amount of data generated from a network and all of the tools necessary to secure and maintain a network, it is difficult for a security analyst to maintain situational awareness of any one resource at any given time. The work done by [20] remedies this by providing a software tool and framework that provides situational awareness to a target of interest provided by the user.

This framework contained some limitations that prevented an analyst, or other tools, from fully utilizing the information generated. Originally, the main source of data was restricted to packet capture data stored in either a MySQL database or a log file. Existing tools such as Network Intrusion Detection Systems (NIDS) provide metadata that describe the traffic they observe. Incorporating multiple sources of data into one tool helps to bridge the gap between all of these utilities.

System administrators and security analysts will be able to view a larger amount of information in one consolidated application. This will also help to facilitate better communication between analysts and those developing the tools that they use. The analysts will be able to look at the data that matters the most to them when determining when an attack has occurred. Developers can correlate this data to techniques that help collect and display this information in a more efficient manner.

The goal of incorporating these changes was to increase the extensibility of the original framework defined and implemented in [20]. A critical, more refined set of features was derived from both the originally defined feature set and the newly incorporated features. This allows packets incident on a network to be classified to a particular empirical attack model with a larger degree of confidence when compared with other subsets of the full feature group. An alternate source of observable data was also implemented allowing the tool to scale with different volumes of input. Context was also provided from the new iCTF data allowing for better verification of the observed behaviors. With publicly available documentation and data, the empirical models generated can be compared against possible scenarios described by the documentation. This data also provides a wider set of test scenarios enabling the development of a more robust implementation of the framework.

Chapter 2

Related Work

To ensure a more robust and scalable environment for network attack characterization, it is necessary to first understand the inner workings of the original framework. Since the framework relies on a naïve Bayesian classifier within an online, unsupervised machine learning environment, different features are required so each new input can be classified correctly. Most of the available packet level features had been previously implemented, with a high success rate, so new observables were sought out. These new features were needed to test the hypothesis that expanding the dimensionality of the input data would provide more precise clustering of the attack behaviors. In an unsupervised machine learning configuration, there is no labeled dataset to check against during development. This creates the need for alternative methods to evaluate the performance of the features utilized in the clustering process.

2.1 Attack Segmentation and Model Generation

Strapp [20] worked to create an extensive framework, implemented in a software tool, that reads in packet level data and provides a higher level view of actors, who may be collaborating together to attack a network. This provides situational awareness around a given target of interest in a network allowing an analyst to examine events related to the given resource. The target of interest represents a server with resources that may be valuable to a potential attacker. A security analyst, protecting targets of interest, needs to be able to identify threats with an easy and efficient method. Different attack behaviors, around a given target of interest, are isolated and displayed as segmented groups to the user. Isolating relevant traffic becomes increasingly difficult with hackers employing malicious techniques such as IP address spoofing and information reconnaissance scanning. Using graphical prior probability in conjunction with packet level features extracted from network telescope traffic, the tool developed by [20] can produce attack models with a much higher accuracy when compared with the naïve four-hop approach. This approach assumes that all traffic four-hops or less from the target of interest is relevant.

This framework provided a novel unsupervised online machine learning based approach, in contrast with previous works, that utilized an offline approach. With offline machine learning, the application is trained until classifiers are built. Online machine learning looks at samples it observes and dynamically updates classifiers as new samples are received.

Both approaches can be implemented in a supervised or an unsupervised manner. Supervised learning uses labeled data to create classifiers which can be iterated in some instances to increase their accuracy. Unsupervised implementations look to create clusters with unlabeled data to best estimate varying groups in the dataset. The limitation with this method is, if the data being classified changes over time, the application must be retrained. With online learning, the tool can adjust dynamically to classify incoming data. However, the classifiers produced may not always represent the data accurately and resources are usually more limited, as the application typically can only view the sample once before deleting it. The novel approach developed by [20] started the first steps towards a newer, more dynamic, tool that security analysts can use to obtain a stronger sense of the malicious traffic inbound to valuable resources on their network.

2.2 Attack Characterization

It is important to characterize what kinds of attacks are inbound on a network and, in some situations, try to project what steps will occur next. In 2010 Du, Liu, Holsopple and Yang [7] worked on situation assessment to project the future steps in a multistage cyber attack. Du *et al.* present two techniques that combine the attack projection estimates for a more accurate prediction. They used the Transferable Belief Model for Capability and Opportunity (TBM-CO) assessments and Variable Length Markov Models (VLMM) to estimate future attack stages based on different alert attributes. The results presented showed that each method individually was able to perform well, but each worked better under different circumstances. TBM-CO detection performed well under strict firewall rules and dedicated service. VLMM worked well with sequential attack patterns by using Fuzzy inferences and was shown to be superior to TBM-CO when working individually. When the attack deviated from its normal attack pattern because of noise, decoy methods, etc. the TBM-CO was useful for recovering from the deviation to continue tracking. These experiments show how identifying and projecting what an attack is going to do allows for a more effective defense to prevent a network from being compromised.

Another set of experiments was conducted by Fava, Byers and Yang [10] with the use of VLMM to capture sequential attack tracks and predict likely future actions of the attack. The advantage to using a variable length Markov model over a static length Markov model is that it can adapt to newly observed behaviors in the attack sequence without requiring specific information about the network it is observing. This means that changes to the network configuration will not affect the accuracy of the model since it can adapt to the new situation. The adaptability of this system is important as cyber attacks change constantly to fool defense systems. Networks change a lot during a typical work day while people enter and exit a campus with mobile devices and while other devices such as laptops, desktops, printers and more are being added or removed. Having the ability to adapt to changes helps large scale networks become more secure.

One area that becomes difficult for defense systems is the ability to detect and react appropriately to group attacks where a larger collaboration of attackers is being used to compromise a network. In 2011, Du and Yang [8] investigated collaborative cyber attacks with social network analysis. Attack Social Graphs (ASGs), which are used to represent cyber attacks on the Internet, identify patterns that may be present from a joint attack from multiple sources. Unique patterns associated with collaborative attacks were identified from the use of ASGs. These patterns allowed for the clustering of these attackers into a single coordinated attack. As more people have become proficient with computers there has been an increase in malicious acts in cyber settings. In some cases, these people may work together in cyber terrorist groups, like Anonymous, to create a botnet or complete common attack goals. Finding a way to identify when groups like this are attacking helps to increase defenses and lower the chances of these groups succeeding in a security breach.

Leicht, Clarkson, Shedden and Newman [15] examined large-scale time evolving networks. They looked specifically at citation networks, but their ideas apply to cyber networks as well. Leicht *et al.* suggest three techniques for analyzing networks such that they adapt with time and are robust to changes. The first technique utilizes a model fitted to the observed network and uses an expectation-maximization algorithm. This kind of algorithm organizes the network data based on how likely it is to follow an expected model. The second technique they explored is a clustering method to maximize the modularity of the models generated. The third technique utilizes the time variation patterns using eigenvector centrality scores. Overall it was observed that a combination of using all three methods at once yielded the best results and provided more options for checks and balances. These findings follow with [7] in that using multiple systems has the best results allowing for better identification of patterns in cases where noise may be introduced. Multiple algorithms that are agnostic to both time and the network structure often provide increased network security.

Many Internet attacks including distributed denial-of-service (DDoS) and spam are now caused by botnets. Botnets are the result of multiple compromised machines that are coordinated either through peer-to-peer (P2P) connections or a botmaster that gives commands. Gu, Perdisci, Zhang and Lee [11] looked into methods for detecting botnet attacks. Currently, most botnet detection methods look for specific command and control (C&C) protocols. These methods will begin to fail as botnets change their C&C techniques to obfuscate their attack from detection systems. Gu *et al.* were able to develop a system that could detect botnets regardless of the communication protocol being employed. They make the assumption that bots within the same botnet will use similar communication protocols and will perform similar malicious activities. Their assumptions worked and led to an accurate detection system that could identify multiple types of botnets such as IRC, HTTP and P2P based botnets. This group of researchers also suggested the utilization of multiple correlation techniques to develop a faster system that can work in high speed and large network environments.

2.2.1 Anomaly-Based Detection with Statistical Analysis

In 2008, Gu, Zhang and Lee [12] discussed their methods to detect botnets with their program BotSniffer. Their tool relies on network-based anomaly detection to identify a botnet based on its C&C channel communication. The advantage to this approach is that it does not require prior knowledge of the signatures or the C&C server addresses. BotSniffer looks for traffic from multiple sources with consistent messages and responses. It is unlikely that human users would send and receive many similar messages all at once like bots in a botnet do while communicating and synchronizing. In the case of the Hypertext Transfer Protocol (HTTP) protocol being used, request types will always have “GET”, “POST” or “HEAD” in the first few bytes of the packet. An Internet Relay Chat (IRC) based botnet will have requests or messages such as “PASS”, “NICK” or “USER” in the packet making it easier to identify. White lists are used to compare the current traffic with destinations that are known not to be infected like Google or Amazon. This means that the detection approach that BotSniffer uses looks to identify spatial-temporal correlation and similarities in the responses of bots. To accomplish this, BotSniffer has a monitoring engine which examines network traffic and records any suspicious behavior. There is also a correlation engine which analyzes events observed by the monitoring engine.

Groups are formed based on the clients’ destination IP and port pair. Those who connect to the same server are grouped together. The Response-Crowd-Density-Check algorithm and the Response-Crowd-Homogeneity-Check algorithm are used to analyze the groups that are generated from the program. The first algorithm looks at the fraction of clients in a particular group whose message/activity behavior meets a certain threshold set by the users of the system. If these clients exceed the threshold then they are considered a dense response crowd. A botnet will have a high probability of being a dense crowd since they are more synchronized than humans. This also means normal traffic will have a much lower probability of being a dense response crowd.

To achieve a reasonable number of crowds to make a final decision about the traffic being a botnet, Gu, Zhang and Lee used the Sequential Probability Ration Testing (SPRT) algorithm which is also known as Threshold Random Walk (TRW). This algorithm calculates a score based on a sequence of observed crowds. As the threshold is reached from a random walk, then it is more likely a botnet. While this approach works well, it does require watching multiple rounds of response crowds, and in some cases, not all bots respond within the time window being observed. The second algorithm looks at how similar the members within a crowd are in terms of their responses. Members of a homogenous crowd might have similar IP address distribution, port range or a message with similar structure and content. This means that larger homogenous crowds have a higher probability of being part of a botnet. A larger number of clients means it is less likely that a homogenous crowd will form due to an increase in diversity. The combination of correlation and similarity analysis algorithms has been shown to be a fairly robust system with few false positives at identifying botnet attacks.

2.2.2 Time Zones

Dagon, Zou and Lee [5] worked to create a diurnal, or daily, model showing how botnets propagate based on time and location. They found that through binary analysis and observation, botnets tended to infect those in the same region. This discovery matches with other evidence of diurnal properties being discovered, which alludes to victims shutting down their computers at night. Geographic regions are important as well since they can host market segments for vulnerable software including special editions of operating systems. This means that depending on the time zone of the localized botnet, better predictions can be made as to when an attack might occur. It also becomes easier to priority rank malware based on time-of-release so that resources can be devoted to help stop faster spreading botnets. While the time zones and geographic regions of botnets are helpful, converting network addresses into these values is difficult with few resources to help. For this reason, previous models have not incorporated time and region. With larger populations of computers now available, it is necessary to consider such properties like time and region as diurnal models show that, based on these factors, the spreading rate of malware is affected. Dagon *et al.* also note that deriving the diurnal shaping function for each time zone requires a lot of data. It is unlikely botmasters will optimize malware release times to allow for the quickest spreading of their malware. This information allows for quicker response times from defenders identifying suspicious behavior.

2.2.3 Flow Detection of P2P Botnets

In 2012, Jiang and Shao [13] examined the evolution of botnet structures from command and control topologies to a peer-to-peer setup. They identified the challenges of detecting a P2P botnet including the traffic flow, encrypted communication, stealthy launches of malicious activities and communication through random ports. As a way to overcome these challenges, Jiang and Shao proposed a three step solution. The first step consisted of finding the intrinsic characteristics of C&C communication, or flow dependency, in P2P bots. Step two is a time-based flow dependency extraction algorithm. The final step is a custom technique to detect a P2P botnet, based on its flow dependency. This means that external tools such as honeypots are not needed for detection since only the flow dependencies of the communication traffic is needed. A honeypot is a decoy system configured to counteract or gather information about attackers on a network. Most P2P botnets “pull” new commands from the botmaster by exchanging messages with peers on its local peer list. These messages work as keep-alive signals or they can be new commands or updates from the botmaster. Since bots typically don’t have the full list of peer nodes, they communicate only with the ones they’re aware of.

The flow of C&C traffic shows which peers rely on each other for information. Common hosts in these traffic flow dependencies are inspected and usually identified as bots. Flows are used to represent the communication instead of raw packet information because they require less storage and computation space. The information for a flow has six parts including a five-tuple containing the IP addresses and ports of the source and destination, as well as the protocol being used, the arrival time of the first packet, the arrival time of the last packet, the total time of the flow which is the difference between the two arrival times, the number of packets total in the flow and the number of bytes in the flow, from the sum of the bytes from all of the packets. Once the flows are generated, they are run through a filter to identify suspicious or malicious traffic from standard traffic. The filter is not necessary for detection purposes, but it helps to reduce the amount of information and noise captured for more efficient detection.

To filter the traffic, Jiang and Shao look for three things: flows with small bytes, flows with a small length and flows that occur frequently. C&C traffic from a botnet typically consists of small packets that occur in short bursts of time and they show up frequently from the bots constantly communicating between peers. To reduce the amount of captured data and increase performance, only flows that start, after a short period of time, from the originally observed flow are analyzed. This is because flows are dependent on other recently occurring flows. Since flows depend on each other, the number of times that each flow from a bot occurs in the observed traffic should be equal. A scoring system is used to determine if pairs of flows are dependent upon each other. A large sample size is used to help reduce the amount of false positives from coincidental pairs. Two-level dependencies or pairs exist in normal traffic, such as between a client machine and a Domain Name System (DNS) server or a web server. This implies that multi-level flow dependency extraction should be used since P2P botnets usually have more than just two peers. Once dependencies are identified then clusters can be constructed. Botnets tend to have dense clusters as bots collaborate. This approach yields a high detection rate with few false positives.

2.3 Intrusion Detection System Alert Correlation

R. Sadoddin and A. Ghorbani [19] worked to conduct a survey that evaluates frameworks which correlate low level IDS alerts and matches them to higher level attack patterns. They state that the majority of techniques utilize either machine-learning based or data-mining based methods to aggregate alerts from different sources with varying levels of confidence. Frameworks typically employ several similar components to accomplish the task of correlating alerts. These components include normalization, aggregation, correlation, false alert reduction, attack strategy analysis and prioritization.

Normalization is the process in which alerts and messages can be exchanged between different IDS components to allow for inter-operability. Each IDS uses its own naming convention which makes it difficult to identify the specific information received from each sensor. Many standards have been proposed, but none have been adopted as an official standard for displaying and sending alert information. Aggregation is the act of bringing all these different alerts together once they have been normalized. Clustering techniques have been used to group alerts of similar types. These techniques are not perfect because each dataset is better distinguished with a different set of features. Identifying the best set of features is a difficult process without previously analyzing the data being used. Attributes are important to consider since different attacks have unique attributes that they rely upon. Other frameworks take this idea into account and work to cluster alerts with similarities in their data fields. Machine-learning and data-mining techniques are used to determine the similarity of features between alerts. Once similar alerts are grouped together, they can then be correlated with attack patterns.

Correlation is done between groups to find causal relationships that enable researchers to reconstruct attack scenarios. Sadoddin *et al.* describe four types of correlation techniques: scenario-based, rule-based, statistical and temporal. Scenario-based correlation is done by looking to see if alerts can directly connect with each other to construct an attack scenario. This can be accomplished with machine-learning methods and other pattern recognition algorithms. Rule-based correlation looks at the rules used to construct the alerts. The example given in [19] details that if alert ‘A’ prepares for alert ‘B’ within its conditionals, then those two alerts are said to be correlated. The limitation with these two methods is that each requires an extensive knowledgebase to be effective. Typically these have to be hardcoded into the application or taught with machine-learning techniques. Statistical approaches correlate two alerts if one alert type is able to cause any of the other arbitrary alert types. Conditional probability is then used to check how an alert type is related to the parent alert types which cause ‘A’. This causal relationship provides insight into which sequence of events may have generated alert ‘A’, which then caused alert ‘B’ to trigger and so on. Temporal correlation correlates two alerts based on how they influence each other over time. If alert ‘A’ can provide significant information regarding alert ‘B’ over time, then it is likely that alert ‘A’ caused alert ‘B’. This method makes assumptions that all alerts should be temporally related, which may cause missed correlations between alerts that experience random time delays when being generated.

Once alerts are correlated, false alert reduction takes place to improve the reliability of the data. In other words, this component works to distinguish between false positive alerts and true alerts that were generated. One approach is to use frequent episode rules. These rules are generated with machine-learning methods using data that contains no attack traffic. While operating in a network, if a sequence of alerts is found that matches a frequent episode rule, then it can be said with adequate confidence that these alerts are false positives. Confidence fusion is another method used to reduce false alerts by estimating the overall confidence of alerts based on evidence from low-level sensors. This is accomplished by evaluating an aggregate of confidence levels from low-level sensors with a predefined threshold value that determines if a final alert should be triggered. A reasoning framework was also developed to fuse confidence levels to determine which alerts were false positives. In this work, an alert-attribute graph is constructed where pre and post condition relationships between alerts are modeled. Conditional probability was used with each alert-attribute pair to estimate the confidence that an alert was not generated under false pretenses. A downside to these works is that a distinction between alerts related to an attacker’s actions and those issued as security events is not always clear. The concept of Fuzzy Cognitive Maps (FCM) was introduced to combat this weakness. The idea behind FCMs was that there are cause events and effect events which represent attack behaviors and security incidents in the system or network respectively. This allows FCMs to model attack scenarios using fuzzy values because different events are more likely to occur on some hosts rather than others. The overall impact that an incident is likely to have on a given host is used to determine the level of confidence in an alert. For the FCM framework, this level of confidence is defined by the concept of Incident Alert Levels (IAL) which varies depending on the fuzzy values of events and their impacts.

Once the alerts have been correlated and pruned of false positives, the data is analyzed to identify the attack strategies. Finding a complete attack strategy is often difficult because sensors may miss pieces of an attack due to false negatives. Higher level correlation techniques are used to bridge the gap between different isolated attack strategies to determine if they are unique attacks or if they are pieces of a larger attack. Similar to strategies employed to correlate the lower level alerts, the attack strategies are analyzed for patterns that will identify trends and relationships in the data. The knowledge base of the system being used is typically relied on during this process to identify patterns and relationships. The final step is to prioritize the processed alerts based on their severity. Alert severity is based on a number of facets including the network topology, security policy and network services. Previous works, as stated in [19], have worked to rank alerts based on these attributes. Alerts are grouped together based on the ranks assigned to them to illustrate incidents upon the network which allow security analysts to better judge counter measures and precautions for the incidents identified.

Valeur *et al.* [21] have developed a framework and corresponding application that describes a general alert correlation model based on a set of comprehensive components. The work in [21] follows the same basic steps that were outlined in [19] with a few minor adjustments. Their framework adds steps to the process where after data is normalized and aggregated, the alerts are then verified, reconstructed into an attack session, combined to evaluate multi-step correlation attacks, analyzed for overall impact and finally prioritized as in other frameworks. In these minor adjustments, the developers took extra steps to allow for the evaluation of broader, higher-level based attack patterns such as multiple attacks from one source or attacks that took place with multiple steps. During the reconstruction process, attack threads are built to refer to multiple attacks launched by one attacker that may be testing for different exploits available on a target machine. This process occurs while alerts are being merged and grouped, based on source and target attributes. Multi-step correlation evaluates the data in search of high-level attack patterns, where multiple steps were taken by an attacker to find an exploit, break into a target and then continue to perform malicious activities on this target device. The framework in [21] looks at two specific scenarios for multi-step attacks, which are recon-break in-escalate and island-hopping. The first scenario looks for attackers that identify vulnerabilities in a system, break into the system and then escalate their privileges to take over the system. In the second scenario, an attacker breaks into a target to take over other target machines on the network. These scenarios demonstrated the effectiveness of each comprehensive component in the framework based on the data being used. Together, these components create an effective correlation system.

2.4 Capture the Flag Data

Capture the flag games are held by many organizations every year. These competitions offer unique opportunities to both collect data for research and an environment to practice and learn techniques to protect networks and services. N. Childers *et al.* [3] discuss the benefits of the international capture the flag (iCTF) events hosted by the University of California, Santa Barbara (UCSB). These games use a newer premise that provides a more realistic scenario than generic CTF games. These scenarios ranged from creating botnets to security “treasure hunts”, where teams focused more on scenarios that real hackers would exploit, in contrast with just attacking other teams and defending a host machine for points. This premise leveled the playing field, as novices and experienced players alike had to start from “square 1” to understand the rules and complete the objectives. Aside from the scenario presented to the participants, there are strict time and resource constraints that put those involved in a more realistic setting to operate within. This not only provides a great education opportunity for those involved, but it also yields a solid source of data for researchers to work with.

In the case of their 2009 scenario, the UCSB presented a case called “Know Thy Enemy” which tasked those present with attacking simulated users with a drive-by-download attack and installing malware that made them part of a botnet. If teams were able to capture a user, they were required to have them report to a server, known as the Mothership, through an IRC channel to simulate actual traffic found when people in the real world are made to be part of a botnet. Since users had to follow strict guidelines to score points, it can be reasonably estimated which attack behaviors are present in the traffic captures. This does not serve as the ground truth of exactly what attacks were used, but it is much more reliable than data captured on the boundary of a given network. When using network boundary captures, it is extremely difficult to know which attack types, if any, are present in the data.

G. Vigna [22] interviewed Carnegie Melon undergrad student, Brian Pak, about his experiences with the 2010 iCTF game and how beneficial it was to participate in it. Pak discussed the amount of preparation required by him and his team to truly understand the playing field to take full advantage of the situation. By analyzing a publicly available Snort filter, provided by the game designers, and examining services that would be available, Pak’s team was able to find numerous vulnerabilities that they could exploit. These exploits allowed them to score points efficiently without being caught by the sensors used by the in-game resources, as well as to install backdoors in the machines used by other participants. With access to other teams, they could corrupt their scripts and services to prevent them from scoring points. Pak’s experiences with the games highlights just how complex iCTF scenarios are and how much one can learn from them. They also bring to light the amount of time and energy that is needed to develop each year’s game. The survey described in [19] highlights the challenges regarding the difficulties by those responsible for designing and implementing the iCTF games each year. However, the benefits that each game brings with it far outpace the difficulties that arise from the design process.

Chapter 3

Methodology

Strapp [20] previously worked to develop a framework which utilized the sparse information available in packet headers to synthesize representations of attack behaviors. The framework from [20] was able to successfully generate empirical attack models and segment irrelevant actors, but it contained limitations. These limitations were related to both the algorithm as well as the usability of the tool that implements the framework. Updating and fixing these areas allows for a framework and software utility that provides a more robust experience and more accurate characterization of observed behaviors. Changes made to the tool and corresponding framework include new features for the classification system. Revisions were also made to the model introduction phase and expectation maximization phase as well as bug fixes to prevent the program from terminating unexpectedly. Figure 3.1 below describes the overall flow of the system to process data.

