4-2019

Synthesizing Cyber Intrusion Alerts using Generative Adversarial Networks

Christopher R. Sweet
crs4263@rit.edu

Synthesizing Cyber Intrusion Alerts using Generative Adversarial Networks

CHRISTOPHER R. SWEET

Synthesizing Cyber Intrusion Alerts using Generative Adversarial Networks

CHRISTOPHER R. SWEET

April 2019

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

R·I·T | KATE GLEASON *College of* ENGINEERING

Department of Computer Engineering

Synthesizing Cyber Intrusion Alerts using Generative Adversarial Networks

CHRISTOPHER R. SWEET

Committee Approval:

Dr. Shanchieh Yang *Advisor* Date
Professor

Dr. Raymond Ptucha Date
Assistant Professor

Dr. Sonia Lopez-Alarcon Date
Associate Professor

Acknowledgments

Most importantly, thank you to Dr. S. Jay Yang for providing me with the opportunity to assist in research throughout my undergraduate study. Your continued feedback and encouragement to challenge myself over the years has helped me to grow into a strong engineer and critical thinker. You have had a profound impact on me, and I can never thank you enough for that. Next, I would like to thank Dr. Raymond Ptucha, Dr. Sonia Lopez-Alarcon, and the rest of the RIT Computer Engineering faculty for their willingness to educate the next generation of Computer Engineers. Finally, I would like to thank (the soon to be Dr.) Stephen Moskal who provided me my first opportunity to research with him and Dr. Yang when I was in my second year of undergrad. The advice, feedback, and good times over the last few years have been invaluable.

To my mother and father, Debbie and Jack Sweet, who taught me the value of a strong work ethic. And to my brother, John, who always set an extremely high bar for me to try and beat. Without your continued support, guidance, and devotion, I never would have been able to achieve all that I have.

Abstract

Cyber attacks infiltrating enterprise computer networks continue to grow in number, severity, and complexity as our reliance on such networks grows. Despite this, proactive cyber security remains an open challenge as cyber alert data is often not available for study. Furthermore, the data that is available is stochastically distributed, imbalanced, lacks homogeneity, and relies on complex interactions with latent aspects of the network structure. Currently, there is no commonly accepted way to model and generate synthetic alert data for further study; there are also no metrics to quantify the fidelity of synthetically generated alerts or identify critical attributes within the data.

This work proposes solutions to both the modeling of cyber alerts and how to score the fidelity of such models. Generative Adversarial Networks are employed to generate cyber alert data taken from two collegiate penetration testing competitions. A list of criteria defining desirable attributes for cyber alert data metrics is provided. Several statistical and information-theoretic metrics, such as histogram intersection and conditional entropy, meet these criteria and are used for analysis. Using these metrics, critical relationships of synthetically generated alerts may be identified and compared to data from the ground truth distribution. Finally, through these metrics, we show that adding a mutual information constraint to the model's generation increases the quality of outputs and successfully captures alerts that occur with low probability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cyber Alert data is complex due to intricate feature dependencies, lack of homogeneity and stationarity, and rarity of malignant samples. In order to further understand these characteristics of alert data it is important to understand how it is collected, each specific challenge presented by the data, and how potential methods of synthetic generation need to account for these challenges.

Network Intrusion Detection Systems

A variety of packet capture tools exist to enable individuals and corporations alike to monitor traffic on their networks. Several tools take this logging a step further and allow for real time network intrusion detection. Network Intrusion Detection Systems (NIDS) are a rule based system which allows for automated flagging of potentially malignant packet traffic through the aggregation of temporally contiguous packets; these alerts typically consist of the believed attack signature, a category that the attack falls under, target machine, timestamp, and more. These systems are employed to flag potentially malicious traffic, note long term patterns in traffic, and provide system administration with a trail of alerts to analyze after a cyber-attack has taken place.

Though these tools provide network operators with a wealth of data to analyze they suffer from a fundamental issue. The data only exists after a cyber attack has already taken place, and therefore only allows for reactionary defense techniques. This has lead

researchers to see if they can use this data, as well as simulations and extrapolation, to further understand network vulnerabilities in the pursuit of proactive cyber security.

Challenges of Cyber Alert Data

Cyber Alert data suffers from two primary challenges; There is a lack of malicious traffic data and the data that does exist is imbalanced, non-homogeneous, and unlabeled.

Access to malicious NIDS alert data has been a long standing problem for researchers in the domain of Cyber Security. As an alternative, probablistic models such as attack graphs [35, 50, 33], have been proposed and expanded to try and create realistic attack data for varying network architectures in an automated fashion. These models provide insight into potential attack paths within a network, the probability a given path will be used, and what vulnerabilities within the network allow for these paths to exist. Other works have defined methods to model these attack graphs as Markov Chains [27], employed statistical graph models [10], and used Variable Length Markov Models [13] in attempts to better understand potential vulnerabilities. Despite the effectiveness of these models they lack any means to consider historical attack data or consider real network alerts.

Of the datasets which do exist, common issues include small data set size, high imbalance between malignant and benign alerts, redundant samples, and intricate interactions leading to misleading labels. One example of this is the KDD Cup '99 [8] dataset. This dataset was prepared by Stolfo *et al.*[43] based off data captured in DARPA'98 IDS evaluation program [29]. It consists of samples from 7 weeks of network traffic collected via TCPdump that was labeled with one of five potential labels $\{$normal, denial of service, user to root attack, remote to local attack, or probing attack$\}$. Recently, it has been used for multiple studies involving cyber attack classification and prediction through the use of recurrent neural networks (RNNs) [25, 42]. However this dataset contains several pathological issues such as synthetic background traffic, underlying issues with TCPdump under intense load, and a lack of precise definitions for what constitutes an attack, as highlighted

by Tavallaee *et al.*[46] and McHugh [31].

Other publicly available datasets struggle with a low signal to noise ratio. Two examples of this are the Multi Source Cyber Event Dataset published by Los Alamos National Laboratory [24] and the DeepSecurity Dataset released by Faber and Malloy [12]. The Multi Source Cyber Event Dataset contains 4.8 KB of textual information pertaining to malicious events. The magnitude of these events pales in comparison to the overall scale of the dataset, which is encompassed in 12.2 GB of textual data. DeepSecurity faces a similar issue with a low percentage of their 600,000 network events being representative of malicious network traffic. The authors note that the availability of quality labeled data and a low signal to noise ratio for malicious activity are both outstanding issues with their studies [12].

Though the distribution of malicious events in these datasets may be representative of real world cyber alert data it creates many challenges for network defense. These challenges include the potential obfuscation of attack behavior, difficulty isolating malicious alerts interspersed throughout a stream of non-malicious alerts, and no ground truth labels. With no commonly accepted means to artificially generate additional malicious alert data, these challenges persist in the field of Cyber Security.

Generative Adversarial Networks

A Generative Adversarial Network (GAN) is a class of neural network where two neural networks are pitted against each other. One network, the generator, attempts to create samples which seem to belong to a ground truth dataset. The other network, the discriminator, takes inputs from the ground truth dataset as well as the generator and flags samples as either real or fake. This structure minimizes the generator loss each time the generator successfully creates a sample that tricks the discriminator into marking the sample as real. Conversely, the discriminator loss is minimized when all samples from the ground truth set are marked as real and all samples created by the generator are marked as fake.