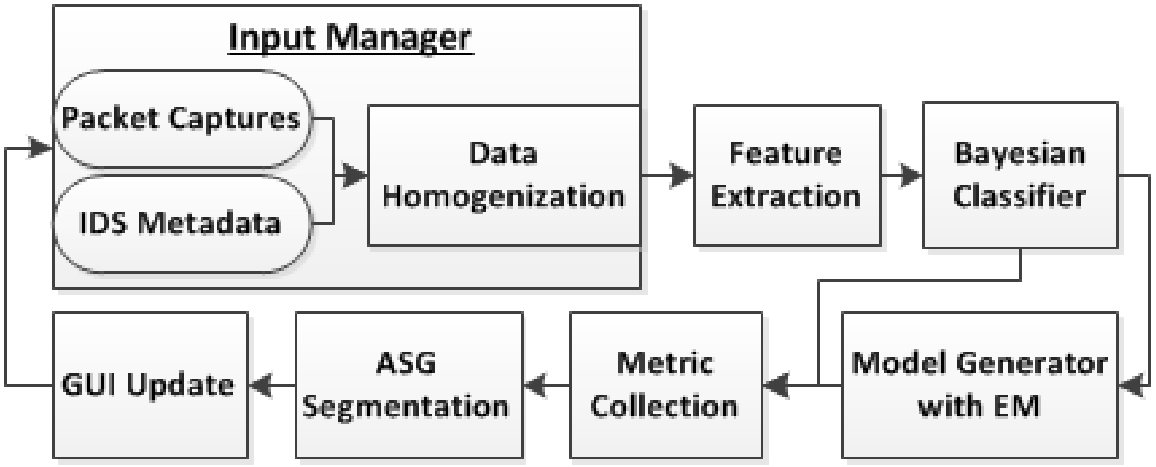
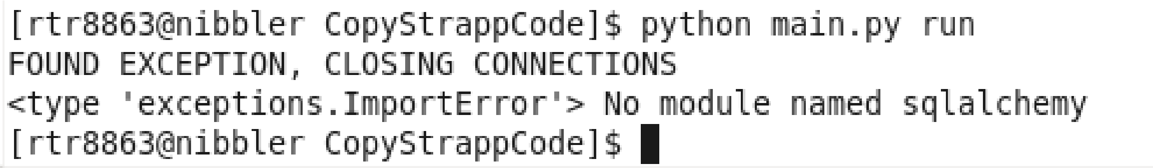


Figure 3.1: Overview of Algorithm and Revisions

3.1 Functionality Updates

3.1.1 MySQL Databases

Originally, this tool was developed to work with data from the Cooperative Association for Internet Data Analysis (CAIDA) group which is stored in a My Structured Query Language (MySQL) database. Connections made to MySQL need to be explicitly closed when not in use. If this is not done, the MySQL daemon, mysqld, keeps the connection open to listen for incoming traffic from the source. Due to the limited resources available on a given machine, the number of connections allowed is limited. If too many connections are opened at the same time, then no new connections can be made to the database. Connections to MySQL, through the SQLAlchemy interface for Python, were not properly cleaned up and closed in situations when the tool encountered an error and crashed. Additional checks were added such that in the event of a crash, connections to MySQL were properly cleaned up. Figure 3.2 below shows an example of how the tool cleans up after itself when an exception occurs.



```
[rtr8863@nibbler CopyStrappCode]$ python main.py run
FOUND EXCEPTION, CLOSING CONNECTIONS
<type 'exceptions.ImportError'> No module named sqlalchemy
[rtr8863@nibbler CopyStrappCode]$ █
```

Figure 3.2: MySQL Connections Closed After Exception

Once the tool could properly clean up connections following errors caused during development, a switch was added to allow for easier transitions between MySQL databases holding different datasets. Originally, only packet captures from a database containing CAIDA data was available for use. Support was added such that the name of a MySQL database on the local machine could be passed in as a parameter from the main method to pull data from that database. This allows for comparisons between newer and older datasets to be made for performance measurements as well as easier expansions to include new databases in the future.

The use of different datasets allows for testing to be done on a wider array of traffic behaviors collected from different sources and network topologies. While darknet traffic from CAIDA is very useful, it is impossible to know what type of traffic behaviors to expect and at what times they will be available. With other datasets, such as iCTF, it is possible to estimate what types of behaviors are available and when to look for them. This is because capture the flag events are monitored and data is collected in a more controlled environment [3].

Since capture the flag events offer more opportunities to test the tool against a wider variety of traffic behaviors, a database containing packet level data from a capture the flag event was created. Specifically, data from the International Capture the Flag event (iCTF) held in the University of California, Santa Barbara (UCSB) was used once the database switch was implemented. This led to the discovery of an error within one of the graph related features. When calculating the position of a node within the graph, a copy of the current graph structure is maintained in a separate part of the framework. The new data revealed that there were cases in which a node could be removed from one version of the graph, but not the other, which caused a path error to occur. Figure 3.3 below shows an example of a node that should have been removed, but was being used as part of a path to the target of interest.

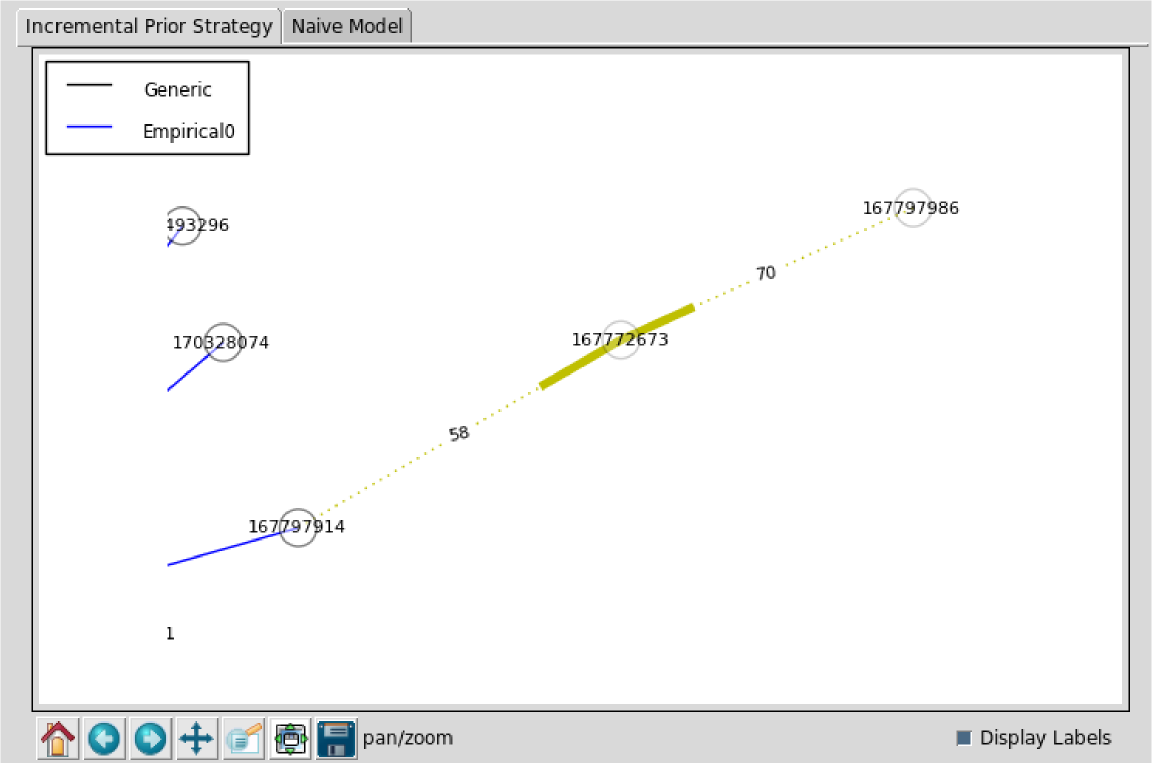


Figure 3.3: Example of Node to be Removed

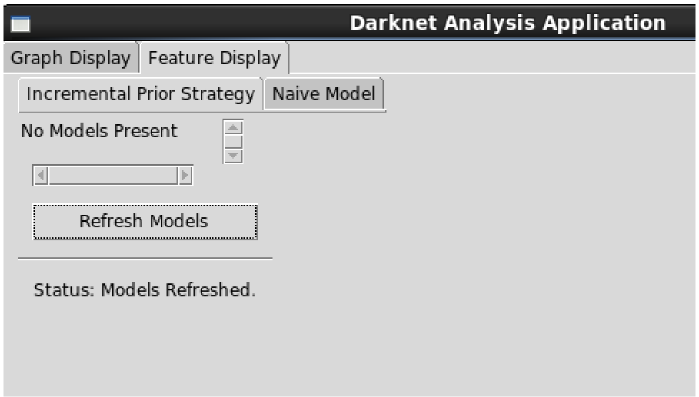
When a model is calculated to be irrelevant compared with the rest of the traffic, it is removed from calculation and shown as a segmented line in the GUI. In this case, the line is segmented, yet the node shown in the middle, 167772673, still existed in a copy of the network graph. This caused the program to attempt to find a shortest path from this node, which does not exist, to the target of interest. To remedy this situation, additional checks were implemented that detected situations similar to this. When this type of situation occurs, the paths that this node might have to any other disconnected nodes are ignored. The intuition behind this is that if a node is not connected to the target of interest, then it is not a relevant node and should no longer contribute path distances to be used during calculations.

3.1.2 Feature and Graph Plots

Feature plots and network graph plots are an important means for the tool to convey information to the user. Third party libraries are used to quickly and efficiently interpret the data from the tool and draw it in more convenient form. Calls to these third party utilities require that data be sent in a specific arrangement so plots and graphs can be properly formatted when drawn. In certain situations, data for a given plot may not be available yet. In this scenario, the tool would attempt to include a malformed list or missing data into a plot, which would cause the tool to cease functioning.

Checks were implemented along with standard try-catch blocks in Python to handle these erroneous situations gracefully. In scenarios where information may be missing, that particular plot is skipped such that other plots with valid information can be produced. If the graph cannot be properly updated, its update is canceled until the next time it needs to be redrawn. During experimentation, this did not occur often and the tool was able to recover quickly in the rare chance that a graph update failed.

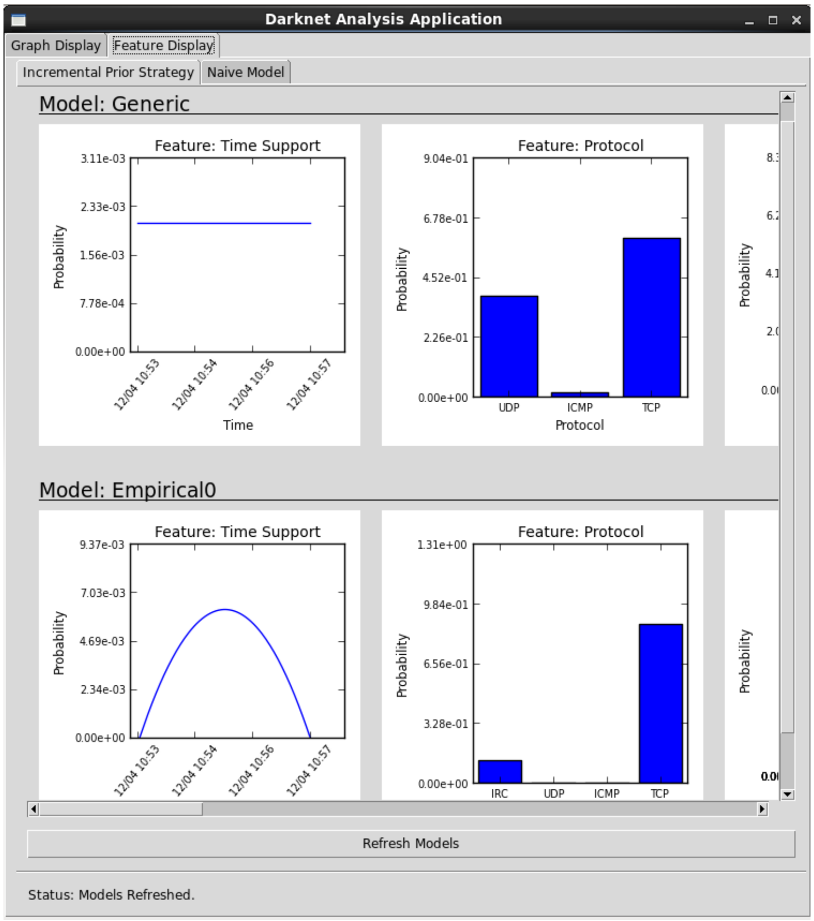
In a similar incident, there was an error in the graphical user interface (GUI) where the user would change views to see the current feature distributions of the models in the graph and the information would not generate. Figure 3.4 below shows what the user would see in this situation. Figure 3.5 below shows the feature distribution window once the error was identified and fixed. The tool relies on different identifier strings to track which modeling strategy the features being displayed belong to. For some of these strategies, the identifier was malformed causing the information not to be displayed. Once the tool could properly identify where the features belonged, the display was able to function normally.



No Models Present

Status: Models Refreshed.

Figure 3.4: Feature Display Tab Not Showing Information



Status: Models Refreshed.

Figure 3.5: Feature Display Tab Fixed

3.1.3 Environment Changes

The Python scripting language is known for fast development time and for being easy to learn and use. These qualities make it an ideal choice for prototyping new software projects. As these prototypes are innovated and improved, the flaws with Python's inner workings tend to shine through more. When developing on Linux based machines, some components of the operating system rely on specific versions of the Python language being available to correctly function. This potentially limits the development of a project depending on what versions the particular operating system supports and which version of Python the project was written in. It is well known that newer versions of Python in the 3.X range are not compatible with versions in the 2.X or lower range.

The Darknet Analysis framework and corresponding software tool were developed using Python 2.7.5 on the Ubuntu operating system. Ubuntu is based on the Debian operating system which supports Python 2.7. While working with the framework and tool, the working environment with Ubuntu needed to be changed to Red Hat Enterprise Linux (RHEL) due to a licensing restriction. While RHEL uses a similar kernel to operate, it only supports up to Python 2.6.6 or Python 3.X. Unfortunately, the third party software that the tool relies on uses specific versions of the different libraries to function. Reverting to the library that was written for version 2.6.6 of Python led to unforeseen compatibility issues in terms of missing functionality or deprecated library calls.

To restore the functionality required from these missing libraries, a virtual environment was created. Virtualenv is a tool developed to isolate Python environments such that packages and tools can be installed without effecting other virtual environments or the version needed by the operating system. Fortunately, virtualenv works with versions of Python that are 2.6+ and 3.X which encompasses the version used for the tool. Using an isolated environment with Python 2.7, the necessary libraries, at the correct versions, were installed. This restored most of the available features in the framework except for the plotting tools that rely on threaded Tcl functions. Plots can still be generated on an Ubuntu machine, but cause the tool to crash in a RHEL environment due to limited stack space. The core of the tool and framework can still function in a RHEL environment, but a different environment is necessary to see a visualization of the feature space of a given model.

To work around this, a Python script was developed based on work done by [20] to generate parallel coordinate plots. These plots include multiple y-axis bars, one for each feature to be represented. The lines that connect between each feature bar aim to illustrate the diversity of a given feature. Figure 3.6 below shows an example for how these plots are used to represent the feature space of a given model.

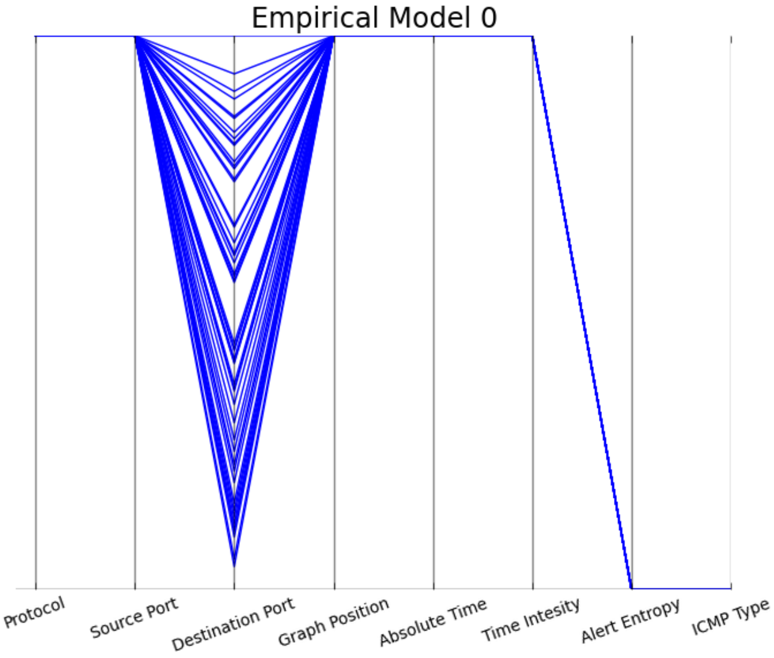


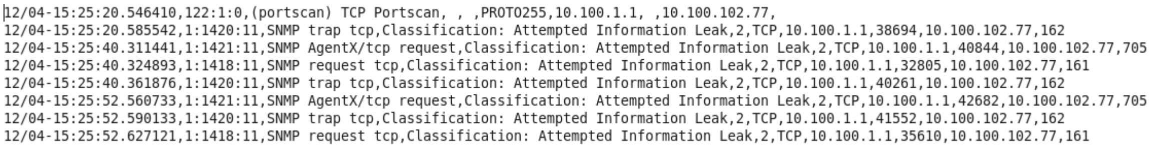
Figure 3.6: Example Parallel Coordinate Plot

From this example, the protocol of all samples in this model are the same since there is only one outgoing line from the 'Protocol' axis. There is a lot of diversity seen in the destination ports of the samples which is seen by the spread of values throughout the 'Destination Port' axis. In this case, it can be observed that the destination port entropy for this model is clearly stochastic while the source port entropy is deterministic. This is determined given that only one point on the 'Source Port' axis connects to all other points in the 'Destination Port' axis. Since the current working environment does not support the creation of these plots within the GUI of the tool, they are generated following the execution of the tool on a dataset based on files created during runtime. This ensures that the data is available for analysis regardless of the working environment the tool and framework are used in.

3.2 Algorithmic Changes

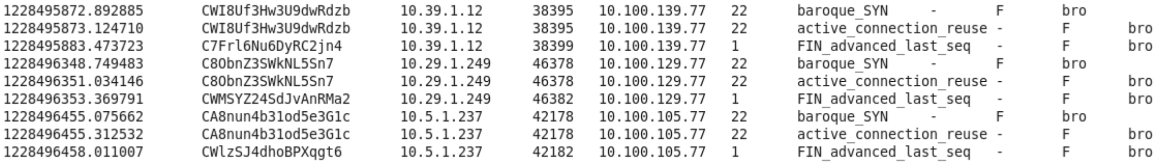
3.2.1 New Observables

The responsibility of a network is to pass packets containing data between different machines that are connected. While packets are the main form of observable data that can be analyzed to determine malicious behaviors, other forms of data and metadata exist from sensors and tools that help to monitor these networks. Intrusion Detection Systems (IDSs), such as Snort, monitor network traffic and search for packets that match known configurations of malicious activities. When an IDS finds a packet that matches a malicious action, an alert is produced that flags the packet and shows what malicious action it believes is being performed. These alerts are saved to log files that can be reviewed and analyzed by a system administrator or a network analyst. Figure 3.7 below shows an example of how the alert information from Snort is stored. Different formats are available, but the Snort fast alert style is commonly used to store this information.



```
12/04-15:25:20.546410,122:1:0,(portscan) TCP Portscan, , ,PROTO255,10.100.1.1, ,10.100.102.77,
12/04-15:25:20.585542,1:1420:11,SNMP trap tcp,Classification: Attempted Information Leak,2,TCP,10.100.1.1,38694,10.100.102.77,162
12/04-15:25:40.311441,1:1421:11,SNMP AgentX/tcp request,Classification: Attempted Information Leak,2,TCP,10.100.1.1,40844,10.100.102.77,705
12/04-15:25:40.324893,1:1418:11,SNMP request tcp,Classification: Attempted Information Leak,2,TCP,10.100.1.1,32805,10.100.102.77,161
12/04-15:25:40.361876,1:1420:11,SNMP trap tcp,Classification: Attempted Information Leak,2,TCP,10.100.1.1,40261,10.100.102.77,162
12/04-15:25:52.560733,1:1421:11,SNMP AgentX/tcp request,Classification: Attempted Information Leak,2,TCP,10.100.1.1,42682,10.100.102.77,705
12/04-15:25:52.590133,1:1420:11,SNMP trap tcp,Classification: Attempted Information Leak,2,TCP,10.100.1.1,41552,10.100.102.77,162
12/04-15:25:52.627121,1:1418:11,SNMP request tcp,Classification: Attempted Information Leak,2,TCP,10.100.1.1,35610,10.100.102.77,161
```

Figure 3.7: Example of Snort Alert File

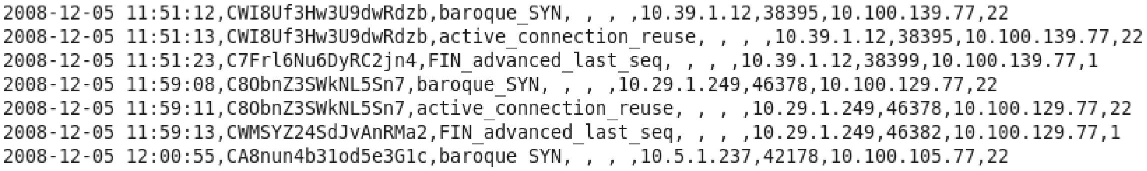


```
1228495872.892885       CWI8Uf3Hw3U9dwRdzb      10.39.1.12      38395   10.100.139.77   22      baroque_SYN     -       F       bro
1228495873.124710       CWI8Uf3Hw3U9dwRdzb      10.39.1.12      38395   10.100.139.77   22      active_connection_reuse -       F       bro
1228495883.473723       C7Frl6Nu6DyRC2jn4       10.39.1.12      38399   10.100.139.77   1       FIN_advanced_last_seq   -       F       bro
1228496348.749483       C8ObnZ3SWkNL5Sn7        10.29.1.249     46378   10.100.129.77   22      baroque_SYN     -       F       bro
1228496351.034146       C8ObnZ3SWkNL5Sn7        10.29.1.249     46378   10.100.129.77   22      active_connection_reuse -       F       bro
1228496353.369791       CWMSYZ24SdJvAnRMa2      10.29.1.249     46382   10.100.129.77   1       FIN_advanced_last_seq   -       F       bro
1228496455.075662       CA8nun4b31od5e3G1c      10.5.1.237      42178   10.100.105.77   22      baroque_SYN     -       F       bro
1228496455.312532       CA8nun4b31od5e3G1c      10.5.1.237      42178   10.100.105.77   22      active_connection_reuse -       F       bro
1228496458.011007       CWlzSJ4dhoBPXqgt6       10.5.1.237      42182   10.100.105.77   1       FIN_advanced_last_seq   -       F       bro
```

Figure 3.8: Example of Bro Log

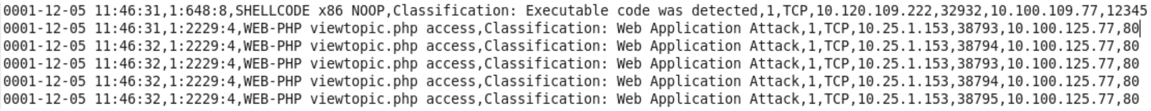
This information can be used to further describe traffic behavior observed on a network. An alert, depending on the IDS that produced it, contains additional information like the string identifying the alert type, the classification of the alert and a session number or identification. Unfortunately, the only consistent information between different IDS files is the packet that triggered the alert and the string describing the action matched. It is important to note that the different IDS sources are used strictly to accommodate varying versions of metadata, not to compare the performance from utilizing different intrusion detection systems. Figure 3.8 above shows a log file generated from the Bro IDS. The Bro system produces several various log files while processing data. Each file contains different information about the data such as the HTTP log describing all HTTP based traffic seen and the weird log containing descriptions about abnormal patterns in the traffic. The format of the file and the information available is much different than what is given in Figure 3.7. To maintain consistency between different IDS metadata sources, the alert strings are encapsulated with the packet level data that generated them to create a new form of observable information. In the dataset utilized, packets that generated alerts were filtered out of the packet capture files. When an alert file is specified, the tool will now reconstruct the traffic stream chronologically from both input sources. With the data reassembled, the overall account of behaviors that transpired on the network can be better observed.