GANs have achieved state of the art results in generating data with respect to images [23, 52, 26], text [44], and sound [9, 16]. Additionally, they have also been shown to perform well at more complex tasks such as scene to scene translation in images [52, 7] and stylized image generation [23]. These architectures allow GANs to serve as a powerful tool to artificially expand datasets. Additionally, high fidelity data generation requires the generator to learn key dependencies between features within each generated sample. A means to reveal and analyze these dependencies would be a powerful tool for analyzing critical features within the dataset.

Despite these successes, GANs do have several shortcomings. They are noted for requiring large numbers of samples per class and are typically trained across very large datasets for many epochs. The loss functions do not represent the quality of the data, as both the generator and discriminator are continually learning; as one network becomes better it's loss may drop, only to rise back up in a few batches when the other network learns something new as well. This lack of convergence makes training and hyperparameter tuning even more important than in traditional Deep Learning models. Allowing one model to overpower the other starves the system from having any useful gradient feedback. Finally, output mode dropping may occur, as the generator may not receive sufficient gradient feedback to encourage full exploration of the dataset.

Problem Statement

In order to address the lack of malicious NIDS alert data for cyber attack studies we explore artificial cyber alert synthesis. Due to the intricate feature relationships, data imbalance, and stochastic nature of alerts we employ the use of GANs. The application of GANs is challenged by potential for output mode collapse and failure of the network to learn realistic output distributions for each feature. Generalized preprocessing techniques are defined to prepare NIDS alert data for usage with GANs while also making model inputs and outputs more intuitive for analysis.

Since there is no commonly accepted metric to score the fidelity of synthetically generated alerts, several desirable attributes for alert fidelity scoring are defined. Subsequently, Histogram Intersection and Jensen Shannon divergence are proposed as metrics which meet these criteria and are used to evaluate synthetically generated alerts from a GAN. Advantages and disadvantages of each metric are reviewed.

Furthermore, conditional entropy and joint entropy are suggested as means to measure the efficacy of the GAN on learning the intricate feature interactions within an alert. Conditional probability tables are also employed to further understand the degree to which GANs learn feature dependencies. These methods provide insight into the critical features of alert data even when applied outside the context of analyzing synthetically generated alerts. Additionally, they provide detailed contextual information by allowing researchers to directly analyze feature-value relationships in alert data.

Finally, mutual information maximization is proposed as a means to regularize the generators output. This addition encourages further exploration of the ground truth dataset and reduces output mode collapse.

Chapter 2

Related Work

With the recent resurgence of Machine Learning research and incredible effectiveness of Deep Learning models, researchers across a variety of domains have begun to apply these methods to outstanding challenges in their fields. Cyber-Security is included in these new-found avenues of research, as Deep Neural Networks have been used for cyber event classification, forecasting, malicious traffic modification, and data generation. This section shall be organized as follows: First, a survey of existing applications of Machine Learning to Cyber Security problems will be reviewed to illustrate the need for rich alert datasets. Then advances in generative models from other fields will be examined to better understand how state of the art generative models achieve such impressive results. And finally, the current use case of generative models for cyber security will be compared to the methods proposed by this work.

Cyber Security and Machine Learning

The predominant application of Machine Learning to Cyber Security has been focused on the challenge of anomaly detection, attack classification, and event prediction. For all of these tasks intricate relationships must be considered between the various features of alerts, as well as entire chains of temporally connected alerts.

In order to maintain these temporal relationships, Recurrent Neural Networks have been employed. Specifically, Long Short Term Memory (LSTM) units allow for the network to

learn how to weight the importance of prior events and when to forgot previous feature values entirely. These models have been applied to Cyber Physical Systems by Filonov *et al.*to identify anomalous behavior. Their results show that LSTMs are able to outperform traditional methods of anomaly detection such as PCA, FDA, DFDA, CVA and SVM [15, 14].

Other works make use of the aforementioned KDD Cup'99 dataset for cyber event classification. Both the works of Staudemeyer *et al.*[42] and Kim *et al.*[25] show that LSTM networks are able to achieve impressive results, with near 100% accuracy when tasked with identifying if a stream of traffic is normal, or falls into one of the four malignant labels.

Similarly, Tuor *et al.*[48] use system logs and a twin neural network model to identify if alerts are malignant. The first model extracts information from the past 24 hours of system logs by embedding a mixture of categorical and event count features into a hidden representation. A series of these hidden representations are then fed into the second model, an LSTM network, which classifies potential cases of malignant behavior. Through the usage of a fixed sliding window the dataset may be continuously updated. This allows for a continuous training approach to be used, keeping models up to date regarding new system log behavior.

Another example of alert classification is AI^2 [49] which integrates expert input into the network traffic used for training an LSTM. The LSTM learns identify anomalous behavior which is then classified by a human analyst. The analyst's feedback is then given to the network and used for future parameter updates so that future events may be flagged with a higher degree of accuracy.

Other works have taken the challenge of prediction and classification a step further and attempted to forecast attributes of future attacks. One example of this is the work of Perry *et al.*[34] who applied LSTMs to a collegiate penetration testing competition to see if alerts early in the competition could be used to identify which team was perpetrating an attack

later on in the event. Further, they also showed that attributes of future attacks could be forecast, such as the alert signature and destination port.

Similarly Shen *et al.*[39] applied LSTMs to a dataset of 3.4 billion security events collected from Symantec's Intrusion Prevention Product deployed on corporate networks which have opted in to it's data sharing program. Their results demonstrate that LSTM can be used to predict target machine, attack severity, and even specific common vulnerability exploits which may be used in the attack with over 80% accuracy.

All of these systems for attack classification and prediction make use of data collected via NIDS. Despite the promising results of each, several of the authors note that they believe an increased dataset would allow for them to further improve the accuracy of their models [34, 12, 39].

Generative Adversarial Networks: Models and Improvements

First proposed by Goodfellow *et al.*[17], GANs are a game theoretic model for generating increasingly realistic data in a semi supervised manner. Two neural networks, referred to as the generator and the discriminator, are pitted against each other. The generator learns a set of nonlinear transformations (T) to apply to noise $\widehat{x}$ sampled from $\mathbb{P}_{\widehat{x}}$. These transformations result in data which imitates that of the ground truth set x sampled from $\mathbb{P}_r$. The discriminator is fed both real, x sampled from $\mathbb{P}_r$, and generated, $\widetilde{x}$ sampled from $\mathbb{P}_g$, data and assigns a probability $d \in [0, 1]$ representing the believed probability that a sample came from the distribution $\mathbb{P}_r$.

Immediately, GANs proved themselves to be a powerful tool for the generation of new samples in continuous value spaces such as images. Rapid advances came through the application of conditional generation [32], Deep Convolutional Networks [36], and Information Theoretic extensions [6].

Simultaneously the training and structure of GANs were updated to improve convergence rates, palliate output mode dropping, and provide meaningful learning curves

[38, 2, 19]. Most notably, the Wasserstein GAN (WGAN) proposed by Arjovsky *et al.*[2] introduced the usage of the Earthmover Distance as the loss function for both the generator and discriminator. Intuitively, this distance represents how much "mass" must be transported from x to y in order to transform $\mathbb{P}_g$ into $\mathbb{P}_r$.

WGAN was subsequently improved by Gulrajani *et al.*[19] by adding a gradient penalty term (WGAN-GP) to regularize the gradients of D. The gradient penalty creates a 1-Lipschitz constraint on the discriminator during training by sampling noise $\widehat{x}$ from $\mathbb{P}_{\widehat{x}}$ and constraining the gradient of the L2 norm of $D(\widehat{x})$ to 1. Additionally, the discriminator was given real samples and generated samples in a 5:1 ratio per epoch of training; this is done to increase the utility of gradients provided to the generator by discriminator. These modifications to training result in the loss formulation shown in (2.1), (2.2), and (2.3) for the discriminator, the gradient penalty term, and the generator respectively:

$$D_{loss} = \mathbb{E}_{\widetilde{x}\sim\mathbb{P}_g}[D(\widetilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] + \tag{2.1}$$

$$\lambda\mathbb{E}_{\widehat{x}\sim\mathbb{P}_{\widehat{x}}}[(||\nabla_{\widehat{x}}D(\widehat{x})||_2 - 1)^2] \tag{2.2}$$

$$G_{loss} = -\mathbb{E}_{\widetilde{x}\sim\mathbb{P}_g}[D(\widetilde{x})] \tag{2.3}$$

Note that in the above λ is a tunable hyperparameter determining the influence the gradient penalty has on discriminator loss; ∇ is the symbol for the gradient of D($\widehat{x}$); $\mathbb{E}$ is the expectation of the values x sampled from the distribution $\mathbb{P}$.

$$D_{KL}(P||Q) = \sup_{T:\Omega\to\mathbf{R}} \mathbb{E}_{\mathbb{P}}[T] - \log(\mathbb{E}_{\mathbb{Q}}[e^T]) \tag{2.4}$$

Separate from the advances of WGAN and WGAN-GP, Mutual Information Neural Estimation (MINE) [3] was introduced as a means to help palliate output mode dropping and improve reconstruction of generative models. MINE uses a neural network optimized using the Donsker-Varadhan representation of the KL-Divergence, given in (2.4), to estimate the

mutual information I between two distributions.

Similar to the Earthmover Distance, the KL Divergence is a metric that quantifies the distance between two arbitrary distributions, P and Q. However, the KL divergence makes use of the logarithm to express this distance in units of nats or bits. In the DV representation given in (2.4) T is the family of transformations that maps the input features Ω to $\mathbf{R}$. Thus $\mathbb{E}_{\mathbb{P}}[T]$ and $\mathbb{E}_{\mathbb{Q}}[e^T]$ are the distributions whose divergence is being measured by a neural network learning transformations T.

In order to palliate mode dropping by the generator, Belghazi *et al.*[3] propose to regularize the generator by the negative-entropy of it's output distribution. Since this is often intractable for real world samples the mutual information between the sampled noise and output samples can be used as a proxy.

The Mutual Information estimate is then added to the generator's loss defined in (2.3), resulting in the formulation given by (2.5).

$$G_{loss} = -\mathbb{E}_{\widetilde{x}\sim\mathbb{P}_g}[D(\widetilde{x})] + D_{KL}(\widehat{x}||\mathbb{P}_g) \tag{2.5}$$

Applications of Generative Adversarial Networks to Network Traffic: Adversarial Sample Crafting

One common application of GANs has been the creation of Adversarial Samples. First noted by Szegedy *et al.*[45], Adversarial Samples are generated by taking ground truth samples, applying a small, human-imperceptible, perturbation, resulting in that sample being misclassified by a neural network with a high degree of confidence. Subsequent research found more powerful ways to generate Adversarial Samples through methods such as fast gradient sign [18], optimization based methods [5, 30, 11], and GANs [51, 37, 28, 21, 1].

Advarsarial Sample Crafting has been applied to network traffic to modify and obfus-

cate malicious traffic [37, 28, 21, 1]. These adversarial samples are created to avoid being flagged by Network Intrusion Detection Systems (NIDS).

Rigaki *et al.*[37] proposed the use of GANs in generating network traffic which mimics other types of network traffic. In particular, real malware traffic was modified by a GAN using LSTM cells to appear as legitimate network traffic. This allowed the malware to avoid detection from the Stratosphere Behavioral Intrusion Prevention System through the modification of three network traffic parameters; total byte size, duration of network flow, and time delta between current network flow and the last network flow. They showed that through the modification of these parameters detection rate could be dropped down to 0%.

Similarly, Lin *et al.*[28] apply GANs to obfuscate traffic with the intention of directly deceiving a NIDS. Their model makes use of 9 discrete features and 32 continuous features to modify attack actions to avoid detection. Available attack actions include denial of service and privilege escalation. Their model is shown to drastically increase the evasion rate of malicious network traffic across several different classifiers when benchmarked using the NSL-KDD benchmark provided in [20].

Summary

The current research in applying Machine Learning to Cyber Security is well varied, but missing the application of generative models to new sample creation and analysis. This work applies state of the art GAN models to cyber alert data collected via the Suricata NIDS to fill this gap. The alerts used for training were collected from 10 student teams during an 8 hour long collegiate penetration testing competition (CPTC) as they attacked identical, isolated, instances of the same network topography. Two different years of competition data were tested independently, each representing different network topographies and attacker behaviors; additionally, illustrating the ability of a single network topography to learn and capture many different data distributions. All data was processed to consider traffic a per target IP basis allowing for different models to be used to represent each potential target in

the network.

Unique preprocessing steps for handling cyber alert data, exhaustive model hyperparameter tuning, and methods of identifying generated sample fidelity are also provided. Namely, the usage of Histogram Intersection and Jensen Shannon Divergenc, for m-tuple feature combinations, are shown to be an effective and intuitive means of judging sample generation quality. Additionally, these metrics may be coupled with joint and conditional entropy calculations to show dependencies between features of an alert, allowing for further introspection of attack behavior.

Finally, the application of Mutual Information Estimate Maximization to WGAN-GP is shown to improve sample generation by increasing the number of output modes captured by the generator. This model is referred to as WGAN-GPMI and is shown to improve generated results through a higher intersection of histograms and closer probability distribution approximation by the generator.

Chapter 3

Methodology

In order to generate and analyze cyber alert data it is important to understand the structure of generic NIDS alerts. The categorical features contained by each alert provide contextual information such as the what, where, and how an attack occurs. By understanding the structure and meaning of each alert feature it is possible to begin selecting which features are worth pursuing synthetic generation of. Furthermore, the selection of these features drives the structure of the generative model used. This section will detail three neural network architectures driven by the features selected from NIDS alerts.

Alert Structure and Feature Selection

NIDS alerts are formed through the compilation of packet traffic seen by each machine on the network. These alerts use fixed rulesets and information collected from multiple packets to summarize the believed intent of network traffic. Features such as HTTP Request Information, Alert Category and Signature, Destination IP, Source IP, and more are all captured by alerts. A subset of alert features are shown in Fig. 3.1.

It is important to note that not all features are populated in every alert and that most features are comprised of discrete categorical values. Additionally, many features are grouped into families of information. One example of this is the HTTP family, which includes Hostname, Method, Redirect, URL, and more.