In the original tool, packet captures were the only means available to analyze the traffic patterns related to the target of interest. MySQL was the main mechanism of retrieving and storing information pertaining to a particular set of data in the original tool. While MySQL provides an easy means to organize and store data, there is overhead associated with interfacing with the database and querying the tables stored within to retrieve data. Changing or updating information held in a database can be a slow process as all entries within the table being changed have to be altered to the desired format. Instead of utilizing MySQL to store alert data, the raw output files were instead processed into a comma separated value format (CSV) which has native support within Python. Figure 3.9 and Figure 3.10 below demonstrate the more unified representation of the data once it has been processed from its raw form. Information that is not present, either because it was not captured or not available within the given file, is represented as a blank space. In particular, the Bro Weird Log, by default, does not store the protocol of the packet it filtered while the Snort file does not store the year the alert was generated. The year of an alert can be recovered by comparing to the date that the packet captures in the same input stream.



```
2008-12-05 11:51:12,CWI8Uf3Hw3U9dwRdzb,baroque_SYN, , , ,10.39.1.12,38395,10.100.139.77,22
2008-12-05 11:51:13,CWI8Uf3Hw3U9dwRdzb,active_connection_reuse, , , ,10.39.1.12,38395,10.100.139.77,22
2008-12-05 11:51:23,C7Frl6Nu6DyRC2jn4,FIN_advanced_last_seq, , , ,10.39.1.12,38399,10.100.139.77,1
2008-12-05 11:59:08,C8ObnZ3SWkNL5Sn7,baroque_SYN, , , ,10.29.1.249,46378,10.100.129.77,22
2008-12-05 11:59:11,C8ObnZ3SWkNL5Sn7,active_connection_reuse, , , ,10.29.1.249,46378,10.100.129.77,22
2008-12-05 11:59:13,CWMSYZ24SdJvAnRMa2,FIN_advanced_last_seq, , , ,10.29.1.249,46382,10.100.129.77,1
2008-12-05 12:00:55,CA8nun4b31od5e3G1c,baroque SYN, , , ,10.5.1.237,42178,10.100.105.77,22
```

Figure 3.9: Bro Log Processed into a CSV File



```
0001-12-05 11:46:31,1:648:8,SHELLCODE x86 NOOP,Classification: Executable code was detected,1,TCP,10.120.109.222,32932,10.100.109.77,12345
0001-12-05 11:46:31,1:2229:4,WEB-PHP viewtopic.php access,Classification: Web Application Attack,1,TCP,10.25.1.153,38793,10.100.125.77,80
0001-12-05 11:46:32,1:2229:4,WEB-PHP viewtopic.php access,Classification: Web Application Attack,1,TCP,10.25.1.153,38794,10.100.125.77,80
0001-12-05 11:46:32,1:2229:4,WEB-PHP viewtopic.php access,Classification: Web Application Attack,1,TCP,10.25.1.153,38793,10.100.125.77,80
0001-12-05 11:46:32,1:2229:4,WEB-PHP viewtopic.php access,Classification: Web Application Attack,1,TCP,10.25.1.153,38794,10.100.125.77,80
0001-12-05 11:46:32,1:2229:4,WEB-PHP viewtopic.php access,Classification: Web Application Attack,1,TCP,10.25.1.153,38795,10.100.125.77,80
```

Figure 3.10: Snort Alerts Processed into a CSV File

Using built-in tools allow for quicker file operations and better portability between different working environments. Interfacing with a MySQL database requires the use of the SQLAlchemy application programming interface (API) within the framework. When using third party libraries, different release versions may not necessarily be available for each potential working environment. Interfacing with these additional libraries requires more complex configuration software and increased error handling. This increase in complexity is needed to prevent the program from terminating unexpectedly due to incidents such as a disconnect from the database software employed. Following the implementation of these modifications to the input data path, the framework is now scalable to handle large volumes of continuous data. Figure 3.11 below summarizes the updates made to the data input stream that is fed into the framework for processing. The highlighted area within the figure represents the changes made to the process flow of the system.

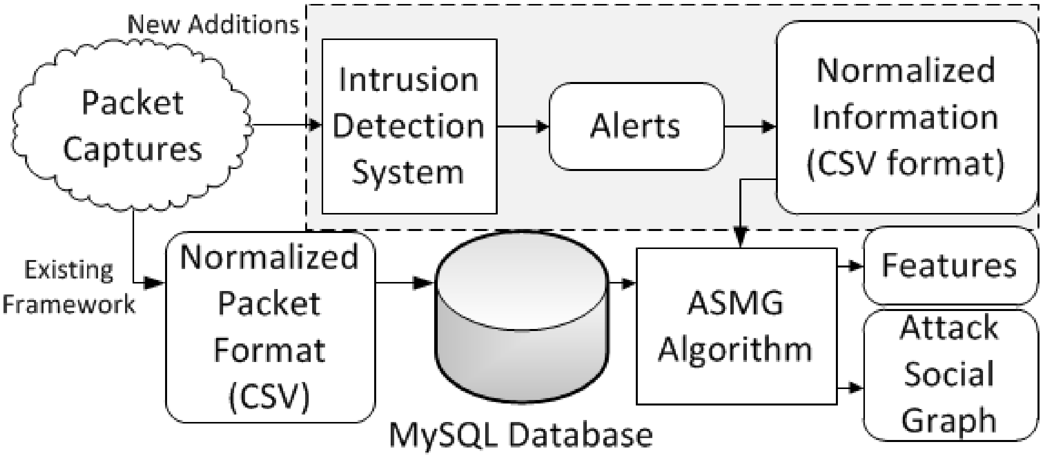


Figure 3.11: Overview of Processing Flow

3.2.2 New Features

Feature selection for a machine learning framework is important for achieving the best possible performance. In a supervised machine learning setting, the data has labels that can help determine which features are relevant. In an unsupervised system, such as the darknet analysis framework, feature selection is particularly difficult as there are no labels to help determine strong features to describe the data. Dy *et al.* in [9] discuss the importance of finding strong features to use for unsupervised machine learning tools. Intuitively, all of the information available seems appropriate to use to try to describe the data. Unfortunately, this notion is not always correct as features may be redundant or irrelevant to the dataset. If a feature is irrelevant, then it may be difficult to distinguish the different clusters in a dataset as the data may overlap in the feature space. A redundant feature is one that does not add any value to the system. An example of this is when multiple clusters can be easily distinguished by one feature while other features fail to describe new dimensions in the feature space. It is important to identify unique dimensions in the feature space that add value to the classification system.

With the inclusion of a secondary source of information to the framework, the original definition of an attack model had to be updated. As defined by Strapp in [20], an attack model consists of a collection of probability distributions based on relevant features extracted from the traffic being processed. Features common to both datasets and those available in the new data source were examined to determine which were apt to describe all potential traffic on a network without overlapping information described by the previous features. The new feature distributions selected from both the packet header information as well as alert file information are Absolute Time, Time Intensity and Alert Entropy. The two temporal features, Absolute Time and Time Intensity, are common to both sources of data since a time stamp is provided with each packet and alert value. Alert Entropy is specific to data extracted from alert files. As discussed earlier, the alert string is used as the basis for this feature as it is a consistent piece of information amongst different IDS files. Temporal features were not previously utilized by the system due to the focus being on spatial aspects. This makes them strong candidates as newly included features. The alert information is a source of new data not previously implemented by the system reducing the potential redundancy of an entropy based feature. Table 3.1 below summarizes the previously implemented features of the original framework. These features were derived from packet header information from data provided by CAIDA. Table 3.2 below provides a brief overview of the newly implemented features based on the new iCTF2008 dataset.

Table 3.1: Previously Implemented Features [20]

Feature	**Description**	**Equation**
Protocol	A simple, discrete feature that represents the communication method between two nodes.	$P(\mathcal{P}) = \frac{\sum_{i=1}^{N} I_{\mathcal{P}}(x_i)}{N}$
ICMP Type	A discrete feature which represents simple packets that may be maliciously performing information reconnaissance to identify a target and learn about potential weaknesses.	$P(\mathcal{I}) = \frac{\sum_{i=1}^{N} I_{\mathcal{I}}(x_i)}{N}$
Port Entropy	Examines the source and destination ports in a packet to see if they are deterministic or stochastic. Steady use of a port usually means a service under attack while use of varying ports may indicate reconnaissance actions.	$P(\mathcal{P}) = P(S) + P(\mathcal{P}\|D) * P(D)$
Graph Position	How close a particular node is to other nodes with a high probability of being in a malicious path within the same attack behavior model.	$GP(i) = \sum_{j \in G} \frac{P(M\|P_{i,j}^{0}) P(M\|P_{i,j}^{-1})}{\sum_{k \in P_{i,j}} \frac{1}{P(M\|P_{i,j}^{k})}}$

The first entry in Table 3.1 above describes the equation used to calculate the probability of seeing protocol $\mathcal{P}$ in a given attack model. This is calculated with the indicator function $I_{\mathcal{P}}$ to count the percentage of protocol $\mathcal{P}$ in a given model M. The percentage is based on the total number of samples observed N and each individual element is described by x_i. Entry two uses the same concept to calculate the probability that a given ICMP Type $\mathcal{I}$ exists within a model. The port entropy of a model is given in entry three which describes the simplified equation used to compute the probability of a port value $\mathcal{P}$. Given both the percentage of the port number and the "randomness" of all port values seen in the model the probability of seeing any given value $\mathcal{P}$ can be computed. The entropy is broken up into the probability of the values being stochastic or $P(S)$ and the probability that it is deterministic $P(D)$. These values are found by statistical bootstrapping with replacement and counting the number of deterministic values. The probability of a port value given that the ports within the model vary or $P(M|S)$ is not calculated because the given port does not matter if they are being chosen randomly. To find the probability of the model given that it is deterministic or $P(M|D)$ is found in the same manner of the previous features, finding the percentage of the value within the model M.

The final row in the table describes the formula used to calculate the position of a given node i in the overall attack social graph. Finding the graph position is based on the concept of closeness centrality or how close a node is to any other given node. The numerator describes the weight of the starting hop $P(M|P_{i,j}^{0})$ and ending hop $P(M|P_{i,j}^{-1})$ within the total path from node i to node j given as $P_{i,j}$. This value will be higher if the path between the two nodes stays within a given model M. The denominator represents the distance calculation of a given hop $P(M|P_{i,j}^{k})$ within the total path. This represents the inverse harmonic mean of distances from node i to all other nodes within the graph $\mathcal{G}$. Nodes that are part of the same attack model with higher probability will contribute more towards the position calculation than those that are further away in the graph or span multiple models.

Table 3.2: Newly Implemented Features

Feature	Description	Equation
Absolute Time	The difference of the absolute date and time between the current packet and the one being compared against. This can represent relevant actors who are working together within a certain window of time.	$dist. = \|t_{current} - t_{previous}\|$ $P(t_{current}) = \frac{window - dist.}{window}$
Time Intensity	Looks at the amount of activity observed between two machines in terms of packets per second. This rate is recalculated any time communication between the same two sources is observed.	$P(\mathcal{R}) = \frac{\sum_{i=1}^{N} I_{\mathcal{R}}(x_i)}{N}$
IDS Alert Entropy	An IDS produces alerts that can be stored in log files. These alerts are typically generated when an observed packet matches certain characteristics defined by the rule set used by the IDS. The entropy of these alert types is used to identify different patterns in observed traffic behavior.	$P(\mathcal{A}) = P(S)+$ $P(\mathcal{A}\|D) * P(D)$

3.2.2.1 Absolute Time

Absolute Time is a continuous feature that determines the probability of an observation based on the date and the time that it was captured. The intuition for using the date and time comes from Dagon *et al.* in [5] who studied the propagation of botnets based on time zone information. They observed that within a given botnet, the zombie host machines responsible for carrying out malicious activities changed throughout the day. In geographic locations where infected users were powering down their machines for the day, the amount of activity would decline. Traffic in time zones where the day was beginning anew increased as users powered on their machines. It can be concluded from this study that the probability of a packet belonging to a given empirical behavior diminishes as the distance in time increases between the time the packet was captured at and the time the behavior was first observed. The theory of dynamic time warp (DTW) was used as a basis for this feature..

The concept of DTW was used to mathematically model this feature. Dynamic time warping is a metric used to measure the similarity between two sequences despite variances in speed or time. Bean in [1] discussed DTW and other methods like the longest common subsequence (LCS) algorithm as possible solutions to analyze the similarity of different attack tracks over time. DTW can also be utilized in other domains such as signature verification. Mohammadi in [16] used this concept to match important points between two signatures to identify legitimate and forged samples. In this case, the DTW method was sufficient as the Euclidean distance formula is employed to calculate the cost, or distance, between any two points in the tracks being compared.

A matrix describing the distance between every possible set of points in the two tracks is created using the Euclidean distance formula (3.1) below. In the equation, a_i represents an element in track a to be compared with b_i which represents a corresponding element within track b.

$$d(a,b) = \sqrt{\sum_{i=1}^{N}(a_i - b_i)^2} \tag{3.1}$$

Once the matrix of costs is calculated, a continuous path, or warping path, is constructed from the starting point $(1,1)$ to the opposite corner of the matrix (n,m). The optimal warped path is one that minimizes the distance between all shared points between the tracks being analyzed. Equation (3.2) below, as stated by [1], describes the methodology behind the algorithm to find the minimal warping path. Essentially, the dynamic time warping between a sequence A and a sequence B is given by the minimum path w where w_i is the i^{th} element in the warping path being considered and K is the normalization factor to account for sequences of different lengths [1].

$$DTW(A,B) = min\, w \left\{ \frac{\sqrt{\sum_{i=1}^{K} w_i}}{K} \right. \tag{3.2}$$

With this feature, the similarity between time stamps of packets is not helpful in determining how likely a given packet is to be part of an attack behavior. Instead, the behavior model that has the minimal distance in time to the packet being evaluated is the most likely behavior that characterizes the packet. Revisiting the Euclidean distance formula given by (3.1), the values of a and b would become the time of the current packet $t_{current}$ and the time of the most recent packet to be classified with the behavioral model being compared t_τ. Since the only dimension being investigated is time, the output of Equation (3.1) becomes the absolute value of the difference in time between two packets. Equation (3.3) below represents the revised mathematical representation of the absolute distance in time between two packets.

$$d(t_{current}, t_\tau) = |t_{current} - t_\tau| \tag{3.3}$$

The distance in time is not enough by itself to describe a packet's likelihood of being part of an attack behavioral model. To measure the probability that a packet belongs to a specific behavior, a relevancy window needs to be predefined. This window in time is different for each dataset used and values that fall outside of this window have zero probability for belonging to a given model. Given a frame of reference, the value calculated by (3.3) can be divided by the value for the time window to determine the probability of this packet belonging to a behavioral model.

Using the straight value of distance over the defined window space gives the opposite probability of the desired value. As the distance in time increases, the probability would increase that the packet belonged to that model. To fix this, the inverse of this value is used to decay the probability as the distance increases. Equation (3.4) below describes the probability equation where d is the value calculated from (3.3) and w is the value set for the relevancy window.

$$P(t_{current}) = 1 - \frac{d(t_{current}, t_\tau)}{w} \tag{3.4}$$

Alternatively, (3.4) can be represented as one unified fraction as seen in Equation (3.5) below.

$$P(t_{current}) = \frac{w - d(t_{current}, t_\tau)}{w} \tag{3.5}$$

With this equation, it is possible to determine the likelihood of a packet belonging to a given attack behavioral model. The probability decays linearly as the distance increases between the time of the current packet and the time of the packet most recently incorporated into the attack model being evaluated.

For the iCTF2008 dataset, packets are time stamped with the UNIX epoch time they were captured at which is measured in seconds. Since traffic was observed frequently due to each of the teams working quickly to compromise different resources on the network, the times of the observed packets do not show much variance. The relevancy window w chosen for this dataset was 120 seconds. Participants would likely try many different techniques to learn the network topology and exploit resources to capture flags during the eight hours alloted to play the game and score points. Two minutes seemed to be a reasonable time frame to isolate the time needed to capture fast scans and other techniques that teams might utilize during the game.

3.2.2.2 Time Intensity

Time Intensity is a discrete feature because it can be modeled with a histogram based on the time stamp of a packet and the edge that it traveled along. This feature represents the rate at which packets are transmitted through an edge in the attack social graph. The intuition behind this feature is derived from two areas. From a behavioral standpoint, some cyber attacks take place over a large period of time and may have a lower rate of packets per second. Other attacks may be quick and try to complete tasks within a smaller time frame resulting in much higher packet rates.

An example of a slow moving attack would be the SlowLoris attack. This is a type of denial of service (DoS) attack in which a program attempts to keep as many connections open to the victim machine as possible. This is accomplished by sending partial requests periodically to the target machine where the request is expanded on, but never completed. The end result is that the target machine exhausts its pool of connections preventing others from accessing its resources. These types of attacks are also difficult to detect as the packets engaging the target often mirror legitimate traffic [6].

A smurf attack is another example of a way to cause a DoS. During a smurf attack, a malicious user sends an Internet Control Message Protocol (ICMP) echo request to a broadcast domain such that all computers on the domain send ICMP echo replies. Normally, this procedure is used by routers to test the reachability of a machine on the network. In this case, the source address is spoofed to be the address of the target machine. By spoofing the target machine's address, the influx of ICMP echo reply messages will consume bandwidth allocated to the victim and cause a DoS to legitimate users [14]. These two examples demonstrate how two attacks can have the same end result, but produce drastically different behaviors. The SlowLoris attack accomplishes a DoS through a slow consumption of resources while a smurf attack tries to consume bandwidth by sending a large volume of messages as quickly possible. These behaviors are distinct and should be distinguishable based on the rate of traffic they produce, which is the intuition behind the Time Intensity feature.

To calculate the rate of traffic per second along an edge, the time stamp, source IP address and destination IP address are extracted from the current sample. The source and destination IP addresses identify the edge in the network that this sample belongs to while the time stamp is used later to calculate the rate of the edge. An edge is defined as the connection between two communicating nodes where a node is the IP address of a machine. A Python dictionary object is used to store the information for each edge. Since this feature is relative to when the edge is first observed, each entry in the dictionary file includes the initial time the edge was created, the time of the most recently observed packet and a running count of the total number of packets seen since the initial time. Equation (3.6) describes how the packet rate on an edge is calculated.

$$EdgeRate = \frac{PacketCount}{CurrentTime - InitialTime + 1} \tag{3.6}$$

What this equation describes is that the larger the packet count is over the current time window, the larger the packet rate over the edge will be. The additional 1 in the denominator acts as an offset to prevent a divide by zero error. In many cases, it is possible to receive multiple packets within the same second so the initial edge rate will be equal to the packet count until the next time interval that packets are received in.

The traffic rate, or edge rate, by itself is not usable as a metric to calculate the likelihood of a packet being part of an attack behavior. To calculate the probability, a histogram is created representing the different rates from all of the relevant edges of a given attack behavior model. Since this is a discrete empirical feature, each entry in the histogram is a discretized range of packet rates. Once the maximum rate has been determined from the edges, the histogram is generated with buckets set to an adjustable size until the maximum value is encapsulated within one of the discrete buckets.

The size of the buckets is adjustable, but an initial value of 5 was used as a default rate to prevent groupings of higher traffic volumes with lower ones. The initial value proved to be an appropriate choice as some empirical models contained multiple edges. These edges consisted of lower rates near 1 packet per second and higher edge rates closer to 20 packets per second. Using the default value, bucket 0 contains packet rates 0 to 5 with bucket 1 holding rates 6 to 10 and so on. The first bucket encapsulates the packet rates that are less than 1 because they are less common than the other values, particularly with the iCTF2008 dataset in which a large volume of traffic was observed throughout the duration of the game.

$$P(\mathcal{R}) = \frac{\sum_{i=1}^{N} I_{\mathcal{R}}(x_i)}{N} \tag{3.7}$$

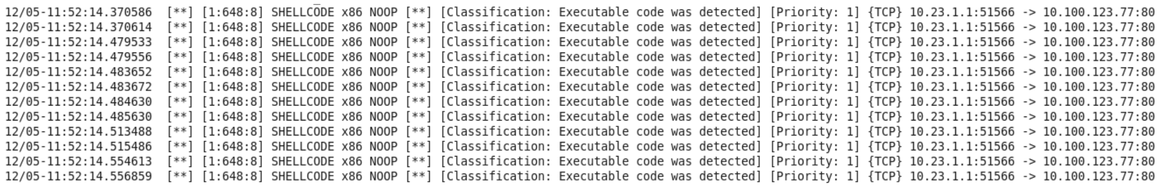
Once the histogram is complete, Equation (3.7) above is used to determine the probability of the rate $\mathcal{R}$ for a given model. To accomplish this, Equation (3.7) counts the number of occurrences of the edge rate $\mathcal{R}$ using the indicator function $I_{\mathcal{R}}(x_i)$ on all N edge rates observed by the current model. The indicator function is simply used to find all occurrences of $\mathcal{R}$ within the set of all edge rates of the attack model. Another way to conceptualize this process is the total number of edge rates in the histogram bucket that $\mathcal{R}$ belongs to divided by the total number of edge rates N in the attack model being evaluated.

3.2.2.3 Alert Entropy

Alert Entropy is another discrete feature that looks at the "randomness" or variance in the alert values captured by the IDS within the network topology. The intuition behind this feature is derived from Strapp in [20], Saddodin in [19] and Valeur in [21]. Work centered around alert correlation efforts demonstrates the value of metadata from IDSs [1, 19, 21]. These values can help to provide a stronger overview of events that may have transpired on a network. The design of the framework already allows packets, and now alert values, to be correlated in a graphical sense by connecting behavioral models together based on having relevant paths to the target of interest specified by the user.

Using the entropy of a feature was originally devised by [20] to calculate the probability of a given port based on the variety of possible deterministic and stochastic behaviors. This allows for a distinction to be made between more meaningful deterministic or "non-random" behaviors and stochastic or "random" attack behaviors. Deterministic behaviors can be generated from both benign and malicious traffic. In the benign case, legitimate network traffic making queries to a website or other remote resource would generate behaviors that utilize the same ports and IP addresses consistently. A malicious example of a deterministic behavior would be a specific service being targeted as in a MySQL injection attack against a target running the database software without proper protection implemented. In the opposite case, stochastic behaviors tend to better describe a larger variety of malicious actions. If an attack were to scan available ports on a victim machine, the variation of the destination ports would appear to be stochastic as no particular service is being targeted.

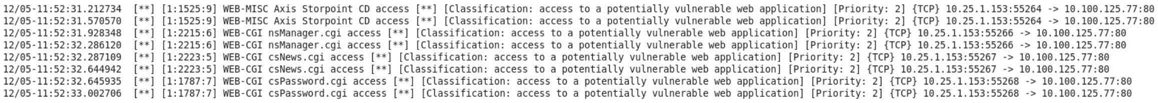
Combining the idea of feature entropy with the newly added alert data allows for a new metric to identify attack behaviors. Similar to the previously implemented Port Entropy feature, alerts can describe both stochastic and deterministic actions against a target machine. Figure 3.12 below shows a sample from a Snort Alert file showing a deterministic attack type being executed against a target machine. It can be seen that the same target is attacked by the same source with the same attack type, as described by the alert string.



```
12/05-11:52:14.370586  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
12/05-11:52:14.370614  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
12/05-11:52:14.479533  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
12/05-11:52:14.479556  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
12/05-11:52:14.483652  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
12/05-11:52:14.483672  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
12/05-11:52:14.484630  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
12/05-11:52:14.485630  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
12/05-11:52:14.513488  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
12/05-11:52:14.515486  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
12/05-11:52:14.554613  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
12/05-11:52:14.556859  [**] [1:648:8] SHELLCODE x86 NOOP [**] [Classification: Executable code was detected] [Priority: 1] {TCP} 10.23.1.1:51566 -> 10.100.123.77:80
```

Figure 3.12: Example of Deterministic Alert Behavior

This set of alerts is consistent with buffer overflow exploits in Transmission Control Protocol (TCP) traffic [18]. A buffer overflow attack is where an attacker will purposefully write information beyond the defined bounds of a memory segment, or buffer, such that a variable, pointer or return address will be changed [2]. If a remote user is successful in changing any one of these fields, the flow of the program can be altered. An example of this exploit would be a login program where an attacker would overrun the buffer storing the password hash of a benign user. The attacker would then supply their password when prompted along with additional characters followed by the hash of their password. These extra characters will overrun the buffer used by the program and inject the hash of the attacker's password in the space normally reserved for storing the hash of the benign user's password. This would cause the program to hash the attacker's password and compare it to the hash injected by the buffer overflow, granting the attacker access to the target machine [2].



```
12/05-11:52:31.212734  [**] [1:1525:9] WEB-MISC Axis Storpoint CD access [**] [Classification: access to a potentially vulnerable web application] [Priority: 2] {TCP} 10.25.1.153:55264 -> 10.100.125.77:80
12/05-11:52:31.570570  [**] [1:1525:9] WEB-MISC Axis Storpoint CD access [**] [Classification: access to a potentially vulnerable web application] [Priority: 2] {TCP} 10.25.1.153:55264 -> 10.100.125.77:80
12/05-11:52:31.928348  [**] [1:2215:6] WEB-CGI nsManager.cgi access [**] [Classification: access to a potentially vulnerable web application] [Priority: 2] {TCP} 10.25.1.153:55266 -> 10.100.125.77:80
12/05-11:52:32.286120  [**] [1:2215:6] WEB-CGI nsManager.cgi access [**] [Classification: access to a potentially vulnerable web application] [Priority: 2] {TCP} 10.25.1.153:55266 -> 10.100.125.77:80
12/05-11:52:32.287109  [**] [1:2223:5] WEB-CGI csNews.cgi access [**] [Classification: access to a potentially vulnerable web application] [Priority: 2] {TCP} 10.25.1.153:55267 -> 10.100.125.77:80
12/05-11:52:32.644942  [**] [1:2223:5] WEB-CGI csNews.cgi access [**] [Classification: access to a potentially vulnerable web application] [Priority: 2] {TCP} 10.25.1.153:55267 -> 10.100.125.77:80
12/05-11:52:32.645935  [**] [1:1787:7] WEB-CGI csPassword.cgi access [**] [Classification: access to a potentially vulnerable web application] [Priority: 2] {TCP} 10.25.1.153:55268 -> 10.100.125.77:80
12/05-11:52:33.002706  [**] [1:1787:7] WEB-CGI csPassword.cgi access [**] [Classification: access to a potentially vulnerable web application] [Priority: 2] {TCP} 10.25.1.153:55268 -> 10.100.125.77:80
```

Figure 3.13: Example of Stochastic Alert Behavior

Figure 3.13 above shows the opposite case seen in Figure 3.12 where the same target machine is receiving different types of traffic from the same malicious host. Even though the same source and destination are used in this example, the ports used by the malicious host are changing, as is the value of the alert string. Since Snort Alerts provide a classification string to help organize the alerts, it is obvious that the malicious user is looking to exploit a vulnerable web application. To accomplish this, they are employing several different techniques to try to gain access to the target machine through this application as described by the alert strings.