Given the wide variety of alert features available it is important to establish which alerts

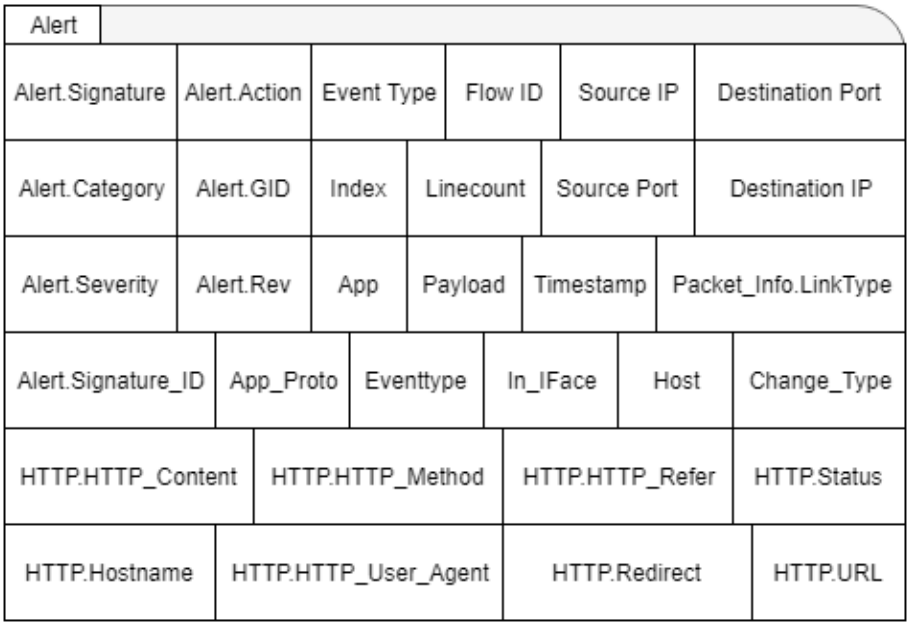


Figure 3.1: Sample NIDS Alert

provide useful contextual information for future network alert modeling. Additionally, some of the features listed are generated through a fixed mapping from other features. For example, Alert Category is determined by a many to one mapping of Alert Signatures. Therefore, by generating Alert Signature it is possible to determine the Alert Category.

When selecting features of an alert to use for any model it is important to consider the contextual data provided by that features. Features such as Alert Signature, Category, Action, and Eventtype provide the what and how an attack is carried out over the network. Destination IP and Port and Source IP and Port provide where the attack originated from and where it's targeting. And Timestamp provides not only when an attack occurs but also it's temporal location with respect to other alerts.

GAN Models

Two GAN implementations were created to generate artificial cyber alert data; a standard Wasserstein GAN with Gradient Penalty and an extension of this model which used a neural estimate of mutual information to regularize the generator output. The mutual information

estimate model architecture is also reviewed in detail.

Wasserstein GAN with Gradient Penalty

The Wasserstein GAN with Gradient Penalty makes no architectural changes to model structure compared to standard GANs. Rather, the loss function is modified to use the Earthmover Distance. This modification has been shown to create empirically better results and allows for flexibility in model selection adapted to the challenge at hand.

Since the goal of this work was to create individual NIDS alerts without temporal correlation a feed-forward network architecture was selected for both the generator and discriminator. Four alert features were selected for alert generation: Alert Signature, Destination Port, Timestamp, and Source IP. Despite this, the models employed could easily be scaled to consider n-many features.

The generator consisted of 2 layers. The first layer sampled from noise space $\mathbb{Z}$ to a hidden representation. The next layer was directly responsible for each feature's output value. Since each of the four selected features was categorical, one hot encoded representations were used to represent each unique value. This layer consisted of four individual mappings from the hidden representation to an output layer with cardinality equal to the number of unique values per feature. Finally, a concatenation was used to take the prior 4 outputs and create a full 1-hot encoded alert.

The discriminator also consisted of 2 layers. The first layer took 1-hot encoded alerts as input and mapped them to a hidden representation. The next layer mapped this hidden representation to a scalar value representing the probability that the alert was from the ground truth dataset. A graphical model of this architecture is included in Fig. 3.2. Inputs to the network are highlighted in yellow. Learnable weight layers are in blue. The concatenation in orange is a non-backpropable layer used only to prepare generator output for input to the discriminator. And the red boxes and lines represent the model loss functions and back-propagation.

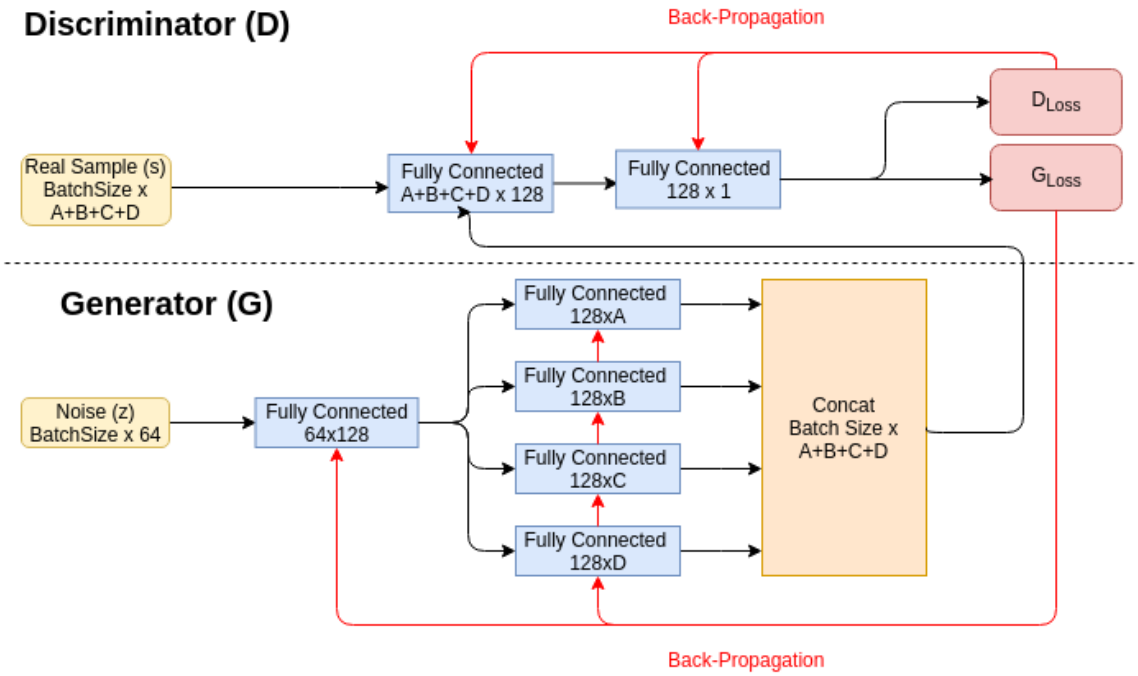


Figure 3.2: WGAN-GP Model

Table 3.1 and Table 3.2, shows the dimensions of the weight matrices and activation functions for each layer of the network.

Table 3.1: Generator Network Architecture: Note that a-d are variable depending on the number of unique outputs

Generator		
Layer	Matrix Dimensions	Activation Function
Input	64 x 128	x^+
Output - Feature 0	128 x A	NA
Output - Feature 1	128 x B	NA
Output - Feature 2	128 x C	NA
Output - Feature 3	128 x D	NA
Concatenation	A+B+C+D	NA

Table 3.2: Discriminator Network Architecture: Note that a-d are variable depending on the number of unique outputs

Discriminator		
Layer	Matrix Dimensions	Activation Function
Input	A+B+C+D x 128	x^+
Output	128 x 1	$\frac{1}{1+e^{-x}}$

Mutual Information Neural Estimator

Mutual Information is a measure of dependence between two random variables. Traditionally, approximations have to be used to estimate the mutual information between high dimensional and continuous variables as exact computation is intractable. The Mutual Information Neural Estimator is a neural network which allows for state of the art approximation of mutual information. This is done by optimizing the network to minimize the Donsker-Varadhan representation of the KL divergence between two variables.