These examples demonstrate the validity of using the entropy of a feature as originally proposed by [20]. The same intuition that was applied to ports from packet headers can be applied to the alert strings found within the files generated by Snort. Since strings are typically more difficult to process and typically require more computation time than numbers, the built-in hash function native to Python was used to convert each alert string into an integer. Hashing the alert string also requires less memory to store the value making it less taxing on system resources. Once hashed, this allows for the use of the original port entropy equation as developed by [20] with the alert integer values as given below by Equation (3.8).

$$P(\mathcal{A}) = P(\mathcal{A}|S)P(S) + P(\mathcal{A}|D)P(D) \tag{3.8}$$

This equation states that the probability of a given alert value is equivalent to the probability of the alert value being trigged with a stochastic behavior plus the probability of the alert value appearing given a deterministic behavior. Statistical bootstrapping is used to determine the value of $P(D)$, the probability of a deterministic behavior, and the value of $P(S)$, the probability of a stochastic behavior [20]. Statistical bootstrapping consists of random sampling from the total set of samples, with replacement, to determine the total occurrences of stochastic and deterministic samples. As seen in Figure 3.13, the specific value of the alert string is less relevant when a myriad of malicious techniques are utilized to exploit a vulnerability in a target. This means that the probability of an alert string appearing due to a stochastic behavior, $P(\mathcal{A}|S)$, is unimportant as any value is equally probable because each observation is treated as independent and identically distributed (i.i.d.) by the classifier. As this observation matches that of the Port Entropy feature, it is appropriate to use the reduced form of Equation (3.8).

$$P(\mathcal{A}) = P(S) + P(\mathcal{A}|D)P(D) \tag{3.9}$$

Equation (3.9) above illustrates the reduced equation used to represent the Alert Entropy feature.

3.2.3 Revision of the Model Introduction Strategy

The role of the model introduction strategy is very important towards the overall functionality of the darknet analysis framework. This portion of the framework is responsible for judging when a new empirical attack model is necessary. Originally, the creation of new empirical models was based on an arbitrary threshold set in relation to the graph efficiency of the generic model. Graph efficiency is a measurement of the closeness of the nodes in a graph. The closer the nodes are to each other, the more efficient the graph. To relate the graph efficiency value to the different attack models within the overall attack social graph, Strapp [20] developed Equation (3.10) below.

$$E(\mathcal{G}_M) = \sum_{i \neq j \in \mathcal{G}_M} \frac{P(M|\mathcal{P}^0_{i,j})P(M|\mathcal{P}^{-1}_{i,j})}{\sum_{k \in \mathcal{P}_{i,j}} \frac{1}{P(M|\mathcal{P}^k_{i,j})}} \tag{3.10}$$

This equation examines the distance between all possible pairs of nodes within the attack social graph. The distance is defined as the inverse probability of the attack model M, currently being processed [20]. This calculation of the distance between pairs of nodes is represented by the denominator in (3.10) above. The contribution of the distance between the nodes i and j is given by the numerator in (3.10). The first term, $P(M|\mathcal{P}^0_{i,j})$, is the probability of the attack model at the starting edge of the path between the nodes and $P(M|\mathcal{P}^{-1}_{i,j})$ is the probability of the model at the terminal edge of the path. Fundamentally, the efficiency of the attack social graph given an attack behavior model is the sum of the contributions of a given node pair (i, j) divided by the total weight of the full path between (i, j).

Conceptually, the use of the graph efficiency of the generic model makes sense to use as once the unclassified samples are clustered tightly enough, a new model will be generated to represent the feature space of these samples. However, if the new samples never cluster tightly enough due to the feature space being too spread out, the defined threshold would never be reached and no empirical model would ever be generated. The feature space used cannot be easily adjusted to counter this problem as each dataset would require special tuning to properly distinguish the different behaviors that may be present. This presents a problem as no behavior is ever defined for the user to investigate. The use of a naïve model in this case would present better performance over the ASMG methodology presented by Strapp in [20] as all potential behaviors are represented in one model and assumed to be collaborating.

To remedy this short coming in the darknet analysis framework, a revision was made to the logic driving the model introduction. Instead of using a statically defined threshold to determine when the unclassified samples are grouped tightly enough, a dynamic threshold based both on the graphical prior and the number of observed samples was implemented. At the start of processing, when only the generic model is available, a simple threshold based on the number of unclassified packets is used to determine when the first empirical model should be generated. This simple threshold waits for 10 unclassified observations to be seen before introducing an empirical model. A value of 10 was chosen due to previous experimental observations showing that, on average, 10 unclassified samples proved to be sufficient in representing an initial feature space for a model.

Utilizing a simple threshold provides a means to kick-start the standard introduction logic by providing an initial feature distribution to compare against. Once this first model is generated, a separate set of logic takes over to judge when further empirical models are required. This main model introduction logic employs a more complex thresholding system to resolve when additional empirical attack models are necessary. The graphical prior probability, given in Equation (3.11) below as defined by Strapp [20], is used in conjunction with a larger threshold to ensure that enough evidence has been accumulated to reasonably conclude that an additional model should be spawned.

$$P(M) = \frac{E(\mathcal{G}_M)}{\sum_{M_i \in M_{all}} E(\mathcal{G}_M)} \tag{3.11}$$

From this formula, the graphical prior is defined as the graph efficiency of the current attack model normalized by the sum of the graph efficiencies over the set of all current attack models. Once the prior probability is calculated for the generic model holding the current set of unclassified samples, it is compared against a dynamic threshold based on the number of currently active models. If this threshold is not obtained as unclassified samples are accrued, a secondary measure is used to ensure that if enough evidence has been obtained, then another attack model is created. Equation (3.12) below summarizes the algorithm now used for model introduction.

$$NewModel = \begin{cases} True & \text{if } N < 1 \quad and \quad S \geq SMALL\_THRESHOLD \\ True & \text{if } N \geq 1 \quad and \quad \Big(S \geq LARGE\_THRESHOLD \\ & \quad or \quad \big(\text{Prior(Generic)} \geq \alpha * \frac{1}{N+1} \\ & \quad and \quad S \geq SMALL\_THRESHOLD\big)\Big) \\ False & \text{otherwise} \end{cases} \tag{3.12}$$

This formula checks the number of active empirical models N to determine which threshold to check. When no empirical models are active, then the small threshold, shown as $SMALL\_THRESHOLD$ above, discussed earlier is used to determine when a sufficient number of samples S has been collected. When at least one empirical model has been made and is still active, the more complex threshold is used, as seen in the second case of Equation (3.12). In this case, the two means of satisfying the overall Boolean condition are illustrated.

The first portion ensures that if enough new evidence has been gathered, represented as $LARGE\_THRESHOLD$, a new model will be generated to represent the feature space of these samples. This larger threshold was set to a value of 100 to allow ample time for new samples to be collected and properly compared against all other active empirical models. In the second portion of this condition, the prior probability of the generic model is compared against the inverse of the number of models plus 1 ($\frac{1}{N+1}$). This ensures that enough previous evidence has been observed to warrant one additional model. A discount factor α is multiplied by this fraction. Discount factors are typically used in the financial field to calculate the return on an investment. In this case, α is used to determine whether newer or older evidence has a stronger influence on the model introduction. Lower values of α emphasizes newly observed evidence over older evidence which is ideal for this system as it continuously observes new data and needs to model the most up to date behaviors. This dynamic threshold is used in conjunction with the original, smaller threshold to ensure that the feature space has stabilized before being moved to a new empirical attack model. If none of the above conditions is met, then no model is generated and processing continues normally.

Figure 3.14 and Figure 3.15 below illustrate the effects of these changes to the system. Before the revisions were implemented, behaviors were not properly identified such that all packets remain in the generic model as unclassified samples. This is shown in Figure 3.14 below. Following the revisions, using the same feature set in the classifier, the different behaviors inherent in the attack social graph become visible as illustrated by Figure 3.15 below.

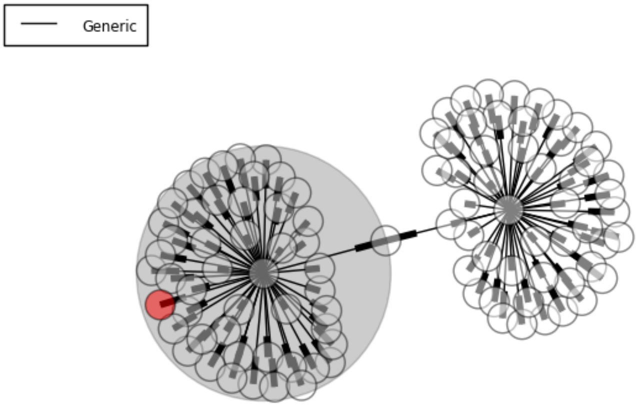


Figure 3.14: Example of ASG Before Model Introduction Revisions

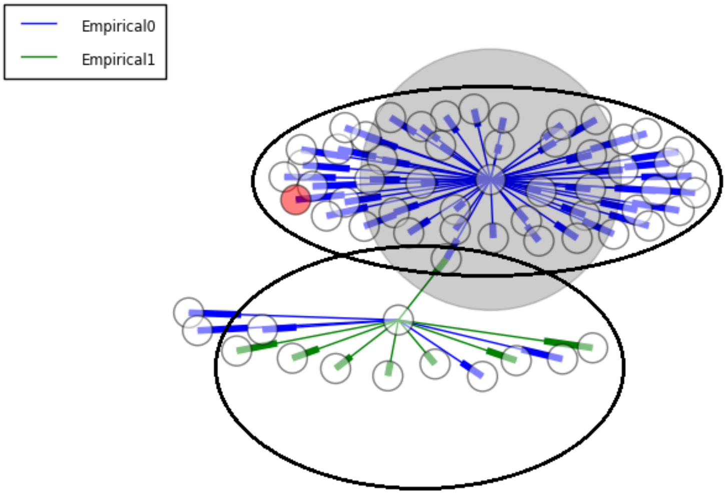


Figure 3.15: Example of ASG After Model Introduction Revisions

3.2.4 Revision to the Expectation Maximization (EM) Phase

In the case that a new model is generated, the expectation maximization (EM) algorithm is executed. This allows each sample in the system to identify with the model that maximizes its posterior probability. Consequently, as samples enter and leave a given empirical attack model, the feature distributions are updated to reflect the change. As samples are fit to their maximum a posteriori (MAP) model, the models will more accurately reflect the feature space of their samples.

The EM algorithm is used in many problem domains including signal processing and machine learning. This algorithm is designed to find the maximum likelihood or the MAP value estimates of the parameters used to define statistical models. Moon in [17] reviews the EM algorithm and how it applies to the domain of signal processing. There are two major steps that take place with this algorithm. The first step is called the expectation step which consists of estimating the unknown variables in the system, based on the current value of the parameters, by generating an expected log-likelihood function. Following the expectation step is the maximization step. During the maximization step, the parameters are updated to maximize the log-likelihood found in the expectation phase. These two steps are iterated until the parameters converge to a constant value [17].

Following the introduction of a new model, the framework iteratively classifies all samples with the set of all active attack models which now includes the newly introduced model. This is similar to the EM algorithm in that samples are classified to the models which updates their feature distributions. Once the feature distributions are updated, the classification process starts again and iterates through the two stages. This iteration is done a finite amount of times to reduce computation time. Originally, the generic model, which used to temporarily store unclassified packets, was included in the set of active models. This allowed certain samples that were not as closely affiliated with the available empirical attack models to re-associate with the generic model.

This is not ideal because all samples should always associate with the behavior model that maximizes their posterior probability, and they should not be left unclassified. When the EM phase of the new model processing logic executes, the generic model is no longer included with the set of active attack models such that samples are classified only with the available empirical models. This ensures that all samples remain classified and that all active models best represent the feature space described by the samples currently in the system.

3.2.5 Implementation of Data Collection Metrics

With all of the changes to the original darknet analysis framework, it is important to measure the performance of the system and verify that the updates work as expected. Since the system continuously reads new input to process, data must be collected regularly to document the performance through time. To measure the confidence of the posterior probability calculated for each sample, the average MAP value, the standard deviation and the standard error were calculated for the set of all samples currently in the system. A snapshot of the current ASG topology and parallel coordinate plots for each active empirical model are also generated to illustrate how the models and network change with time. The average MAP value is calculated by Equation (3.13) below where x_s is the MAP value of sample s and N is the total number of samples.

$$\mu = \frac{\sum_{s_i \in S_{all}} x_{s_i}}{N} \tag{3.13}$$

The standard deviation is also calculated to represent the amount of variation that exists from the average value μ. Equation (3.14) below shows how the standard deviation is calculated.

$$\sigma = \sqrt{\frac{1}{N} \sum_{s_i \in S_{all}} (x_{s_i} - \mu)^2} \tag{3.14}$$

Once the mean and standard deviation have been calculated, the standard error as given by (3.15) below is calculated and used as a means to measure the confidence. The standard error represents the variance in the estimate of the population mean. Since only a finite amount of samples can be stored by the system at any given time, older samples get replaced by newer samples as they are observed by the tool. This means that the value calculated by (3.13) represents a sample mean to the total population of samples collected. This set of measurements is done for both active packets used by the attack models as well as segmented packets. The information is kept for the segmented values to demonstrate that these packets were indeed meant to be removed and that they no longer fit with any active behaviors in the ASG.

$$SE_{\bar{x}} = \frac{\sigma}{\sqrt{N}} \tag{3.15}$$

Two data collection points were chosen to document how the system performs while processing data. The first area chosen was within the logic that processes a newly observed packet. An adjustable variable determines how often data is collected. Originally, a collection occurred every 10 observations to maintain a high resolution track of the changes occurring in the system. This value was chosen due to previous observations showing that, on average, 10 samples were needed to create a new empirical attack model. A resolution of this size allowed for important changes of the ASG to be documented including the generation of new models. Unfortunately, with such a high resolution, system resources would be fully utilized which caused some experiments to fail. The resolution was changed to be collected after every 100 observations to work around this issue. Even with fewer data points collected, sufficient information was collected at this resolution to accurately depict system performance.

The second data collection point was placed immediately after the EM phase. This was done to visualize the changes to the ASG topology after all samples and attack models had converged to their optimal models and feature distributions respectively. Once these two data collection points were implemented along with some helper scripts written in Python, data collection from the tool was done autonomously. While the tool is running, the metrics discussed earlier are being recorded along with snapshots of the ASG. Once the tool finishes, parallel plots, CSV files and charts are generated automatically for later analysis.

Chapter 4

Results

4.1 Design of Experiments

Traffic captures and alert information provided by the University of California, Santa Barbara (UCSB) from their international capture the flag event in 2008 were used to measure the performance of the updated framework following the updated mechanics. This dataset was used instead of the original Cooperative Association for Internet Data Analysis (CAIDA) because the developers of the event created additional public information to document the network topology given to the teams as well as information describing the participants. CAIDA data has very little, if any, documentation associated with it which increases the difficulty of assessing the abilities of the framework. Since the original framework relied on this data as a metric for performance evaluation, some new test cases utilized this data to serve as a baseline. Ideally, the updated framework should be able to handle the original test cases while also functioning properly with tests derived from the new data source.

While the information included with iCTF data is in no way a ground truth about the different attacks and events that transpired during the game, it provides a means to judge what traffic patterns to expect. These inferred patterns are used to measure the performance of the graphical output of the tool. The attack social graph (ASG) topology should be able to at least moderately match the expectations generated by the documentation associated with the game. A lot of effort was put into designing the games and creating rules to facilitate certain attack types while prohibiting others. DoS attacks, for instance, were strictly prohibited due to the limited time and resources allotted to the game. Traffic leaving a team's subnetwork was also purposefully routed through a central server known as the "mainbox" for all teams so statistics could be collected about the network usage [3].

Two key areas were focused on to validate the changes made to the original system. The first area was in evaluating the features, both the original set of features and the newly implemented features to see how different pairings of these sets influenced the classification system. The second area of focus was ensuring that the tool could identify and model the changing complexity of the ASG as defined by the incident behaviors observed over time. These essential focus points aim to provide maximum coverage to the amendments made to the system including changes to improve reliability and robustness to non-ideal data configurations. The original system was designed around the CAIDA data while the new implementation was made to be more generic allowing for diverse types of datasets and different sources of data to be used.

4.2 Feature Evaluation

The choice of features utilized within a machine learning environment are extremely important. If the features used to describe the data are too weak, then samples may not be properly classified or grouped together [9]. Incorporating the feature set described by Strapp [20] along with the newly defined feature set, there are now seven total features available to describe attack behaviors. Subgroups of the seven features were generated and evaluated to determine which group best characterizes the largest variety of attack behaviors.

Table 4.1 below describes the groupings of the features experimented with. Theoretically, there are 2^7, or 128, possible combinations of these features that could be evaluated. It would be impractical and time consuming to generate information for each combination of the total feature set for every potential target of interest in the dataset. These groupings are based on the most representative types of features and work to simplify the total set of required experiments necessary.

Table 4.1: Feature Subgroups to Characterize Traffic

Feature	**All**	**Critical**	**Packet**	**Original**	**New**	**Temporal**	**Spatial**
Protocol	✓	✓	✓	✓			
ICMP Type	✓		✓	✓			
Port Entropy	✓	✓	✓	✓			
Graph Position	✓	✓		✓			✓
Absolute Time	✓				✓	✓	
Time Intensity	✓	✓			✓	✓	
IDS Alert Entropy	✓				✓		

The first grouping defined as "All" represents the total feature space. Using the entire feature space represents the naïve assumption that each one is relevant and concise where this group provides the strongest descriptions of the input data. As warned by Dy *et al.* in [9], this assumption is not always accurate because features may overlap and describe the same part of the feature space. If this happens, then the performance of the classifier may deteriorate and not display the best representation of the data.

The "Critical" group represents the select features from the overall set that are able to describe at least as many behaviors as the naïve grouping of all features. This group serves to represent the diversity of the feature types without overlapping the other features in the group. Looking at the members of the group, there is representation of an entropy feature, a spatial feature, a temporal feature and a packet level feature. The intuition behind the features chosen for this group is to get the best possible representation of the elements that characterize a generic input dataset. These features were selected for this grouping once the performance of the other feature groups was evaluated and parallel coordinate plots were generated to visualize the feature space of those groups.

Features in the “Packet” group are all derived from information found in packet headers. These were all defined for the original version of the darknet analysis framework as packet level information was all that was available for processing. Since malicious users have the ability to spoof or obfuscate their intentions by providing misleading information in the packet headers, these features are not necessarily the strongest. Despite this disadvantage, not all attackers spoof packet header data which can still provide vital information regarding their intended malicious action.

The next two feature groupings are simply a measure of the performance for the original feature set and the new feature set to see how they operate on their own. The original feature set was tested against numerous targets of interest from the CAIDA dataset and demonstrated strong performance [20]. This set did not include any temporal features however, as focus was mainly put on packet header information and the novel graphical based approach. To make up for this oversight, the temporal aspects of the input data were utilized along with the elements found within the alert files. The essential component being evaluated is how well the two sets of features can operate on a wider set of data given their more specialized definitions to the datasets they were originally defined for.

The final two groupings “Temporal” and “Spatial” were inspired by the preceding two groups. In many problem domains, the use of time and position are often employed to measure data. Due to the availability of these features within this problem domain, it was appropriate to evaluate the importance of these feature types. Temporal and spatial features also provide sufficient overlap between the different sources of input data now available to the framework as packet level information is available from each input path.

4.2.1 Determining a Baseline Measurement

Figure 4.1 below represents the baseline performance metric from the CAIDA dataset that all other feature groups are compared against when working with CAIDA data. This image was produced by the original tool with the features listed under the “Original” grouping in Table 4.1 activated. It is clear that multiple attack behaviors were identified and distinguished from each other. This is also an example of where many of the behaviors identified were determined to have connected with the target of interest by means of probablistic intersection and have been segmented from the model. They remain in the image as dashed lines to maintain a visual history of the changes incident to the ASG.

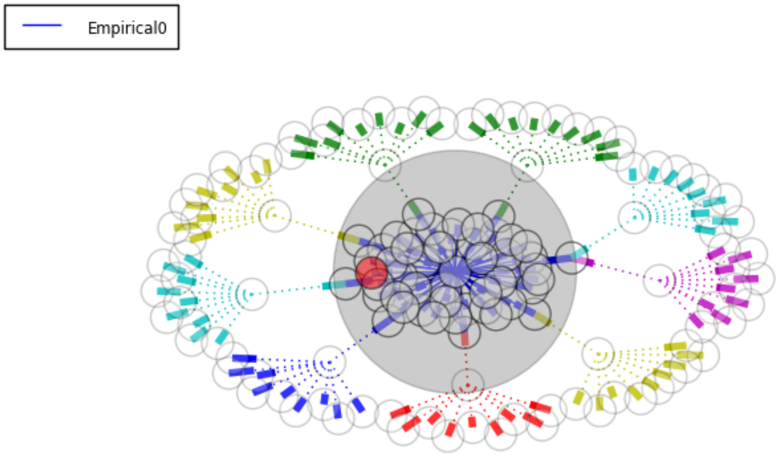


Figure 4.1: Baseline Model from CAIDA Data

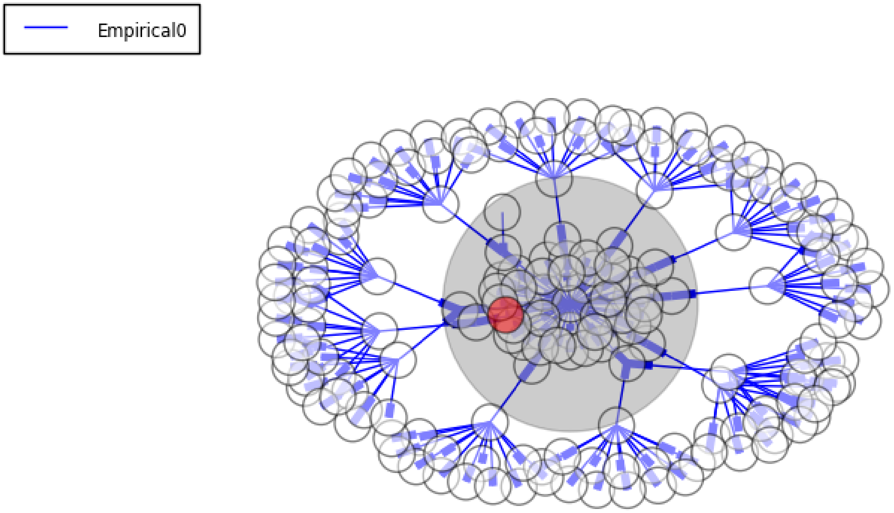


Figure 4.2: Naïve Model from CAIDA Data

Figure 4.2 above shows the representation of the naïve model generated by the tool. This model makes the basic assumption that traffic four hops or less away from the target of interest is relevant. Only one empirical model is generated and all behaviors are matched to this model regardless of how distinguished they are from each other. If a feature group is unable to distinguish behaviors better than the naïve model when the baseline model provides a much clearer distinction, then the given set of features are not strong candidates for membership in the "Critical" grouping.

To test the hypothesis that the full set of features serves as a viable baseline to compare against, all relevant features were activated and run under the same conditions that the ASG in Figure 4.1 was generated with. A relevant feature is defined as one where information can be extracted from the dataset being used. Since CAIDA data does not include alert level information, the Alert Entropy feature had to be deactivated despite being a member of the "All" feature group.

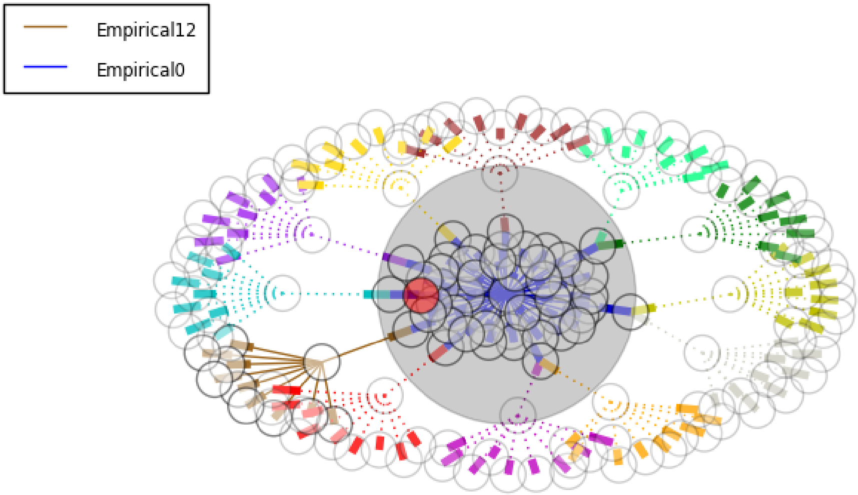


Figure 4.3: Models Generated by Relevant Group for CAIDA Data

The ASG illustrated in Figure 4.3 above was generated using the relevant features for the CAIDA dataset. A relevant feature is any one from the "All" group in Table 4.1 that also exists within a given dataset. Data from CAIDA, for example, would not include alert level information making the Alert Entropy feature irrelevant. In this case, the tool was modified such that every empirical model has its own unique color associated with it. Previously, the models cycled through a finite range of basic colors causing confusion as to which model some nodes belonged to. This issue can be seen in Figure 4.1 where several different models of the same color exist. For this particular dataset, the full range of features appears to perform successfully by identifying and displaying the same groups of behaviors seen in the baseline image.

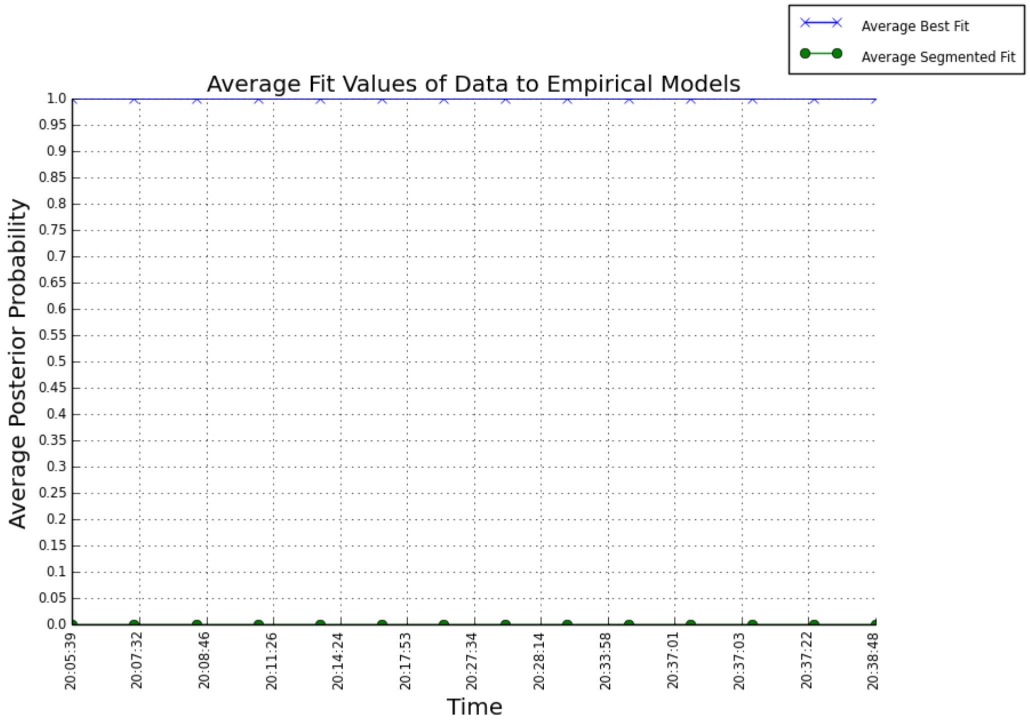


Figure 4.4: Confidence Data of Relevant Group for CAIDA Data

Figure 4.4 above confirms the strong performance as speculated by the groupings shown in the corresponding ASG. The line with 'x' ticks marking each data collection point represents the average best fit of all samples in the system to their MAP model. To verify that the segmentation process was not negatively impacted by the changes, the line with 'o' marks was plotted. This line shows the fit of samples that have been deemed irrelevant and segmented from processing. Since the model they belong to was removed, each time a data point is created the segmented samples are re-associated with an active attack model to determine how well they fit with processing.

In each case, the data fits with high confidence as indicated by the lack of visual error bars. This means that there is no variance in the mean and all samples fit with high probability to their models. In the case with the segmented packets, their low posterior probability to the active models validates the segmentation logic as well as showing clear distinctions between the attack models generated. This highlights the use of the full feature set as a baseline going for future tests. Looking at the parallel plots produced from the software, this hypothesis is further confirmed.

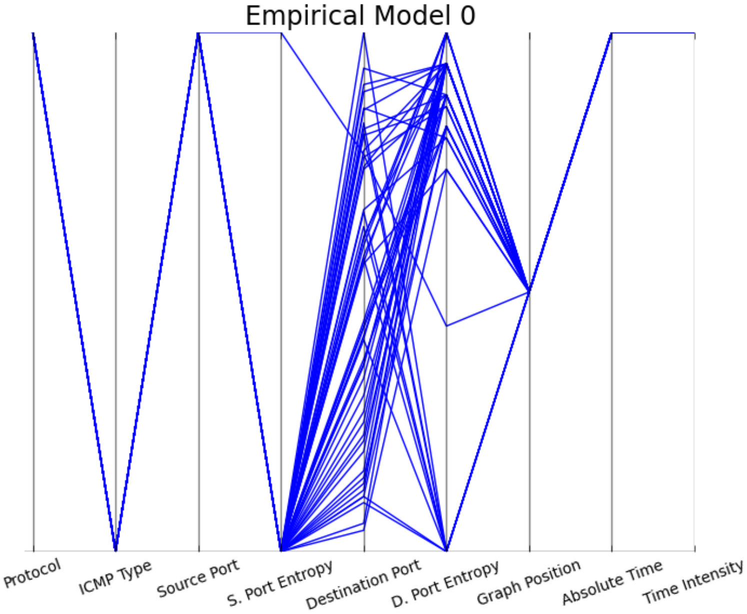


Figure 4.5: Parallel Coordinate Plot for Empirical Model 0 Feature Space

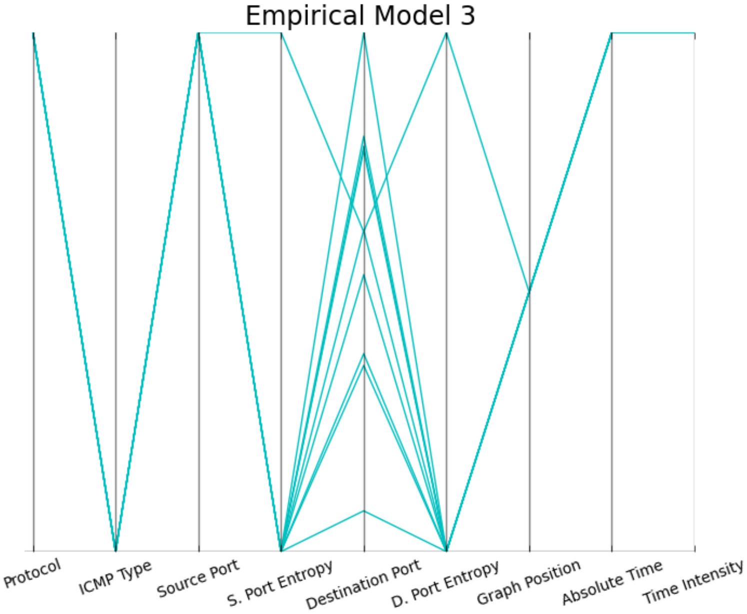


Figure 4.6: Parallel Coordinate Plot for Empirical Model 3 Feature Space

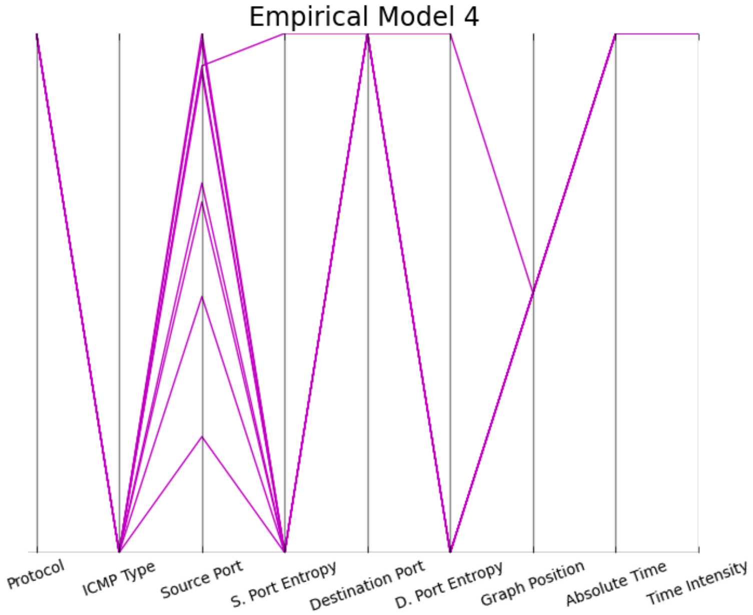


Figure 4.7: Parallel Coordinate Plot for Empirical Model 4 Feature Space

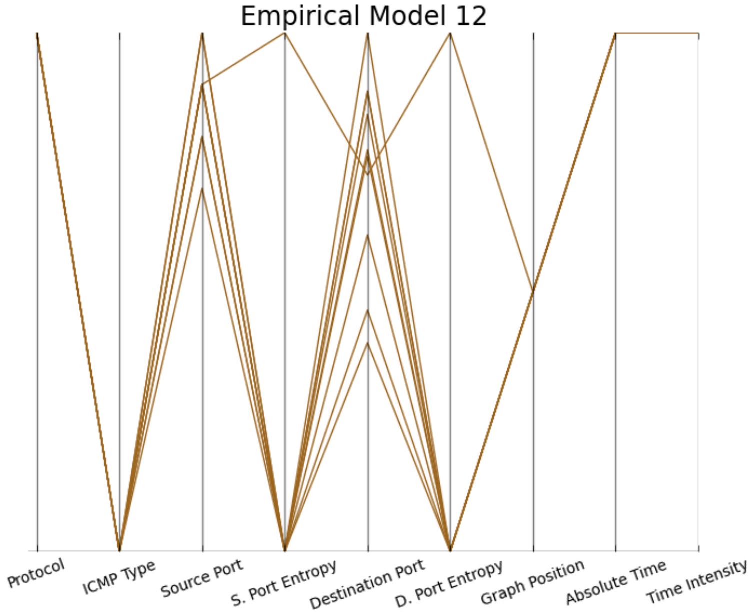


Figure 4.8: Parallel Coordinate Plot for Empirical Model 12 Feature Space

Figures 4.5, 4.6, 4.7 and 4.8 above represent some example feature distributions from this experiment. Each plot shows the set of features utilized while data was being processed. The feature space of each plot is very distinct and unique which underscores the performance in this case. In particular, the entropy of the source and destination ports varied between each model which helped to isolate the different attack behaviors.

4.2.2 Performance of Feature Groups with CAIDA Data

The other groups of features did not perform as strongly as compared to the "All" case when they were all working together. The temporal group, for example, performed poorly in terms of identifying the variety of attack behaviors incident on the network, but was able to confidently determine which model any given sample belonged to.

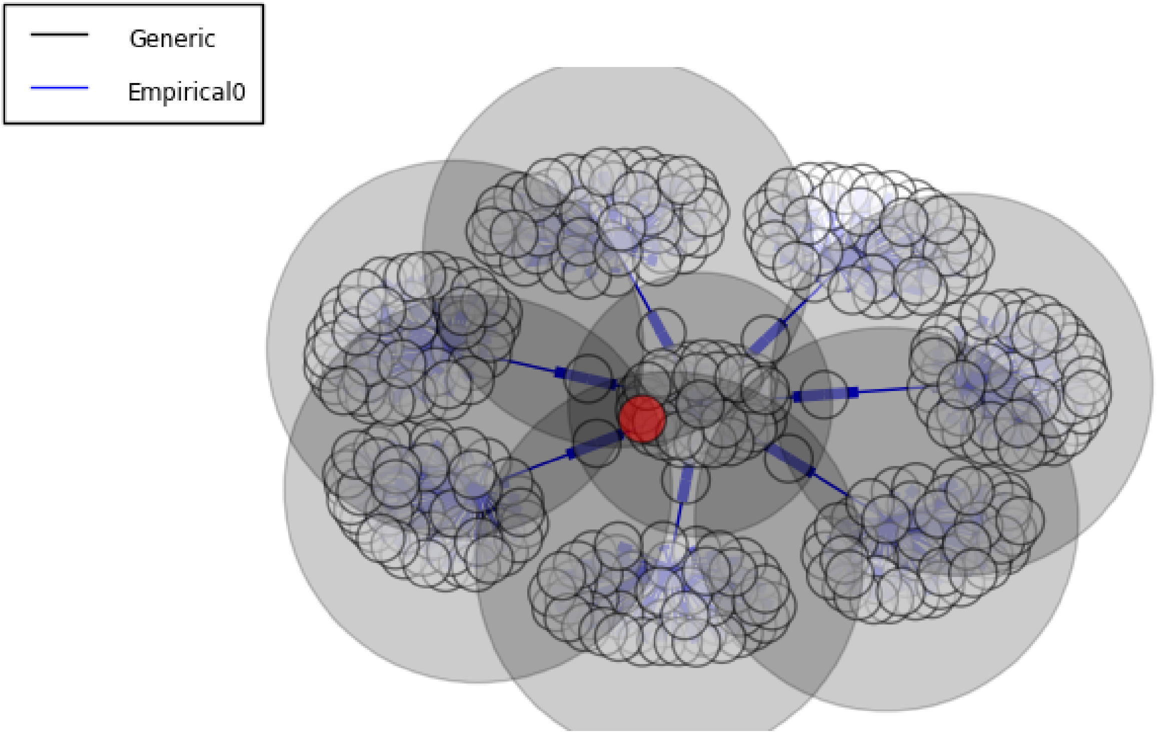


Figure 4.9: ASG Generated from Temporal Features on CAIDA Data

Figure 4.9 above shows the ASG generated from the group of temporal features on CAIDA data. This graph more closely resembles the naïve ASG shown in Figure 4.2 than the more ideal version seen in Figure 4.3. Investigating the feature space, it becomes clear that there was not enough variation with respect to time to visualize distinct attack behaviors. This is an example where the information from these features may overlap and be unable to define clusters in the feature space as clearly as warned by [9].

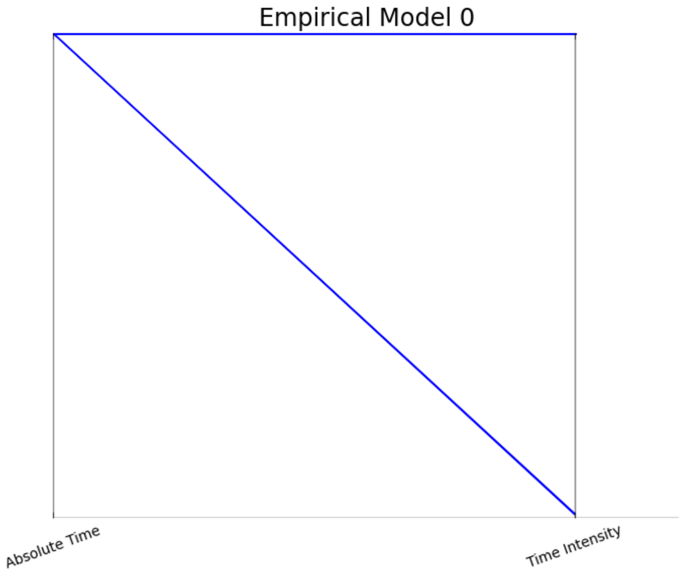


Figure 4.10: Empirical 0 Model of Temporal Features on CAIDA Data

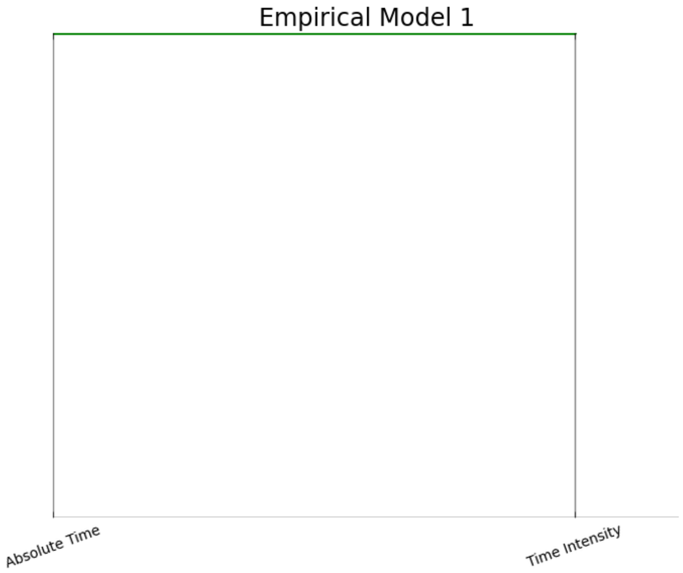


Figure 4.11: Empirical 1 Model of Temporal Features on CAIDA Data

Investigation of the feature space as visualized by the parallel coordinate plots generated by the tool confirms that there is little variation between samples. This is expected with the CAIDA dataset as most samples are generated within a short time span of each other and the traffic that occurs over edges is uniform. The samples that originally belonged to the Empirical 1 model most likely either occurred at a slightly later time than the samples stored in the Empirical 0 model or had a smaller edge packet rate as evidenced by the second line heading towards the bottom of the Time Intensity axis in 4.10.

Since the ASG in 4.9 shows all packets as part of the Empirical 0 model, it can be concluded that during the EM phase, following the creation of the Empirical 1 model, that samples originally included in this group were folded into the existing model. Figure 4.12 below shows that samples were able to be clustered with high confidence and a high MAP value to the existing models. For this particular case, no edges or nodes were segmented from the ASG so the ideal consistency of 0 probability shown by the line marked with ‘o’ for the data points is expected.

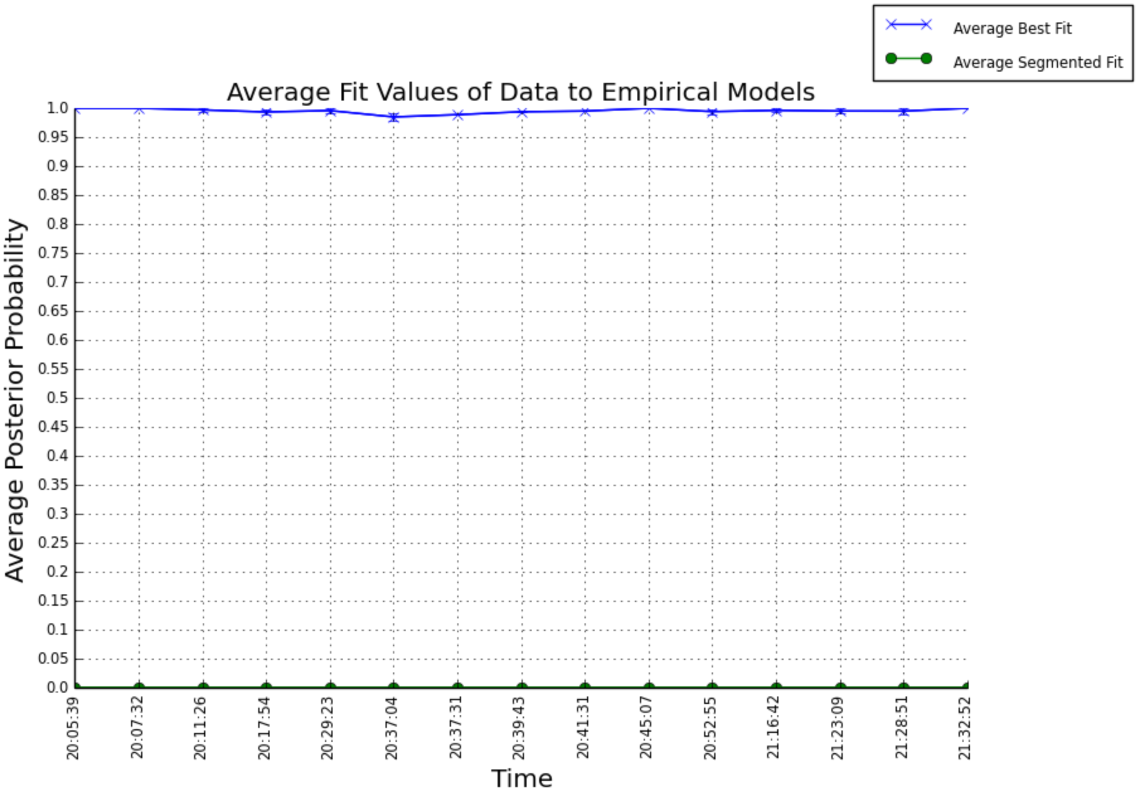


Figure 4.12: Confidence Measurement of Temporal Features on CAIDA Data

Not all cases demonstrated such ideal performance from the segmented packets however. When the original feature set was used with the updated system, it was able to perform better than some of the other groups, like “Temporal” or “Spatial”, but in terms of attack behaviors identified it was unable to distinguish attack strategies as well when compared to the baseline. The snapshot of the ASG in Figure 4.13 below shows that some additional behaviors were identified despite matching similarly to the naïve model.

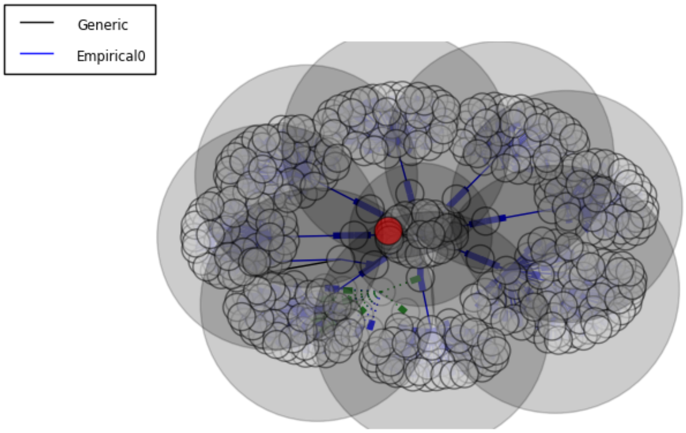


Figure 4.13: ASG of Original Features on CAIDA Data

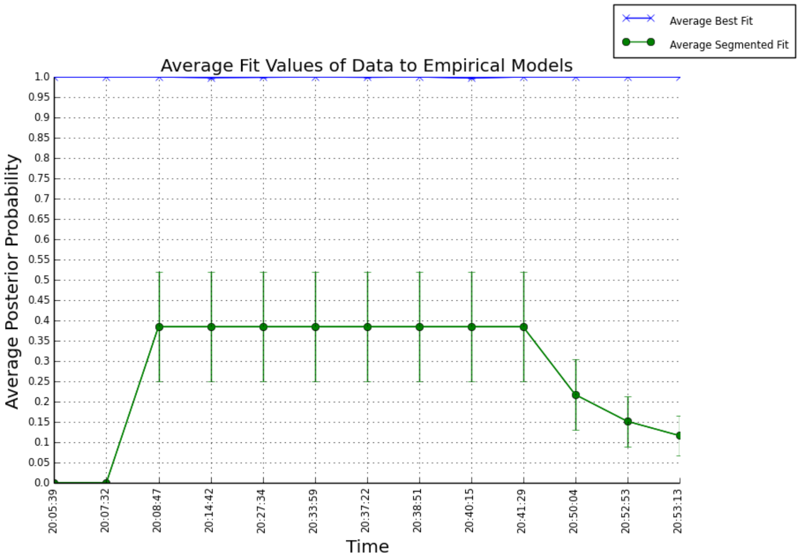


Figure 4.14: Confidence Measurement of Original Features on CAIDA Data

Viewing the confidence data shows that for active samples in the system, this original feature set was still able to classify them to models with a high MAP value. Figure 4.14 shows the confidence measurement of this feature set with the original dataset. The segmented packet samples in this case have a non-zero probability and demonstrate the use of the confidence bars which are calculated based on the standard error of the average sample MAP value. The average MAP value of the segmented samples is still far below that of the non-segmented samples further demonstrating the functionality of the segmentation logic.