A feed forward neural network was implemented to learn the mutual information between the gaussian noise sampled from $\mathbb{Z}$ and the generators output. This network consisted of 2 layers. The first layer took input from each of the aforementioned sources and mapped them to separate hidden representation layers and added together. Then the second layer mapped the hidden representation to a single output value representing the mutual information estimate. Table 3.3 shows the matrix dimension for each of the layers in the network.

Table 3.3: Mutual Information Estimator Network Architecture: Note that a-d are variable depending on the number of unique outputs

Estimator		
Layer	Matrix Dimensions	Activation Function
Input - Generated	A+B+C+D x 128	NA
Input - Noise	64 x 128	NA
Addition	128	NA
Output	128 x 1	NA

WGAN-GP with Mutual Information Constraint

In order to improve mode dropping in the GAN model described in Section 3.2.1 the Mutual Information Neural Estimator in Section 3.2.2 is added to the WGAN-GP model. This is done by using the mutual information between the generated samples and the input noise as a proxy for the neg-entropy of the samples. This regularizes the generator's weights and

encourages full exploration of the feature space. We refer to this model as the Wasserstein GAN with Gradient Penalty and Mutual Information (WGAN-GPMI). Fig. 3.3 shows what the full model consists of. The addition of the Mutual Information Estimation Network helps to enforce that the generator must learn all output modes of the distribution by fully exploiting the noise sample.

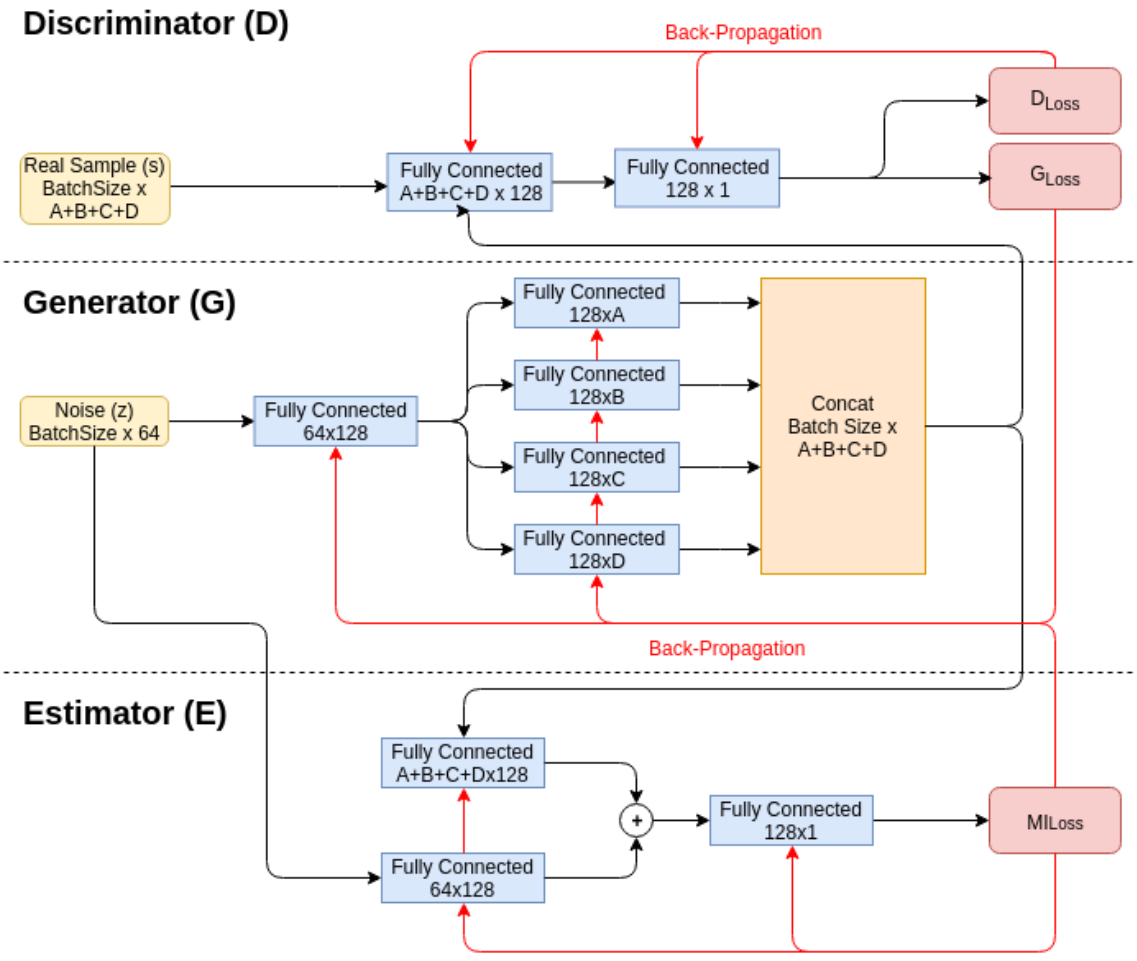


Figure 3.3: WGAN-GPMI Model

Since mutual information is theoretically unbounded, gradient updates resulting from it could overwhelm the adversarial gradients resulting from the Earthmover Distance. In order to address this all of the gradient updates to the generator are adaptively clipped to ensure that the Frobenius norm of the gradient resulting from the mutual information is at

most equal to the adversarial gradient [3], as shown in (3.1).

$$g_{norm} = g_a + min(||g_a||, ||g_m||)(\frac{g_m}{||g_m||}) \tag{3.1}$$

Note that g_{norm} is the normalized gradient, g_a is the adversarial gradient resulting from (2.3), and g_m is the gradient resulting from (2.4).

Chapter 4

Analysis Methods

The usage of NIDS alert data with Machine Learning algorithms suffers from two issues; significant preprocessing must be performed to make the data have contextual meaning and methods to analyze the similarity of alerts are not well defined. For example, predicting a specific port which will be attacked only provides a small piece of significant data. However, a more significant feature to generate would be what type of service will be attacked. Additionally, when analyzing cyber alerts identifying the fidelity of data is not straight forward. In tasks such as event prediction results may be quantified by cross entropy loss on a per feature basis. But when generating new alerts there are complex interactions between various features which must be accounted for by the model. Simply generating *a* realistic signature and port category individually is much less important than generating a realistic *combination* of port signature and category.

To address these challenges this section will cover the unique preprocessing applied to NIDS data collected from Suricata. Specifically, the features of Alert Signature, Destination Port, Timestamp, and Source IP will be considered. Additionally, intuitive metrics for analyzing alert fidelity are introduced as an inclusive system to address how realisitc generated alerts are compared to their source alerts from the ground truth dataset. Finally, metrics to analyze dependency are introduced and used to verify that high level relationships between alert features are preserved by the model.

Cyber Alert Data Preprocessing

The data used for these experiments comes from the National Collegiate Penetration Testing Competition from 2017 and 2018 (`https://nationalcptc.org/`). CPTC'17 features data that was collected from student teams penetrating into a mock election system. This network topology featured several server systems hosted across a variety of subnets. Students were tasked with modifying votes or exfiltrating voter data from the network. CPTC'18 data featured an entirely new network topology as students penetrated into a network of autonomous cars featuring embedded systems, mobile phones, and host systems for data processing. Each team had around 8 hours to scan, infiltrate the network, and exfiltrate information from the target. Both datasets provide a unique opportunity for Machine Learning experimentation as they are completely comprised of malicious actions as teams attempt to penetrate the target network. Though this data is unique to the competition it is worth noting that the preprocessing described herein is applicable to any dataset consisting of NIDS alerts.