For most of the segmented data points, the error bars show quite a bit of variance. This is most likely due to high variance in the behaviors captured in the Empirical 0 model. With a larger amount of variance, the confidence of the average value is lower as the true value of the population mean may fall anywhere within that range. Inspection of the feature space affirms the suspicion about the variance recorded by the models. Viewing the feature space of the Empirical 0 model given in Figure 4.15 one can see large variations in the port based features while the other active features are relatively deterministic. This is the most probable cause behind the uncertainty found in the segmented samples originally assigned to the irrelevant models found in the ASG. Figures 4.16 and 4.17 are example feature space distributions of segmented models from the ASG in Figure 4.13. The samples in these example feature spaces show significant overlap with the space defined by Empirical Model 0. There is enough difference in the non-port features to distinguish between these behaviors, but samples still have a likelihood of fitting with Empirical Model 0. The behaviors described by model 3 were identified after model 1 was segmented. This explains how they have similar behaviors, but are still distinct from each other.

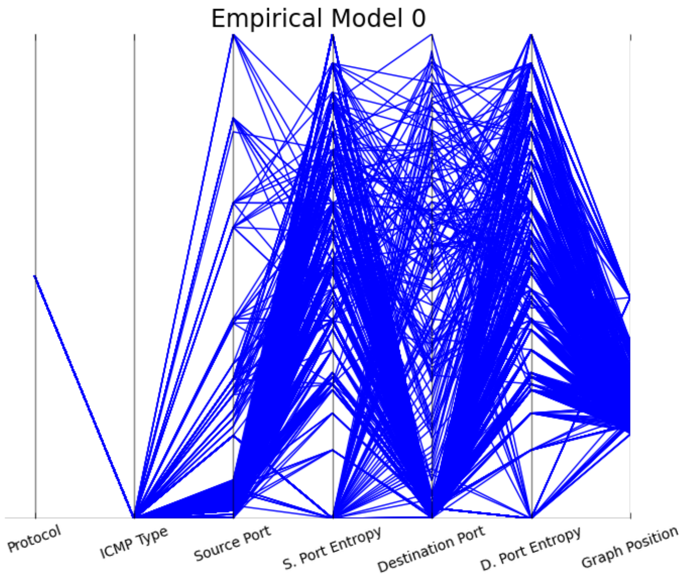


Figure 4.15: Empirical Model 0 of Original Features on CAIDA Data

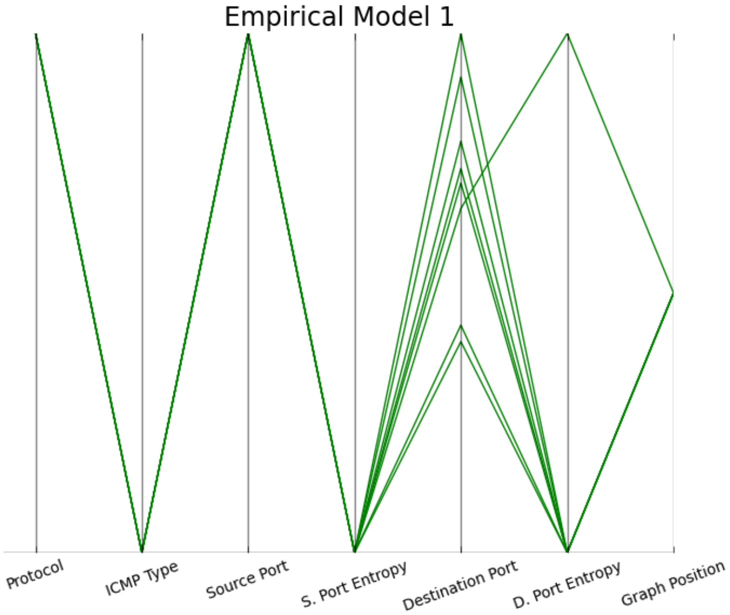


Figure 4.16: Empirical Model 1 of Original Features on CAIDA Data

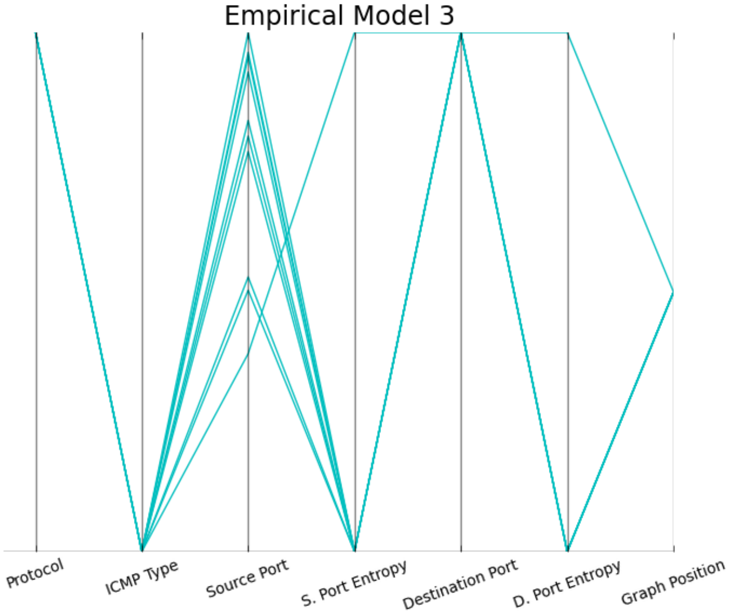


Figure 4.17: Empirical Model 3 of Original Features on CAIDA Data

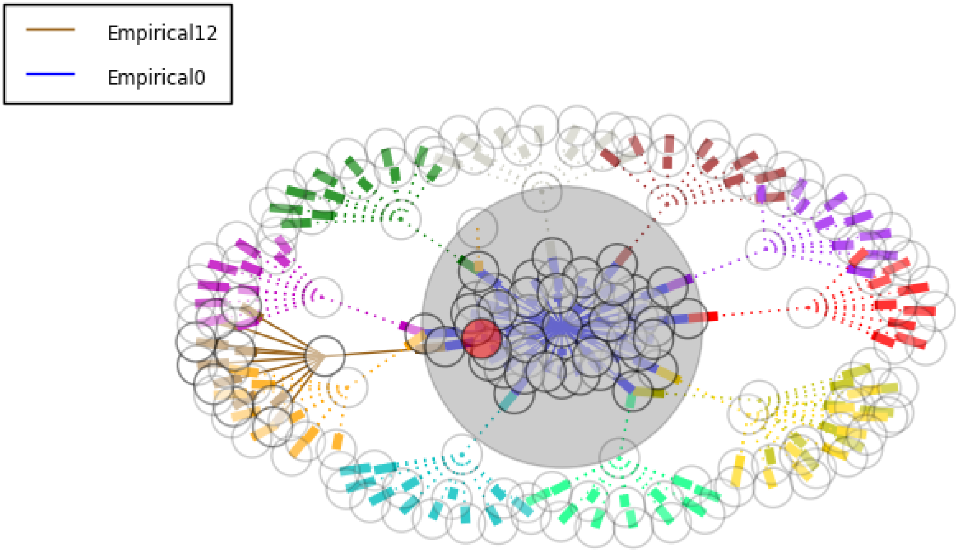


Figure 4.18: ASG of Critical Features on CAIDA Data

After comparing all of these groups, the feature set comprising the “Critical” group was chosen and evaluated. As the ASG in Figure 4.18 above illustrates, the performance seen by the full feature set is matched in terms of firmer distinctions defining the different attack behavior models. Evaluating the confidence data for this feature set, given below in Figure 4.19, confirms the high level of confidence in fitting samples into the appropriate model. The dip in the confidence data corresponds to the introduction of Empirical Model 11. This is most likely due to overlap in the feature distributions that compose each model.

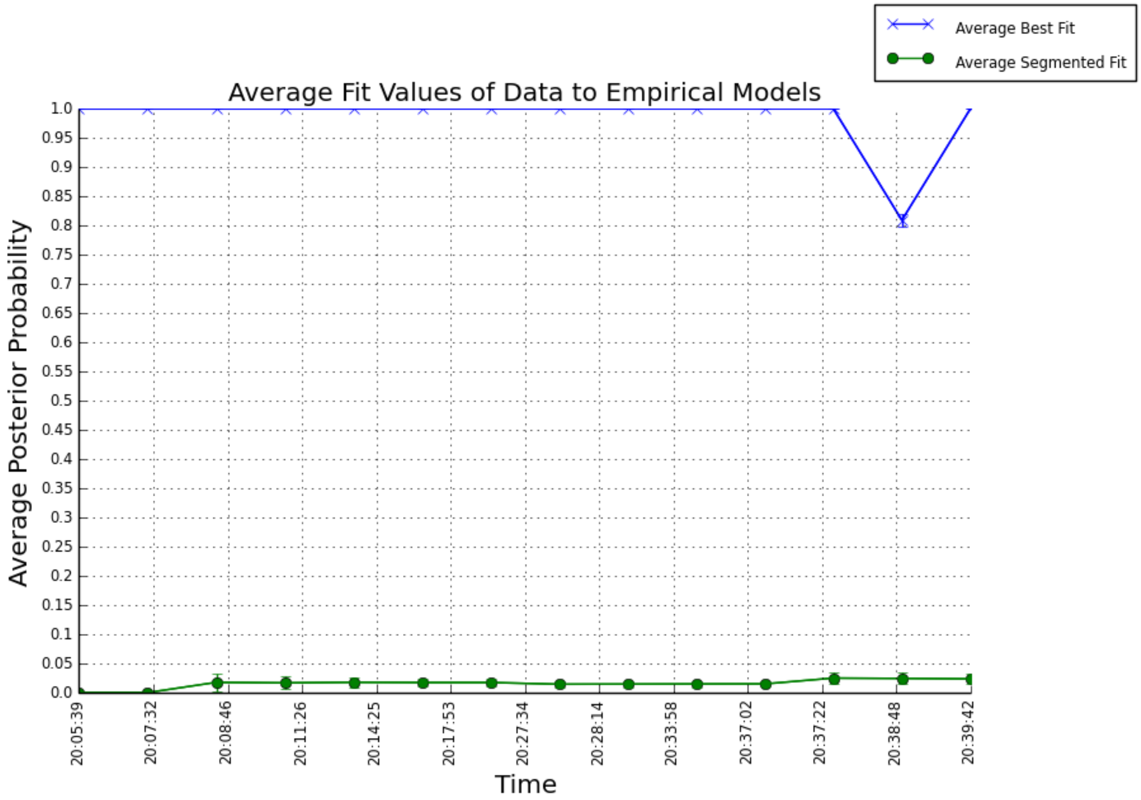


Figure 4.19: Confidence of Critical Features on CAIDA Data

Upon evaluation of the feature spaces for the active Empirical Model 0 and the space describing Empirical Model 11 shown in Figures 4.20 and 4.21 respectively, it can be seen that there is in fact some resemblance between the two distributions, particularly around the spread of destination ports utilized. This explains the drop in average likelihood as some samples will fit each model with non-zero probability. Following the segmentation of the Empirical 11 Model, the samples that remain fit with a higher average MAP value since they are no longer compared with the feature distribution of Empirical 11.

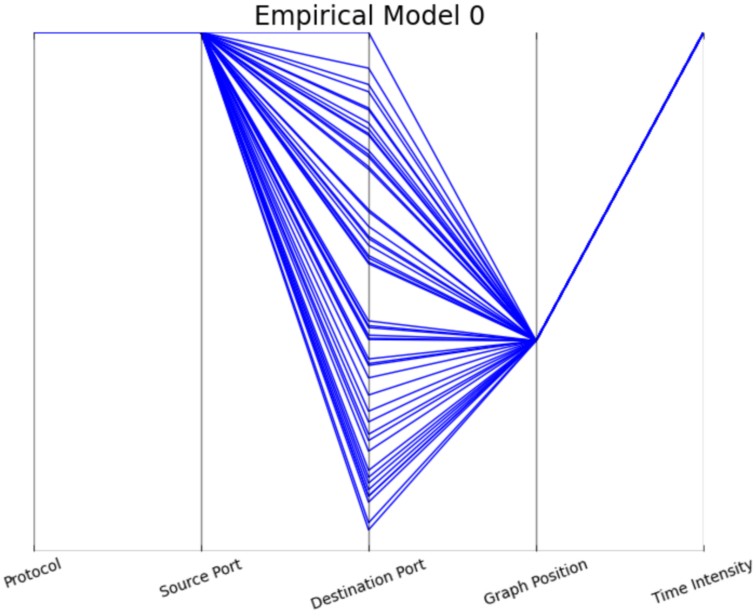


Figure 4.20: Empirical Model 0 of Critical Features on CAIDA Data

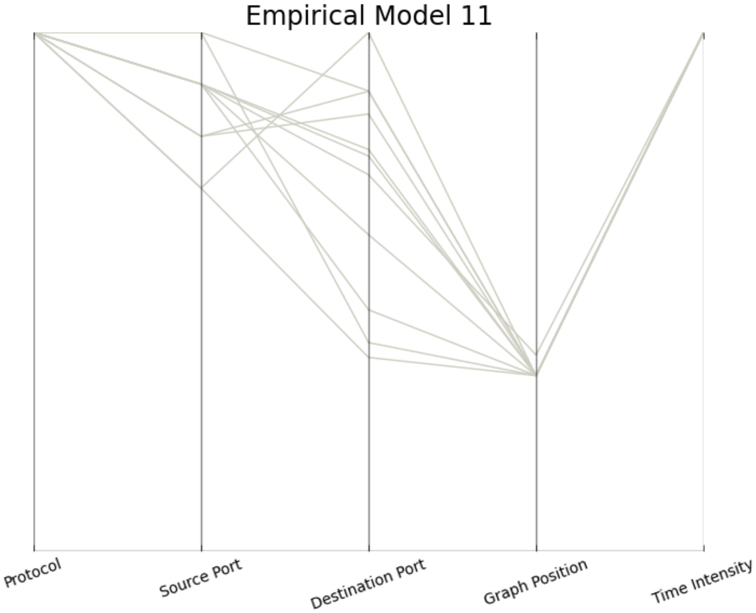


Figure 4.21: Empirical Model 11 of Critical Features on CAIDA Data

4.2.3 Performance of Feature Groups with iCTF2008 Data

With an established metric of evaluating features and generating a baseline to compare against, the iCTF2008 data was used to provide a wider array of test cases. This dataset contains a larger set of complex attack types that will change over time as teams try different tactics to complete the objectives of the game. As with the CAIDA dataset, the full set of features was tested against a target of interest selected from the iCTF2008 dataset. The target was chosen to maximize the use of both input sources, packets and alerts. Figure 4.22 below shows the ASG produced from the full feature set being utilized. Since ICMP information was not included with this dataset, the ICMP Type feature was deactivated. Figure 4.23 below shows the naïve model produced based on the same assumption used earlier.

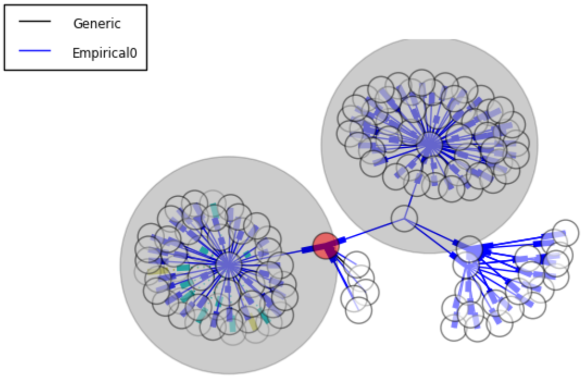


Figure 4.22: ASG of Relevant Features on iCTF2008 Data

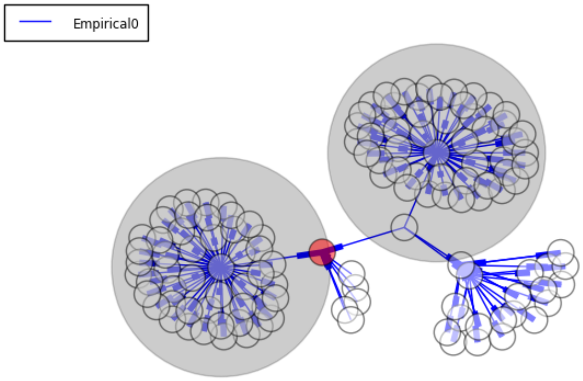


Figure 4.23: Naïve ASG from iCTF2008 Data

Unlike the CAIDA dataset, the iCTF2008 dataset with the full set of features does not serve to fully distinguish the ASG from the naïve approach which bundles all attack behaviors together into one model. In this case, much of the traffic is bundled into one attack model, but it is clear that during the experiment other behaviors were identified and segmented out from processing. Figure 4.24 below shows the confidence of the tool in placing samples into their respective models.

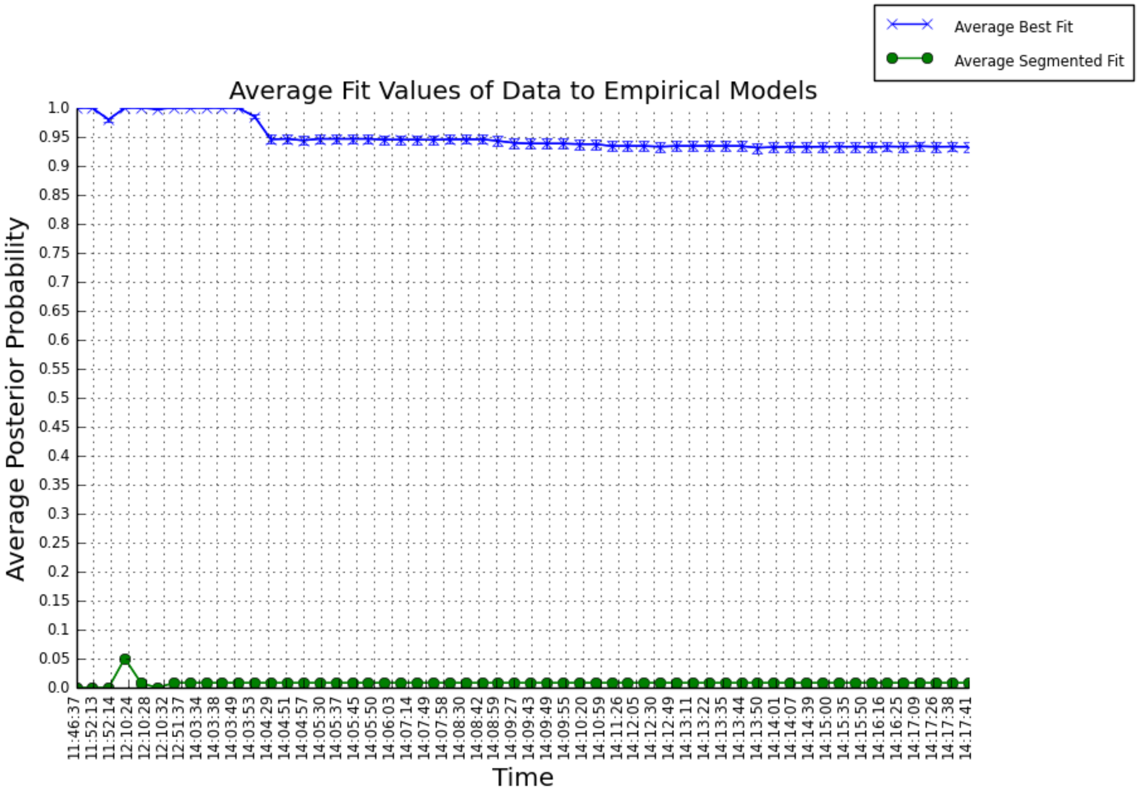


Figure 4.24: Confidence Data of Relevant Features on iCTF2008 Data

Throughout the run of the experiment, it can be seen that the relevant feature set is able to perform well. As multiple attack behaviors are involved from different sources that change throughout time, it is expected to see a small drop in performance as time goes on. This is because models will update their feature distributions putting more weight on the newer samples. Older samples will not fit as well to the current version of the model resulting in the drop seen.

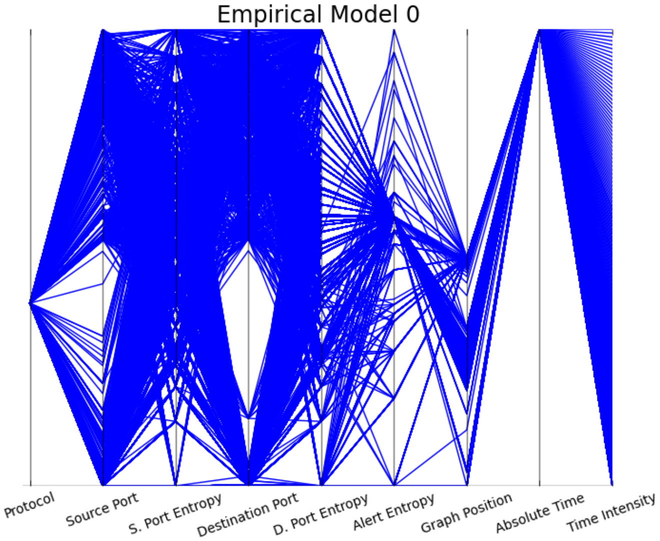


Figure 4.25: Empirical Model 0 of Relevant Features on iCTF2008 Data

Figure 4.25 above shows the large distribution of values seen for each individual feature. This increases the possibility that all future packets will fit within Empirical Model 0 even if they describe a behavior that would benefit from having its own empirical model. After applying only the features within the "Critical" set, an increase in performance can be seen. Figure 4.26 below shows the attack social graph generated by using the reduced feature set. With these features enabled, the ASG has evolved more over time than the baseline in Figure 4.22.

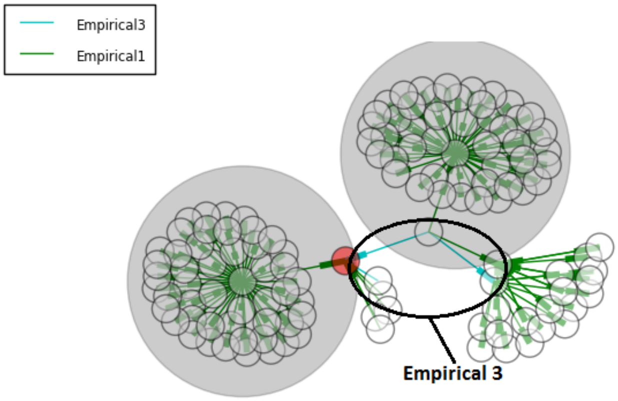


Figure 4.26: ASG of Critical Features on iCTF2008 Data

Evaluation of the confidence data, seen in Figure 4.27 below, shows that with the addition of new edges in time, the average MAP value of the samples begins to drop. Unlike with the full set of active features, the critical features are able to recover in time with the addition of a new empirical model at time 14-17-39. With the addition of the new model, a subsection of the behaviors originally associated with the larger Empirical Model 1 were now held in their own model. This reduced the diversity of the behaviors within the original model allowing for a better fit of the samples that belonged to each.