The first preprocessing step applied to the data was to separate alerts on a per Destination IP basis. This allowed individual models to be trained for each system on the network, typifying the type of traffic seen at that target. Additionally, data from all of the teams could be compounded, allowing for the number of potential attacks taken on a single target to be more fully expressed during training. Segmentation on a per-target basis has several intuitive benefits: First, it allows for different vulnerabilities to be highlighted on each machine given commonly occurring alert features at that target. Secondly, it helps to remove noisy alert influence from critical nodes in the network. For example, internet facing IPs may contain a significant amount of scanning activity, drowning out exfiltration related alert features at nodes further embedded in the network. Finally, the information extracted from alerts on a per target basis is actionable, as network administrators can use commonly targeted vulnerabilities to tune network settings for future defense.

Next, the dimensionality of the destination port feature was reduced based off common service categories run across a collection of ports provided by the Internet Assigned Numbers Authority [47]. This reduction drops the number of unique values from 1516 destination ports to 69 destination port categories for the CPTC'17 dataset. Additionally, the dimensionality reduction step can easily be expanded or customized on a per network basis given a corporation's configuration of services. Contextually, this has the effect of indicating what service is being targeted by attackers, rather than just knowing a specific port number. Herein the processed Destination Ports are referred to as Destination Services.

Finally, a set of simple statistical criterion were used to segment timestamps into bins. Traditional modeling of cyber attacks use killchain stages to segment actions into a series of contiguous stages with dependencies on previous stages. The beginning of an attack may consist of reconnaissance based actions, yielding information about which IP to target in later attack stages. Similarly, the CPTC dataset may be segmented to try and capture unique behaviors into different Time Bins.

Following the methodology shown by [34] bins are generated by smoothing the histogram timestamps and taking the first derivative to identify local minima and maxima. Then stages are cut if they contain at least 10% of the total data and consecutive events at the candidate cut-point contain less than 0.5% of the total data. The goal of this ruleset is to capture significantly different types of traffic that does not split bursts of data into multiple stages.

Finally, if applied to real world NIDS data, another potential preprocessing step would be to segment source IPs based off of subnet. However, given that CPTC occurs in a virtualized network for a collegiate competition this preprocessing step is ignored. Additionally, it is worth noting that in real world data, the Source IP could easily be obfuscated through the use of a proxy.

Methods of Analyzing Alert Data

Analyzing the degree of realism for artificially generated alert data is non-trivial. While other fields such as Computer Vision have created well defined metrics such as Inception Score [38] or allow for direct human analysis of image quality, no analogue exists for NIDS alerts. Several works have proposed the use of graph based metrics such as comparing nodes and their connectivity for both generated and real network traffic [40, 22], or looking at low level parameters such as distributions of packets [41, 4]. However, despite these works, there is no widely accepted methodology.

It is important to consider the desirable attributes of a *good* metric. A good metric must provide an intuitive, scalable way to summarize what would otherwise be an intractable amount of data to comprehend. Other desirable properties include ways to directly visualize the results of the metric so that trends may be identified visually, able to capture high level dependencies, and tolerance to samples with the value 0.

To this end, we propose the usage of several metrics for analyzing NIDS alert data. First, Histogram Intersection is considered; this metric compares the similarity of two histograms within the same domain by computing the amount of overlap between them. Histogram Intersection meets several of the above criterion, as it is naturally bounded between 0 and 1, easily visualized by directly plotting the histograms being compared, and can be extended to accommodate m-many tuples of features. The m-many tuples can be thought of as a joint histogram, where m is the number of unique features considered in the joint distribution. This can be done automatically by iterating over all M choose m *combinations*, where M is the total number of unique features in the dataset. Mathematically, the Histogram Intersection is defined in (4.1), where *P* represents the ground truth data histogram and *Q* represents the generated data histogram, each of which has N samples.

$$G(P,Q) = \frac{\sum_{i=0}^{N} min(P_i, Q_i)}{max(\sum_{i=0}^{N} P_i, \sum_{i=0}^{N} Q_i)} \tag{4.1}$$

Another powerful trait of the Histogram Intersection is that it can be used to reveal dependencies between features within a single alert. This is accomplished by looking at the difference in Histogram Intersection scores between $m \pm 1$ tuples and observing the intersection drop. An intuitive example of this can be thought of as follows; If the intersection for feature A is 0.9 and the intersection for a 2-tuple histogram consisting of features A and B is 0.875 then it is expected that a dependency exists between A and B. It is important to note that this dependency is not inherently bidirectional, as A and B may have varying intersections to begin with. Fig. 4.1 illustrates a graph based schema to identify these dependencies visually.

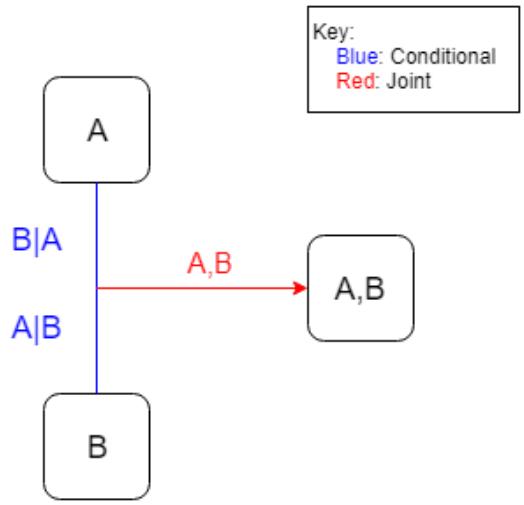


Figure 4.1: Example Feature Graph Highlighting Conditional and Joint Entropy

In order to confirm these dependencies we introduce our second metric; conditional entropy. The conditional entropy of each unique *permutation* may be calculated for each m-tuple of features. Continuing with the previous example this means the conditional entropy of both $A|B$ and $B|A$ are computed. In order to compute a single value that represents the average conditional entropy for all input condition values, the entropy term is computed using (4.2). This calculation weights the entropy of each possible input combination based off the probability that input i occurs as $\frac{|w_i|}{|w|}$. $p_{i|j}$ represents the probability of the output feature value at index i occurring given the input feature values at index j and must be

computed for all unique input value combinations Z.

$$\widehat{H}_{Y|X_0,X_1,...,X_m} = \sum_{i=0}^{N} \left(\frac{|w_i|}{|w|} * \sum_{j=0}^{Z} \left(p_{i|j} * \log(\frac{1}{p_{i|j}}) \right) \right) \tag{4.2}$$

In order to further strengthen the argument that there are dependencies between features it is important to consider the overall randomness of the m-tuple joint distribution. To do this (4.3) may be used to compute the joint entropy of the distribution. Using the aforementioned example with features A and B, the joint of these variables is denoted A, B.

$$H_{X_m} = -\sum_{x_m} p(x_0, x_1, ..., x_m) * \log\left(p(x_0, x_1, ..., x_m)\right) \tag{4.3}$$

Given that natural logarithms are used for the calculation in (4.2) and (4.3), the resulting value is given in the natural unit of information (nats), a log base e equivalent to bits. Given that the distribution of m-tuple feature histograms is a discrete distribution with finite support the upper bound of entropy is given by the uniform distribution $\mathbb{U}$. Note that the cardinality of $\mathbb{U}$ varies to match the number of unique values in the conditional or joint probability being normalized. Using this quantity, (4.2) and (4.3) can be normalized as shown in (4.4) and (4.5). This has the benefit of naturally bounding the entropies between 0 and 1, similar to the intersection score defined in (4.1).

$$\overline{H}_{Y|X_0,X_1,...,X_m} = \frac{\widehat{H}_{Y|X_0,X_1,...,X_m}}{H(\mathbb{U})} \tag{4.4}$$

$$\overline{H}_{X_m} = \frac{H_{X_m}}{H(\mathbb{U})} \tag{4.5}$$

If the drop in intersections is indeed correlated to feature dependency, then the conditional entropy should be directly proportional to this drop. Additionally, the benefit of capturing feature dependencies is apparent by comparing to the joint entropy without any

conditioning information. Note that in Fig. 4.1 the edges corresponding to conditional and joint entropies are labeled using the notation given in the examples above.