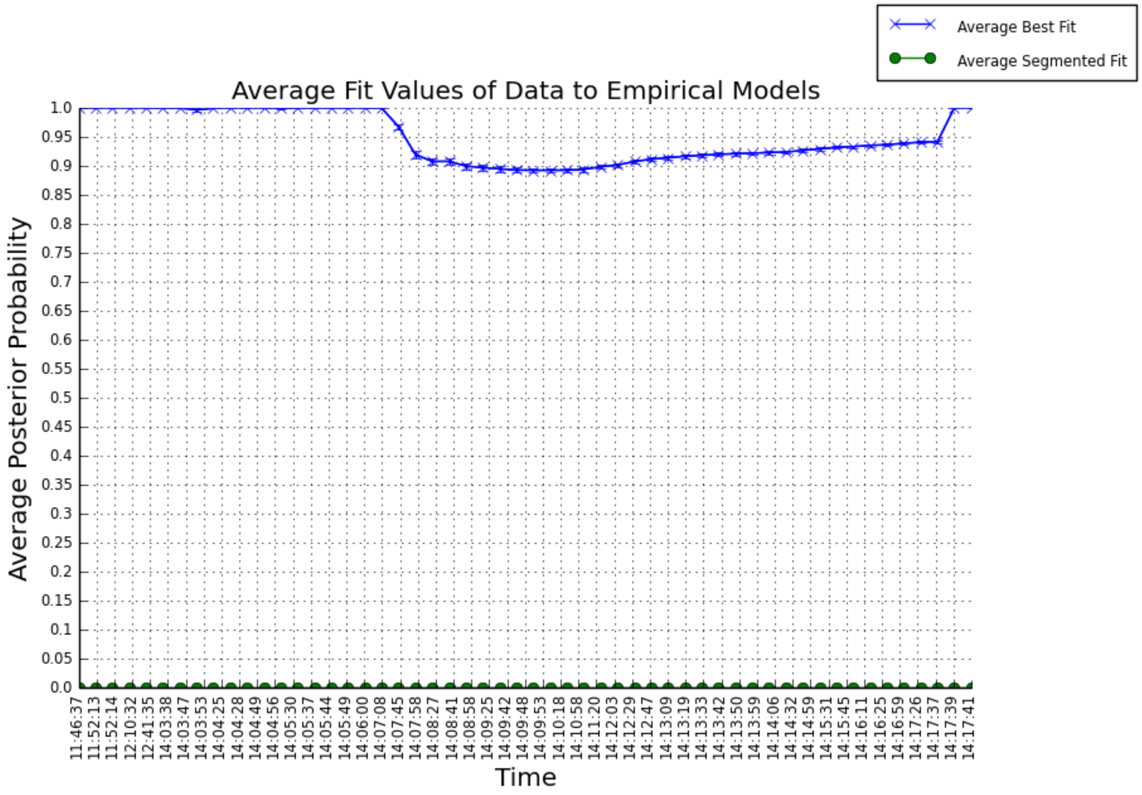


Figure 4.27: Confidence Data of Critical Features on iCTF2008 Data

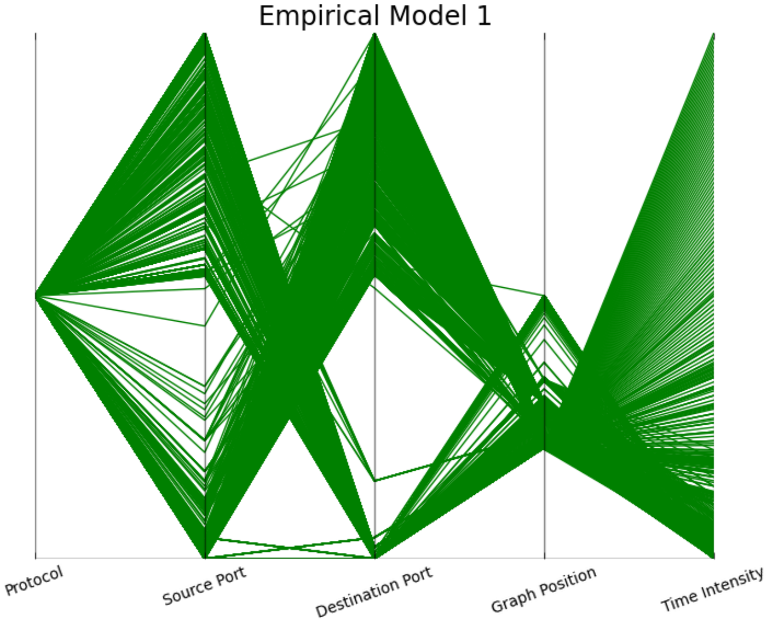


Figure 4.28: Empirical Model 1 of Critical Features on iCTF2008 Data

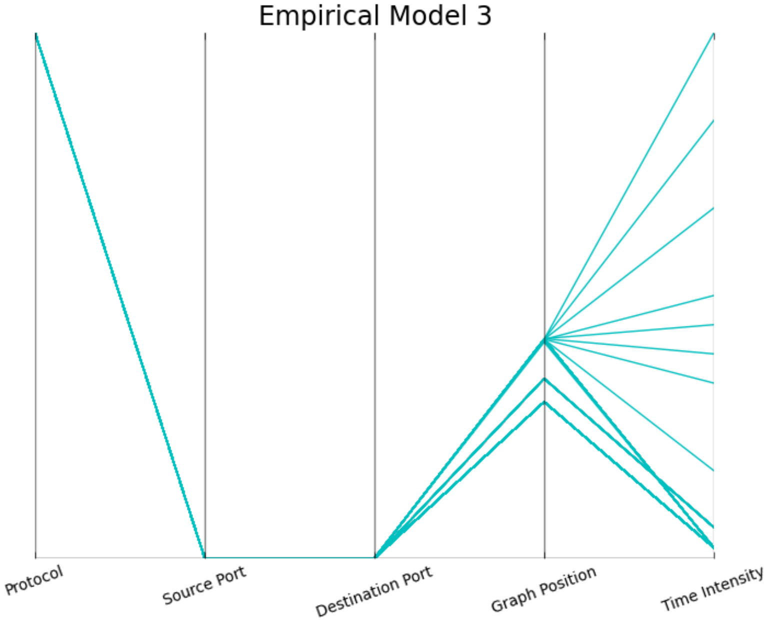


Figure 4.29: Empirical Model 3 of Critical Features on iCTF2008 Data

Evaluation of the feature space generated by the "Critical" group of features confirms that samples associated with a subset of the edges with varying rates of packet traffic and portions of the graph position. Evaluation of the CSV file storing the confidence data shows that the use of the ICMP protocol was a distinguishing factor from the variety of protocols used earlier in Empirical Model 1. This type of traffic makes sense as participants to the game used a variety of tools to connect to and interact with the network topology provided by the game developers. Participants are expected to connect to the main network box through a VPN connection and then scan the network topology to find vulnerable sources. Reconnaissance actions such as these typically employ the use of ICMP packets to scan the network to see what devices are connected. Tables 4.2 and 4.3 below show sample data taken from the CSV files used to generate the parallel plots seen above in Figures 4.28 and 4.29. The distinction in traffic such as this represents the success of the framework employing the "Critical" feature set to identify the different behaviors.

Table 4.2: Sample of Feature Distributions for Empirical Model 1

Protocol	**Source Port**	**Destination Port**	**Graph Position**	**Time Intensity**
TCP	22	51385	0.236537148	3
TCP	54562	80	0.499985944	0.213017751
SSHv2	58862	22	0.238043551	1.938073395
TCP	58854	22	0.238088047	1.967213115
SSHv2	58854	22	0.238088577	1.906976744
SSHv2	22	53205	0.234975826	2.109375
TCP	53205	22	0.234976856	2.375

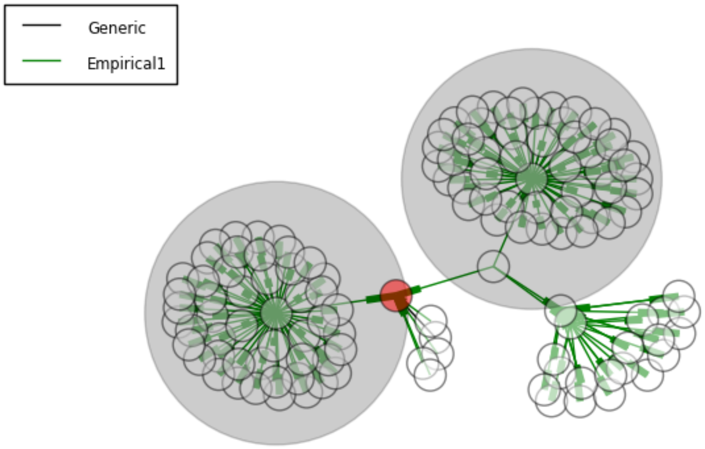
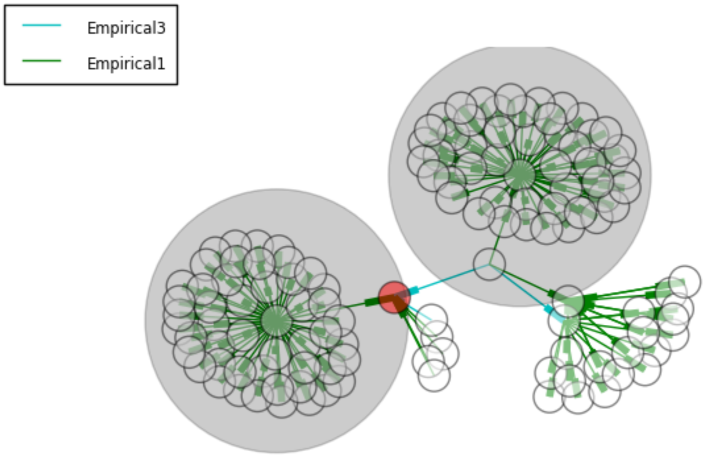
Table 4.3: Sample of Feature Distributions for Empirical Model 3

Protocol	**Source Port**	**Destination Port**	**Graph Position**	**Time Intensity**
ICMP			0.297619033	0.120401338
ICMP			0.297619033	0.120401338
ICMP			0.297619033	0.120401338
ICMP			0.341904735	0.352941176
ICMP			0.297619033	0.120401338
ICMP			0.297619033	0.120401338

The remaining feature groups proved to perform similarly to the naïve model fitting all behaviors to one model with high confidence. An interesting exception was the set of original features. The original features were able to produce an ASG similar to the one seen in Figure 4.26 which was produced from the critical features, but with much less confidence. Comparing this to the set of new features, the ASG produced showed a much more naïve representation of the features, but with a higher level of confidence.

Considering the features composing the "Critical" group is a mixture from both of these sets, it promotes the strength in using distinct, non-overlapping features to best describe the different dimensions of the data. Table 4.4 below shows a comparison of the two ASGs generated from each of these groups.

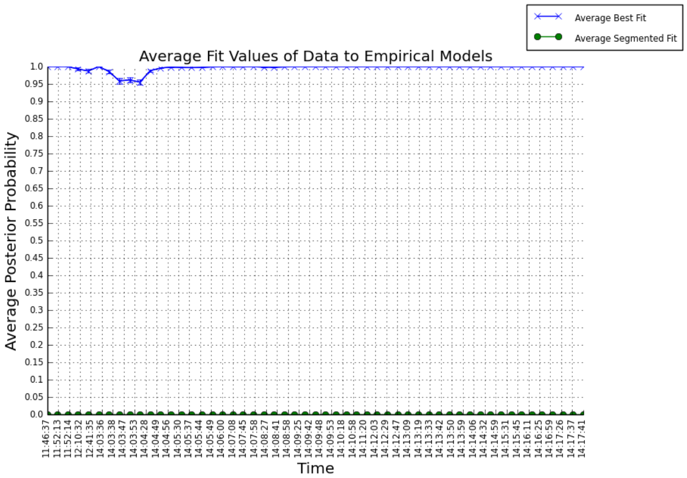
Table 4.4: Comparison of ASGs Generated from "New" and "Original" Feature Sets

New

Original


Reviewing the confidence data shows the strengths and weaknesses of each group set. The iCTF dataset is very prone to experiencing a number of behaviors through time where as the packet and graph structure are not as likely to change. The newer features are more based on temporal descriptions of the data while the original set was more centered around specific information found within the packet headers and their spatial positions. This most likely explains how the temporal features within the "New" group can classify with higher confidence while features from the "Original" group are better at distinguishing patterns within the traffic.

Table 4.5: Comparison of Confidence Data from "New" and "Original" Feature Sets

New



Original

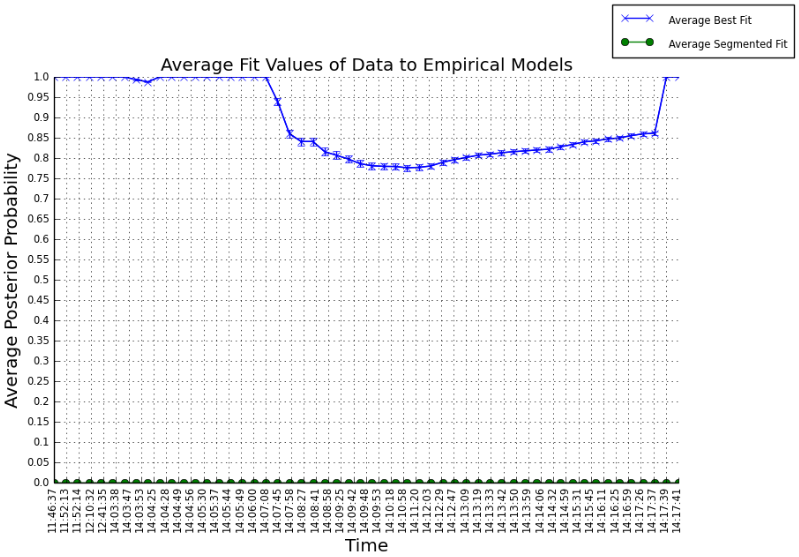


Table 4.5 above shows a comparison of the confidence data generated by each of these groups. While the original features are able to recover their confidence towards the end of the experiment, with the introduction of a new model, the majority of the time samples

have a lower average MAP value overall. Referring back to Figure 4.27, the pattern where the overall confidence dips and recovers is very similar to that produced by the features in the "Original" feature set. The overall confidence of the critical features dips much less than that seen by the set of original features in Table 4.5 most likely due to the stronger confidence seen with the "New" feature set. By utilizing the stronger features from each set, the graph in Figure 4.27 asserts the validity of using only the strongest components from the available types of features. Using a reduced feature set composed of only the principle components shows that better overall results can be obtained.

4.3 Behavior Interpretation

The other critical aspect of the darknet analysis framework is its ability to produce a visual to allow for the interpretation of the events that occurred within the network. It is important to distinguish that this framework is not designed to explicitly recover all events that transpired, but to present as much information as possible to a network analyst or a system administrator. One of the main reasons for switching to the capture the flag dataset was because of the larger amount of documentation that was available to describe the topology and expected cyber attack behaviors from participants. Figure 4.30 below represents the network topology described by the game designers [3].

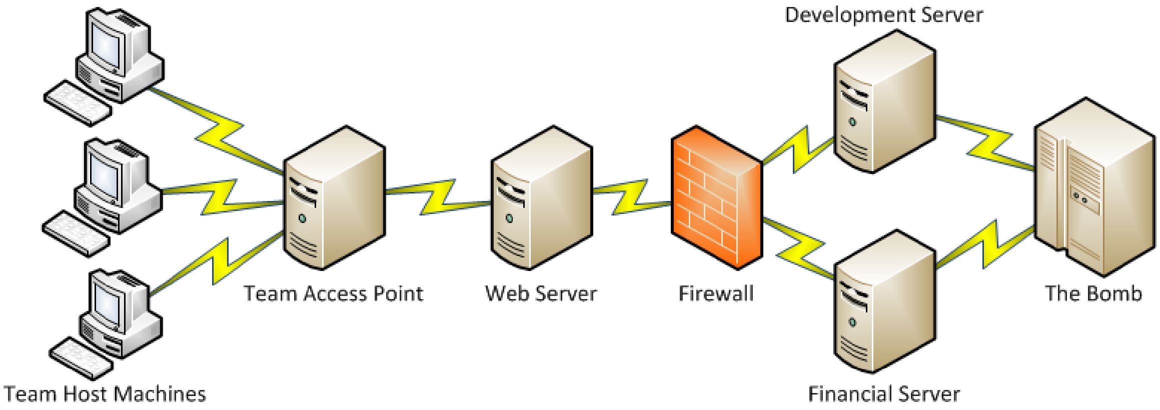


Figure 4.30: Network Topology for iCTF2008 Game

Each team was given a virtual copy of the network shown above, however all traffic leaving the team's personal subnet was required to be routed through a central server for data collection. The primary objective given to each team was to find and shut down a server known as "the bomb" which would go off at the end of the game. Teams were expected to gain access to a web server located at an IP address provided to all teams and exploit a series of vulnerabilities to open up new pathways to the internal network. Once teams gained access to the web server, they were to scan the network and find other vulnerable hosts that they could then exploit and island hop to. An island hop is when an attacker gains access to a target machine and then proceeds to continue a multistage attack from this acquired machine. In essence, an attacker is hopping from target to target until they achieve their goal or are caught and blocked from the network. This process would continue until the bomb server was reached and could be shut down.

Given this scenario and topology, it would be expected that early on in the time line of the game that patterns related to collaborators scanning the network topology would be identified. Since a common server was used to collect metrics about traffic on the network topology, it would be expected that IP addresses found within range of the address space given to the teams would be identified. Each team was given an unroutable address in the form of $10.TeamNumber.1.0/24$. There were 39 unique schools with teams participating with one school having two teams. Given this, the value for $TeamNumber$ ranged from 1 to 40. Since the framework converts IP address strings (where each digit is a base 256 number) into integers for more efficient computation, integer values beginning with "16..." are found to be within range of the participant's address space while values beginning with "17..." represent the unroutable addresses that were not specified by the owners of the dataset. No ground truth was provided detailing specific attacks used, what time teams completed objectives or even the addresses assigned to the machines detailed in the topology diagram. Due to this, attack behaviors and targets of interest selected are based purely on inference given the overview provided from the game developers and from examination of the source data.

The targets of interest used during the experimentation of feature evaluation on the iCTF2008 dataset were used as a starting point to examine the types of behaviors incident on the network. As the ASG developed, probable targets were chosen based on the traffic flow connecting them to other nodes in the network. Figure 4.31 below is a snapshot collected during the evaluation of critical features of the iCTF dataset. In this image, highly-active elements have been identified as indicated by the larger gray circles encompassing the various clusters. A highly-active element is defined as an attack behavior that generates edges at a very high rate in which a node has an inbound degree or an outbound degree of one [20]. Behaviors that are likely to cause this are spoofed addresses from DoS attacks and random fast scans.

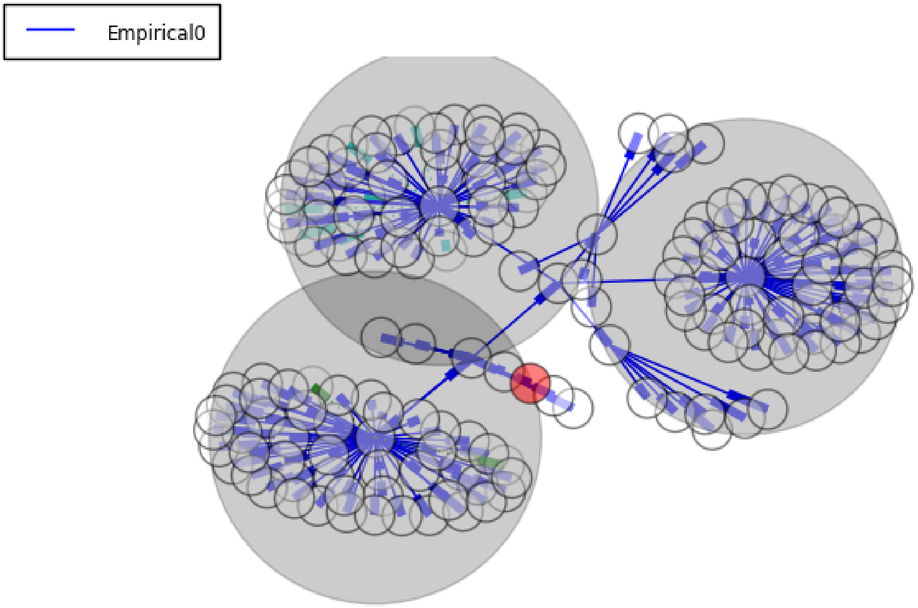


Figure 4.31: Full ASG with Scanning and Island Hopping

Figure 4.31 above shows the full topology of the ASG from which the following example of a multi-step behavior was observed. This ASG shows a high level view of the captured behaviors on this network topology. From this perspective it is difficult to interpret these behaviors and understand the situation. Figures 4.32, 4.33 and 4.34 are closer looks at this ASG showing that there is evidence to support the framework's ability to capture and display complex cyber attack behaviors. When a context is provided describing both the network topology and goals of the attackers, it becomes possible to infer and identify the events that may have occurred on the network. Combining the information acquired from the given context with the intuition of the expected behaviors it is possible to closely examine the ASG as it forms over time to verify that the models resemble expected behaviors. Following the behaviors generated from a single team, it is possible to verify the steps being taken to achieve the final goal of the game.

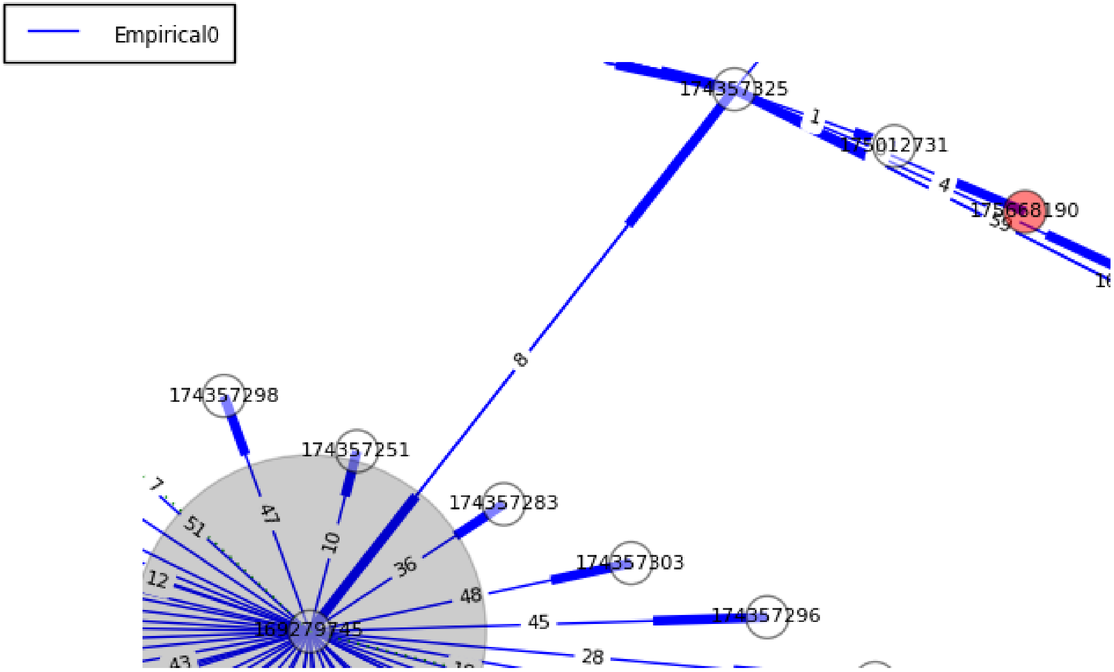


Figure 4.32: Network Topology Scan iCTF2008

Given this information, the node at the center of the gray area in Figure 4.32 is likely performing a vertical scan looking for other machines connected to the network. The address seen at the center of the gray area, node 169279745, is the integer form of the IP address 10.23.1.1 which is allocated to the team "We_0wn_Y0u". Addresses beginning with the digits "17..." can be seen as destinations from this team's address which suggests that team "We_0wn_Y0u" is performing a vertical scan. The next expected phase would be to see an edge from a "16..." address connecting to a "17..." address with further extensions signifying a potential island hop in the traffic. Edge 8 stands out in this image because it shows a double ended arrow between the highly active address and what is most likely the access point at 174357325, or 10.100.123.77.