One drawback of using the intersection of histograms is that the metric does not perform well when the ground truth distribution has one value which occurs with high probability. The data generating model can learn to output that value in a purely deterministic manner and receive an intersection score equivalent to the probability of that given value occurring in the ground truth set. This issue is known as output mode collapse and has been historically problematic for GAN based models. In order to identify this issue a metric which penalizes failure to accurately represent the probability distribution of the ground truth set is required.

Kullback Leibler Divergence, given in (4.6), was considered as a candidate however it does not have the property of zero tolerance and is asymmetric. This would require special handling of null outputs and would require a defined convention of which probability distribution is considered P and which is given by Q.

$$D_{KL}(P||Q) = -\sum_{x \in X} P(x) \log \frac{Q(x)}{P(x)} \tag{4.6}$$

The Jensen Shannon Divergence, given in (4.8), was then considered. It is both zero tolerant and symmetric while maintaining the penalty for failing to accurately represent the ground truth probability distribution. This metric has the downside of having no upper bound and by extension is not as intuitive as the Histogram Intersection. However it can be used as a drop in replacement for Histogram Intersection and can still be used in conjunction with the weighted conditional and joint entropies to identify feature dependency.

$$M(P, Q) = \frac{1}{2}(P + Q) \tag{4.7}$$

$$D_{JS}(P||Q) = \frac{1}{2}(D_{KL}(P||M) + D_{KL}(Q||M)) \tag{4.8}$$

Other Useful Ways to Analyze Alerts

The purpose of this section is to review other methods of analyzing artificially generated alerts. The methods presented in this section are comprised of subsidiary steps for the computation of the previously presented metrics and or provide useful information in understanding what is driving model behavior.

Using Conditional Probability Tables to Evaluate Generated Alerts

In the aforementioned (4.2) weighted conditional entropy is computed to provide a singular score representing conditional randomness in m-tuple histograms. An intermediary step in this computation involves the creation of conditional probability tables which show all possible input conditioning values and their impact on output value probability. It was found through experimentation that directly outputting these tables and applying highlighting to reveal sparsity and determinism is an effective means to evaluate specific feature value relationships. This method has the drawback of becoming intractable as the number of tables generated is equal to the number of unique feature permutations. A sample table is given in Section 5 to illustrate specific feature-value relationships.

Measuring Output Mode Capture

One important attribute of artificially generated data is the number of output modes that the model manages to capture. Traditionally, GANs have suffered from output mode collapse, where outputs that have a low probability of occurring in the ground truth do not ever occur in the model's output. This can be thought of as a false negative for the model. Additionally, if the GAN has not been trained sufficiently then there is the potential for it to generate noisy samples which never occurred in the ground truth dataset. This can be thought of as a false positive for the model.

Generating a table of the false negatives allows for direct testing of the WGAN-GPMI

model, which decreases output mode collapse through a mutual information constraint on the generator. Additionally, a table of false negatives allows for direct testing of the amount of noise generated by the model, potentially indicating that more training steps or data is required to obtain realistic results. Ideally, both of these values should be driven towards zero if the model is performing well. Note that this does not indicate a perfect model however, as the probability distribution for output modes may not reflect those of the ground truth distribution.

Chapter 5

Results and Analysis

Each of the GAN models described in Section 3.2 were used to create artificial NIDS alert data. Using the methods described in Section 4.2 the fidelity of the learned model was analyzed. This analysis can be broken down into the following sections:

1. Thorough Hyperparameter Search - Individual hyperparameters were tuned for each model to see their impact on Histogram Intersection. Top candidate values were selected for a full hyperparameter search where all combinations of hyperparameter values were tested. The results are presented for both WGAN-GP and the improved WGAN-GPMI.

2. Alert Fidelity - A subset of target IPs from the CPTC'17 dataset were used for training WGAN-GP and WGAN-GPMI models. The results of these models were analyzed and visualized using histogram intersection and Jensen-Shannon Divergence.

3. Alert Dependency - For the same subset of target IPs alert dependencies were identified by using drop in Histogram Intersection, entropy computation, and conditional probability tables.

4. Output Modes Captured - The number of output modes captured by the model is comprised of two components. How many of the true output modes are output by the model? And how many output modes by the model do not occur in the ground truth?

5. Generality of Models - To show the ability of these models to perform on a variety of datasets we introduce the CPTC'18 dataset. Histogram Intersection and Jensen Shannon Divergence are used to measure the performance of each model when trained on a new dataset.

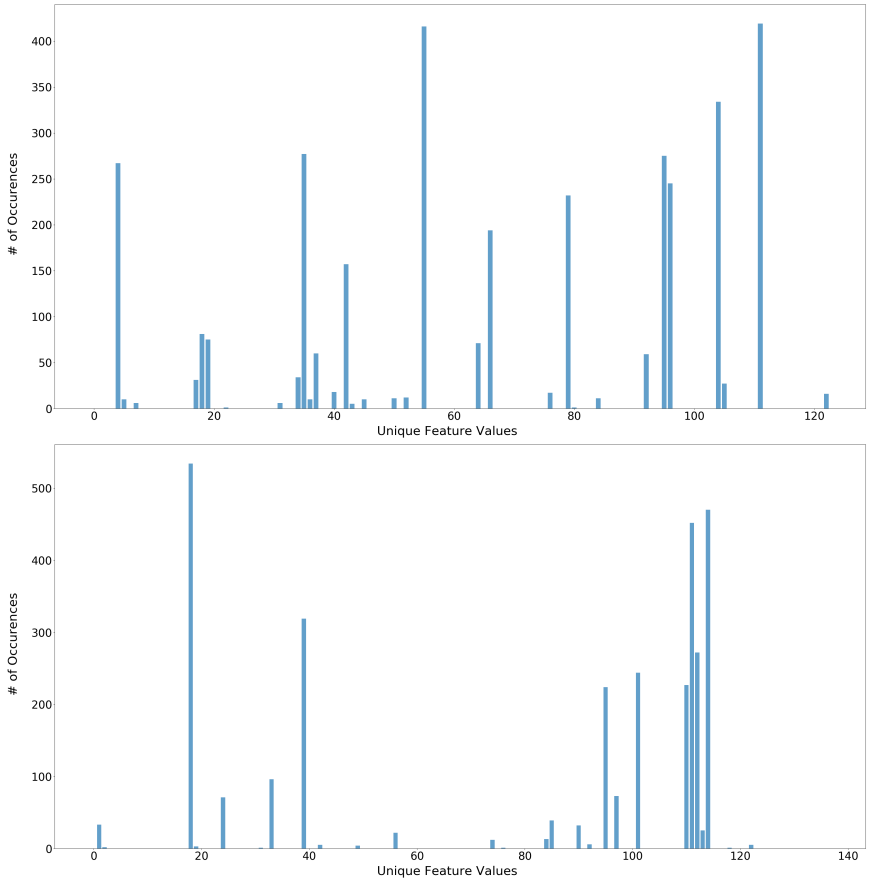
The CPTC'17 dataset was segmented on per-target IP basis and used across all experiments. For the hyperparameter search the IP which contained the most alerts, 10.0.0.100, was used. For the following experiments on alert fidelity, modeling dependency, and output modes captured the following four IP's were used: 10.0.0.100, 10.0.0.22, 10.0.0.27, 10.0.99.143. These four IP addresses provided a mixture of Windows and Linux Machines, with varying purpose, and contained the 4 greatest counts of alerts. Table. 5.1 summarizes the differences between each of these machines.