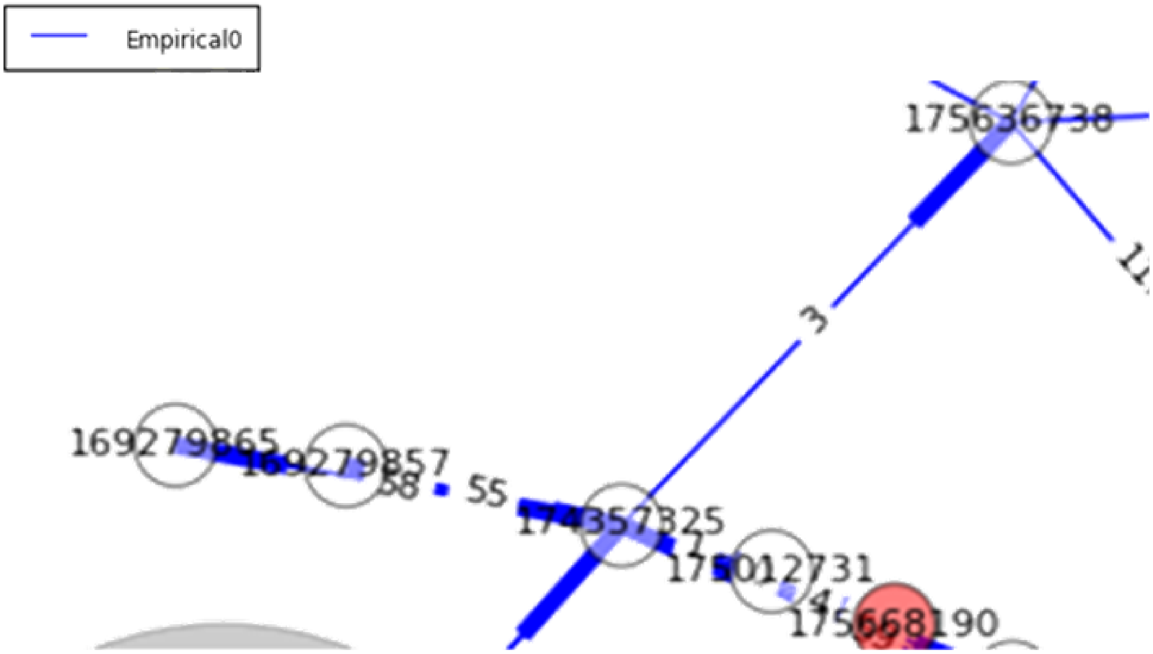


Figure 4.33: Island Hop Example

Figure 4.33 above centers around access point 174357325 to evaluate actions taken beyond the initial connection to the topology. Connections from addresses 169279865 and 169279857, or 10.23.1.121 and 10.23.1.113 respectively, also belong to the address space of the team “We_0wn_Y0u” further establishing node 174357325 as an access point to the main network topology for this team. In addition to these host nodes, edges connecting to other “17...” addresses from the team can be seen which is indicative of scanning. The traffic leaving the access point is trying to find a significant resource to exploit and control. Further investigation confirmed that these edges had only one or two instances each which is indicative of a vertical scan. Edge 3 in Figure 4.33 shows an outgoing connection to an additional “17...” address (175636738). Unlike the previously seen “17...” addresses, further hops were identified following the hop to 175636738. Focusing on this edge shows early signs of hopping from one resource to additional resources on the network in the search for “the bomb” server.

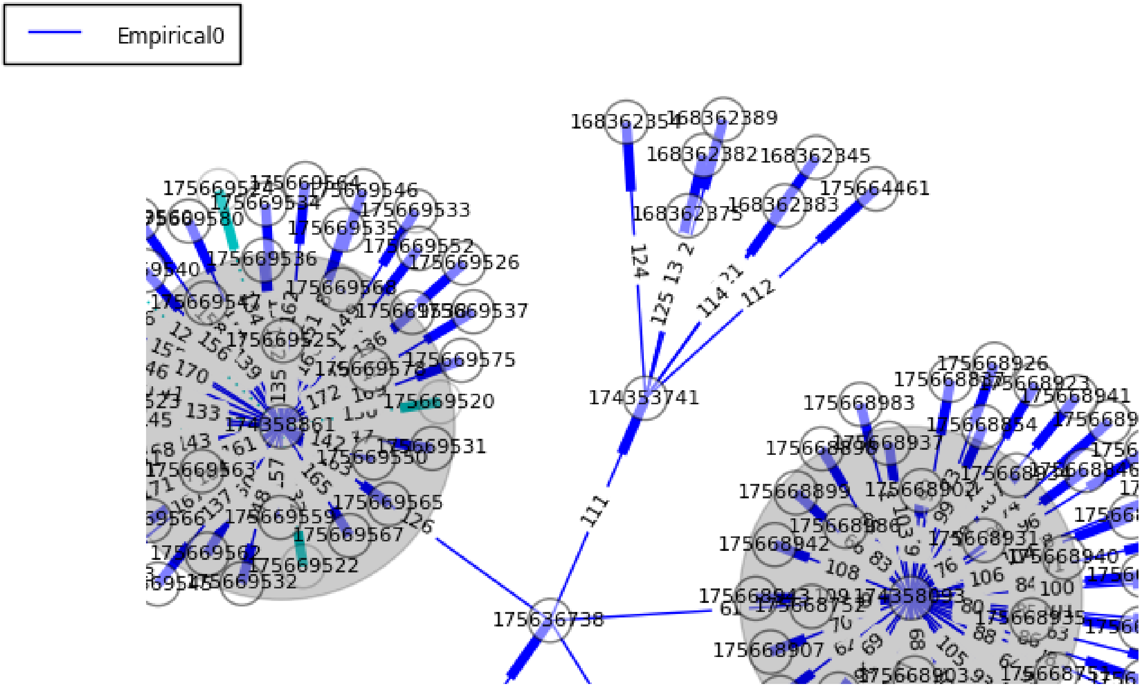


Figure 4.34: Second Stage Island Hop Example

Figure 4.34 above shows an example of a second stage island hop in the network topology. In this snapshot of the ASG, highly active nodes can be observed including address 174358861 (10.100.129.77) in the upper left hand corner and 174358093 (10.100.126.77) in the lower right hand corner. Outgoing edges from the previous address 175636738, or 10.120.1.2, connect to these highly active nodes as well as a node with a smaller amount of traffic in between these two clusters. These edges display evidence that members of the team “We_0wn_Y0u” successfully compromised the previous node to traverse the network topology. This is supported by the image because each of the outgoing edges connect to other “17...” addresses which fan out to other nodes not within a participating team’s address space. The pattern of scanning and hopping to resources expected from this dataset is demonstrated by the progression of behaviors synthesized by members of the “We_0wn_Y0u” team. The middle, non-highly active node appears to be an access point for the team “squareroots”. Outgoing edges from this access point include “16...” IP addresses for hosts from the “squareroots” team and other “17...” addresses. This is an important observation because it shows that teams have the ability to scan and observe the access points and address spaces of other participants. Combining this observation with the fact that the collection point includes traffic generated from all teams means that the feature space for this ASG will incorporate a variety of strategies.

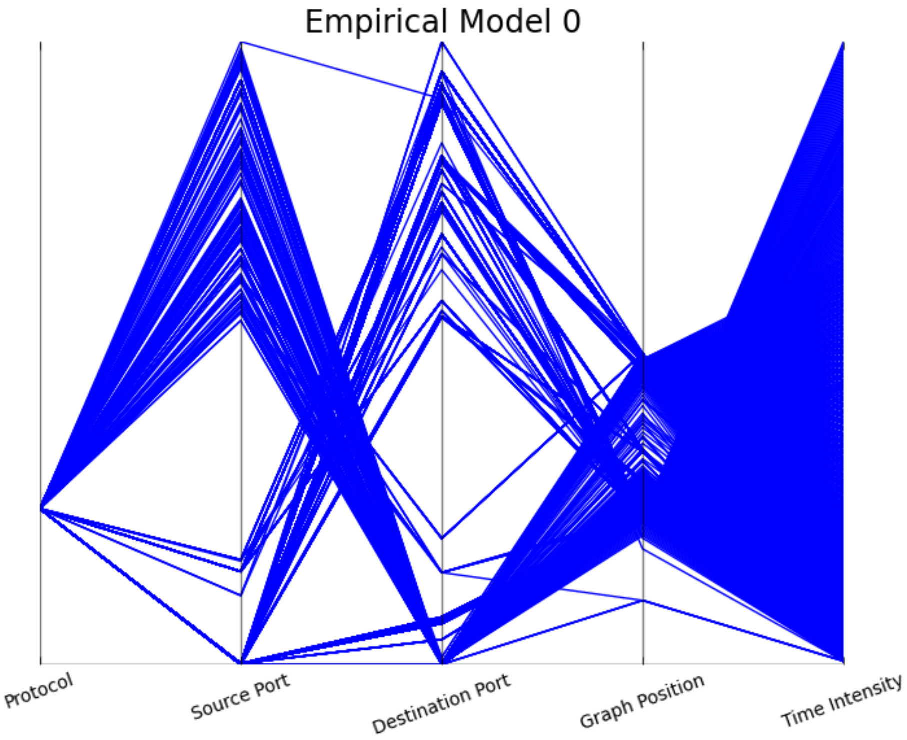


Figure 4.35: iCTF2008 Empirical 0 Model Feature Space

Figure 4.35 above shows the parallel coordinate plot for the feature space composing the larger Empirical Model 0 seen within the images earlier. As predicted, many dimensions of the feature space hold a large variety of values. The Protocol feature is the most homogeneous with only a couple of values observed, compared with the Time Intensity feature dimension which is very diverse. Having a heterogeneous model such as this one means it will act like a “blob” and incorporate multiple behaviors. Additional behaviors were identified while observing traffic for this experiment, but they were quickly segmented out. These behaviors were most likely synthesized after a small amount of observations existed within the Empirical 0 model and were later found to not have a strong path to the target of interest. Figures 4.36 and 4.37 below show the feature space of these segmented behaviors. The protocol and port values are the likely differentiators in these cases compared with the communication methods captured in the more diverse Empirical 0 model. These segmented behaviors are seen emanating from highly active nodes in Figure 4.34. It is likely that other teams were scanning or communicating over similar edges at different times which distinguished the behaviors on these edges. Unfortunately, without a ground truth to compare against, it is impossible to fully verify the accuracy of the framework and tool to display a representation of the events that arose on a network. However, given the previous examples, there is strong potential for an experienced user to be able to interpret the information and react accordingly based on the behavior of the traffic incident on their network.

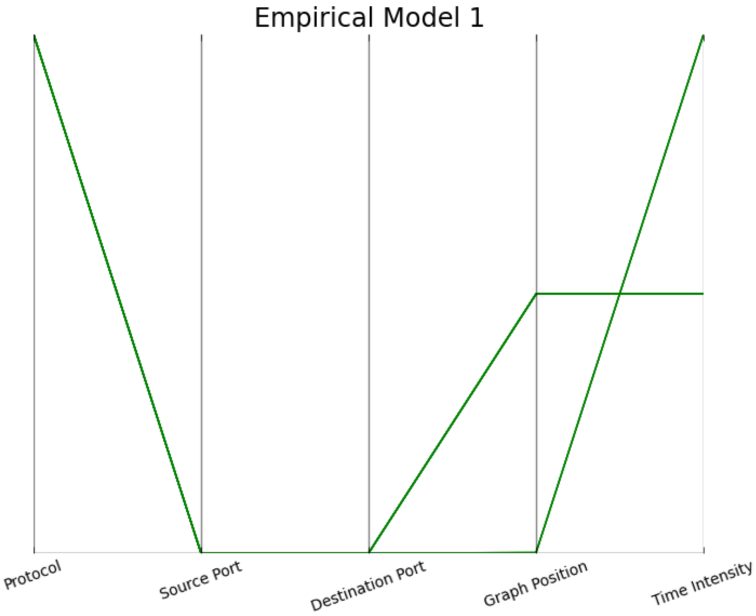


Figure 4.36: iCTF2008 Empirical 1 Model Feature Space

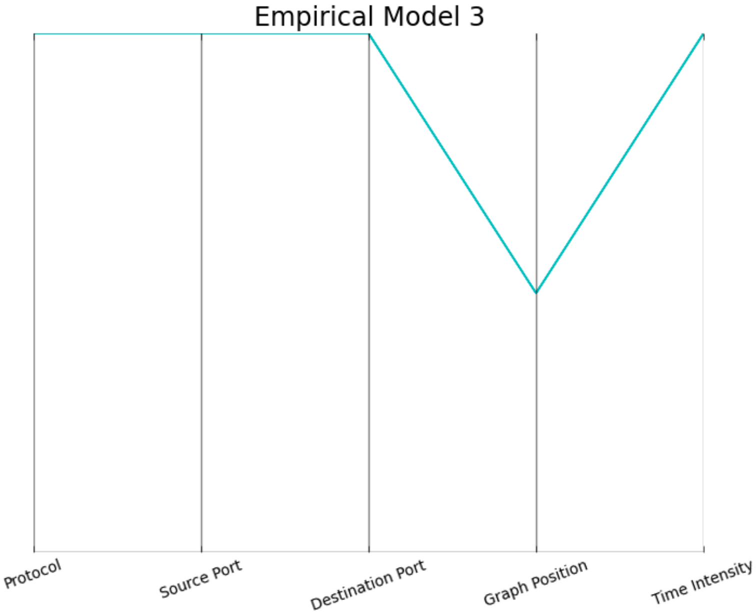


Figure 4.37: iCTF2008 Empirical 3 Model Feature Space

Chapter 5

Conclusions

The original darknet analysis framework employed the novel Attack Segmentation and Model Generation algorithm to create a semi-supervised online learning mechanism. While this algorithm proved successful at identifying and describing attack behaviors, there were limitations in the design of the tool. The principle contributions towards enhancing these limitations includes the implementation and evaluation of an expanded feature set, the ability to scale the size of the input data and revising the logic driving the introduction of new empirical models.

Previously, features were selected to describe the dataset based on limited information provided from packet headers. This scarcity of available information was due, in part, to the dataset utilized for evaluation of the original implementation of the tool. The Cooperative Association for Internet Data Analysis has restrictions in place on many of their datasets to anonymize the information captured to reduce issues related to liabilities. This coupled with the limited documentation available make verification and evaluation of features difficult. These difficulties led to the use of the international capture the flag (iCTF) dataset created in 2008 provided by the University of California, Santa Barbara (UCSB). Switching to this new source of data provided several advantages over the original CAIDA dataset. These advantages included no restrictions on the use of the data, alert information produced by the Snort IDS and limited documentation describing the background scenario guiding the actions of the participants of the game.

Using iCTF data allowed for the inclusion of a secondary dataset as well as context to compare identified attack behaviors. Having a context to compare against, even if it is not a true ground truth, allows for a deeper evaluation to be conducted on the performance of the system. This context also allows for the identification of a principle set of features which provide the best possible classification of samples in any generic dataset. The additional source of input data, IDS metadata, served to successfully allow the framework to analyze a larger pool of information. This expansion of information helped to better characterize and reconstruct some of the events that transpired on the network.

An expanded feature set was also possible due to the new input information available. The expanded feature set provided valuable dimensions to describe samples and further characterize network attack behaviors. The iCTF2008 dataset offered the opportunity to test the framework against a wider range of attack behaviors. The availability of extra test cases exposed weak areas in the design of the tool and framework allowing for a more robust implementation to be produced.

5.1 Future Work

Development of the enhanced implementation of the darknet analysis framework showcased several opportunities for future work. While evaluating the network topology described by the ASG, it was observed that several edges were segmented from processing that were part of a group most likely detailing a vertical network scan from an attacker. The edge itself may have been irrelevant at the time of segmentation, however in a network where behaviors can change drastically over time, these edges may become important or relevant again in the future. Under the current implementation, once an edge is segmented from processing, future traffic related to the edge is ignored. The reintegration of segmented edges has the potential to provide a more dynamic means of analyzing the information presented by the ASG and puts more emphasis on current attack behaviors rather than stale models that are less relevant.

Scaling the input data types supported by the framework allowed for new features to be derived that expanded the dimensionality of the feature space used to characterize behaviors. Improving the depth and scale of the existing the framework may afford new features to surface which will further describe the variety of existing attack behaviors and allow for more specific descriptions of the attacks being observed. Sensor data from network based intrusion detection systems was incorporated to expand and improve the existing feature set defined by the framework. Other sensors, such as host-based intrusion detection systems, may provide additional and more specific information that can be utilized to enhance the feature space of the framework.

Another potential source for improvement with the system would be to experiment with different classifiers for processing samples. The naïve Bayesian based classifier has its benefits, but the assumption of independence for each observation is a potential source for performance degradation. Attack types may consist of multiple stages to accomplish the goal that the malicious user is trying to achieve. The probability of a certain sample being observed may be skewed depending on the sample packets previously seen. For example, if an attacker is looking to exploit a JavaScript vulnerability in a website, it is more probable that following packets would be attempts at JavaScript injections or other forms of cross-site scripting (XSS) behaviors than the attacker sending packets more indicative of a MySQL injection attack. The ability to skew or weight the probability of a packet belonging to a given behavior based on the current track of behaviors can potentially provide a stronger distinction between attack behaviors as they change through time.

The feature selection process is currently very manual in nature. Each dataset typically performs better with a different subset of features from the total available. Implementing a mechanism to automatically select features will help in optimizing the system to better describe empirical attack behaviors. In addition to the performance enhancement, less time will be needed prior to running the dataset through the system to find the best set of features. Going forward, these benefits will increase the overall usability of the framework. A system administrator or a network analyst can focus more on the feedback synthesized by the tool rather than spending time configuring the system and potentially missing important information. Ideally, the user will spend minimal time installing and configuring the system to their needs and maximize the time spent examining and analyzing traffic.

Removing irrelevant information is an important aspect of the attack segmentation and model generation (ASMG) algorithm powering this framework. Once an edge is segmented from processing, it is never revisited. The only indication that it has been removed is a dashed line representing where the edge once was. While performance is improved by removing this unnecessary information, it is conceivable that these removed edges may become important again in the future. In the iCTF dataset, for example, edges were segmented out earlier in the traffic flow as they were the result of scanning and were only observed once or twice. As time goes on, these edges may become important again as teams traverse the network in search of a path to the goal. If this path or part of the path is thought to be irrelevant before more interesting traffic is observed along the edges, then the system will not capture or display this to the user. In practice, the user will have full knowledge of the network topology and important resources to manage as opposed to the potential attack types with minimal knowledge of the network. The user will be able to tell if a segmented edge leads to an important resource in practice, but there is currently no mechanism in place to reactive the edge. Incorporating this functionality will equip the user with the ability with a finer degree of control over the system while it is running.

The Python scripting language provides many benefits including weakly-typed variables which allows for rapid prototype development and dynamic run-time modification. Native support for regular expressions and easy configurations to handle file I/O make working on localized and smaller projects very convenient. Unfortunately, when developing scripts for larger projects the Python language can prove to be difficult to work with and have negative impacts with productivity. When working with a large code base, for example, small bugs injected into the software are not checked at compile-time. This can mask simple errors until hours into a test when the bad code is eventually executed. Errors in the code can lie dormant for long periods of time causing delays in production when they are eventually exposed. This may not be much of a problem in environments with rapid test and debug cycles, but it is certainly not an ideal aspect of the language [4].

Another disadvantage with the Python language is the reliance on third party libraries for more complex scientific and mathematical operations. For example, the Matplotlib library provides Python users with the ability to produce plots of their data. If a different computer needs to execute the script and has the wrong version of the library installed, the program will not run and require either a work around in the software or changes to the environment to be made. An example of this is the reliance on third party support for graphic user interface (GUI) development. To accomplish this in Python, the TkInter library is required. TkInter is a Python module that wraps Tcl and Tk and allows for tools such as widgets to be developed for a GUI. The TkInter library, however, is not thread-safe which restricts the functionality that may be desired in a GUI and potentially increases the chance for run-time errors.

The Python scripting language has distinct advantages when developing a series of classes as one can take advantage of the object-oriented nature of the language. However, a drawback to Python is that variable privacy is not a feature of the language. This increases the difficulty of handling the complexity of the program. There is no protection available to prevent a user from accessing class members and changing their values in a manner that could hinder or break the flow of the program. This also adds complications when handling larger code bases as a bug related to a data member in a class can exist in a larger number of places and be much harder to track down and fix.

Going forward, it will be beneficial to port the framework into a statically typed language such as Java. Statically-typed languages offer compile-time error checking, good performance and better overall control of the code base which allows for more hardened components to be developed [4]. Java, for example, also offers a large variety of native support for different applications. GUI development can be done using built-in, thread-safe libraries which are documented and supported by Oracle. Java is a compiled language which means once the binary has been generated, the code can run on any Java-compatible machine. This allows for greater portability of the software and robustness to changes in the run-time environment. Porting the darknet analysis framework to the Java programming language will allow for greater control, maintainability and portability in future iterations of the software.

Bibliography

[1] Jordan Bean. *CHAracterization of Relevant Attributes using Cyber Trajectory Similarities*. Rochester Institute of Technology, 2009.

[2] M. Bishop, S. Engle, D. Howard, and S. Whalen. A Taxonomy of Buffer Overflow Characteristics. *Dependable and Secure Computing, IEEE Transactions on*, 9(3):305–317, May 2012.

[3] Nicholas Childers, Bryce Boe, Lorenzo Cavallaro, Ludovico Cavedon, Marco Cova, Manuel Egele, and Giovanni Vigna. Organizing Large Scale Hacking Competitions. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 132–152. Springer, 2010.

[4] Douglas Cunningham, Eswaran Subrahmanian, and Arthur Westerberg. User-Centered Evolutionary Software Development Using Python and Java. In *Proceedings of the 6th International Python Conference, http://www. python. org/workshops/1997*, volume 10, 2010.

[5] David Dagon, Cliff Changchun Zou, and Wenke Lee. Modeling Botnet Propagation Using Time Zones. In *NDSS*, volume 6, pages 2–13, 2006.

[6] Evan Damon, Julian Dale, Evaristo Laron, Jens Mache, Nathan Land, and Richard Weiss. Hands-on Denial of Service Lab Exercises using SlowLoris and RUDY. In *Proceedings of the 2012 Information Security Curriculum Development Conference*, pages 21–29. ACM, 2012.

[7] Haitao Du, Daniel F Liu, Jared Holsopple, and Shanchieh Jay Yang. Toward Ensemble Characterization and Projection of Multistage Cyber Attacks. In *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*, pages 1–8. IEEE, 2010.

[8] Haitao Du and Shanchieh Jay Yang. Discovering Collaborative Cyber Attack Patterns using Social Network Analysis. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 129–136. Springer, 2011.

[9] Jennifer G. Dy and Carla E. Brodley. Feature Selection for Unsupervised Learning. *J. Mach. Learn. Res.*, 5:845–889, December 2004.

[10] Daniel S Fava, Stephen R Byers, and Shanchieh Jay Yang. Projecting Cyberattacks Through Variable-Length Markov Models. *Information Forensics and Security, IEEE Transactions on*, 3(3):359–369, 2008.

[11] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection. In *USENIX Security Symposium*, pages 139–154, 2008.

[12] Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. 2008.

[13] Hongling Jiang and Xiuli Shao. Detecting P2P botnets by Discovering Flow Dependency in C&C Traffic. *Peer-to-Peer Networking and Applications*, pages 1–12, 2012.

[14] S. Kumar. Smurf-based Distributed Denial of Service (DDoS) Attack Amplification in Internet. In *Internet Monitoring and Protection, 2007. ICIMP 2007. Second International Conference on*, pages 25–25, July 2007.

[15] Elizabeth A Leicht, Gavin Clarkson, Kerby Shedden, and Mark EJ Newman. Large-Scale Structure of Time Evolving Citation Networks. *The European Physical Journal B*, 59(1):75–83, 2007.

[16] Mitra Hamedanchian Mohammadi and Karim Faez. Matching Between Important Points using Dynamic Time Warping for Online Signature Verification. *J. Sel. Areas Bioinf.(JBIO) (2012)*.

[17] Todd K Moon. The Expectation-Maximization Algorithm. *Signal processing magazine, IEEE*, 13(6):47–60, 1996.

[18] Juniper Networks. SHELLCODE:X86:NOOP-STLTH-TCP. https://services.netscreen.com/restricted/sigupdates/nsm-updates/HTML/SHELLCODE:X86:NOOP-STLTH-TCP.html, 2007. [Online; accessed 15-July-2014].

[19] Reza Sadoddin and Ali Ghorbani. Alert Correlation Survey: Framework and Techniques. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*, page 37. ACM, 2006.

[20] Steven E Strapp. Segmentation and Model Generation for Large-Scale Cyber Attacks. *http://www.scholarworks.rit.edu/theses/23*, September 2013.

[21] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard A Kemmerer. Comprehensive Approach to Intrusion Detection Alert Correlation. *Dependable and Secure Computing, IEEE Transactions on*, 1(3):146–169, 2004.

[22] Giovanni Vigna. The 2010 International Capture the Flag Competition. *Security & Privacy, IEEE*, 9(1):12–14, 2011.