Table 5.1: Mapping of Target IP Address to Machine Usage/Purpose

IP Address	Operating System	Machine Usage	Number of Alerts
10.0.0.100	Windows	Active Directory Server	3388
10.0.0.27	Ubuntu	HTTP Server	3166
10.0.0.22	Ubuntu	MySQL Server	2974
10.0.99.143	Ubuntu	HTTP Server	2182

These targets were also selected because of the characteristics of their alert distributions. For each target the distribution of alerts cannot be modeled using simple traditional distributions and vary with respect to other target IP addresses. For example, when examining the distribution of 4-Tuple Feature combinations for Target IP 10.0.0.27 the combination of features which occurs at ID number 19 dominates the probability distribution. When compared to Target IP 10.0.0.100 the probability of the same combination occurring is much lower. Conversely, one of the dominating combinations for Target 10.0.0.100, ID number 57, doesn't ever occur in the distribution of 10.0.0.27. These are shown visually in Fig. 5.1.

Though there is no direct weighting applied to the results of each feature combination the importance of each feature generated does play a role in how realistic the alert seemed.



(a) Target IP 10.0.0.100

(b) Target IP 10.0.0.27

Figure 5.1: Differences in Distribution of 4 Alert Feature Combination Between Varying Target IPs

For example, generating realistic alert signatures provides directly actionable information regarding what types of attacks or vulnerabilities are targeted on the given source. However, this information becomes much more useful when paired with a realistic destination service. Later on, the results will show that the models presented by this work are capable of learning these useful feature value combinations.

Thorough Hyperparameter Search

A two part hyperparameter search was employed to find optimal values for generating alerts from the CTPC datasets. First, individual parameters were tested in order to find several values which showed promising results when applying the Histogram Intersection metric. For each parameter value tested, the Histogram Intersection was computed for all possible feature-value combinations. These values were then plotted against all other parameter setting results for WGAN-GP and WGAN-GPMI. To save space in the plots each feature combination was assigned a numeric key. Table 5.2 provides the key mapping to each potential feature combination.

Table 5.2: Key for Feature Combinations

Feature Combination	ID	M-Tuple
Alert Signature	A	1
Time Bin	T	
Destination Service	D	
Source IP	S	
Alert Signature, Destination Service	A,D	2
Source IP, Time Bin	S,T	
Alert Signature, Source IP	A,S	
Alert Signature, Time Bin	A,T	
Destination Service, Time Bin	D,T	
Source IP, Destination Service	S,D	
Alert Signature, Source IP, Time Bin	A,S,T	3
Source IP, Destination Service, Time Bin	S,D,T	
Alert Signature, Source IP, Destination Service	A,S,D	
Alert Signature, Destination Service, Time Bin	A,D,T	
Alert Signature, Source IP, Destination Service, Time Bin	A,S,D,T	4

Then, these values were taken and used for a full parameter sweep, which tested every possible combination of the parameter values available. Several candidate values were selected for each of the hyperparameters due to the unknown nature of hyperparameter interaction.

This two stage search was carried out twice, once for each of the GAN models presented in Section 3.2. Both models only used data from the target IP with the most number of alerts, 10.0.0.100, to try and avoid information constraints related to small dataset size. The parameters tested included lambda, batch size, learning rate, hidden dimension, and number of epochs.

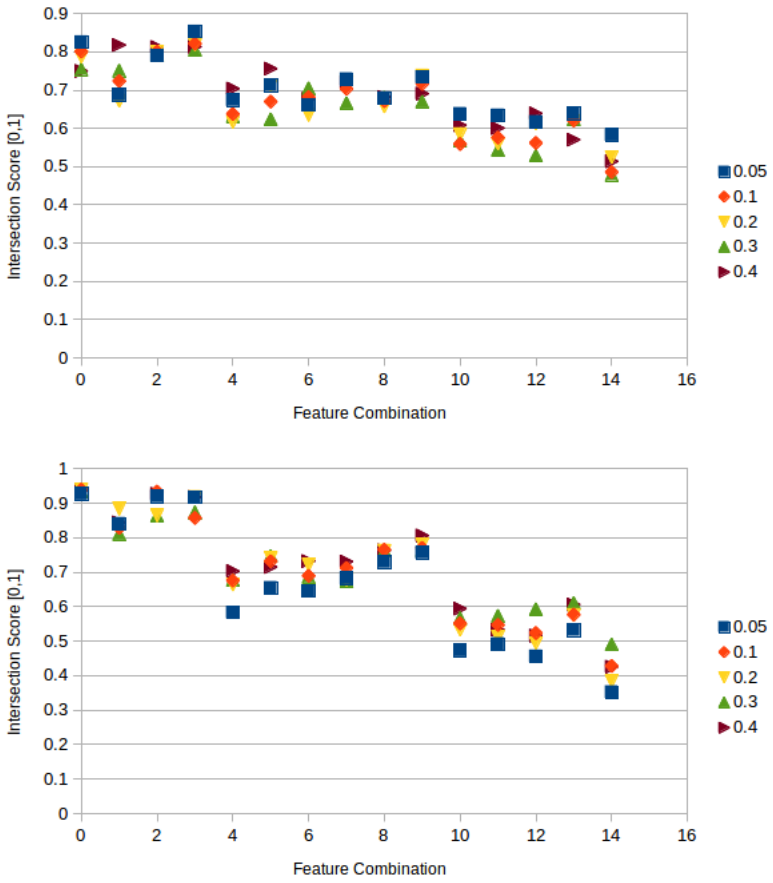
Lambda

The lambda parameter was used as a coefficient to the gradient penalty term applied to the discriminator. The values tested for lambda were $\{0.05, 0.1, 0.2, 0.3, 0.4\}$. The intersection vs. parameter setting plots for WGAN-GP and WGAN-GPMI may be seen in Fig. 5.2a and Fig. 5.2b respectively.

The performance of all the values tested was very close. For the WGAN-GP model smaller values such as $\{0.05, 0.1, 0.2\}$ performed best. In the WGAN-GPMI model the larger values tested, $\{0.2, 0.3, 0.4\}$ performed best. This suggests that the WGAN-GPMI model requires stronger enforcement of the gradient penalty than the WGAN-GP model does. Given that the gradient of the generator is changed from an outside source (the mutual information estimate) in addition to the discriminator feedback, the discriminator may be trying to make larger gradient changes to identify generated samples.

Batch Size

The batch size determines how many alert samples were fed into the model in parallel. Higher batch sizes are more computationally intensive, but provide a better representation of the ground truth data distribution. Additionally, larger batch sizes reduce the number of



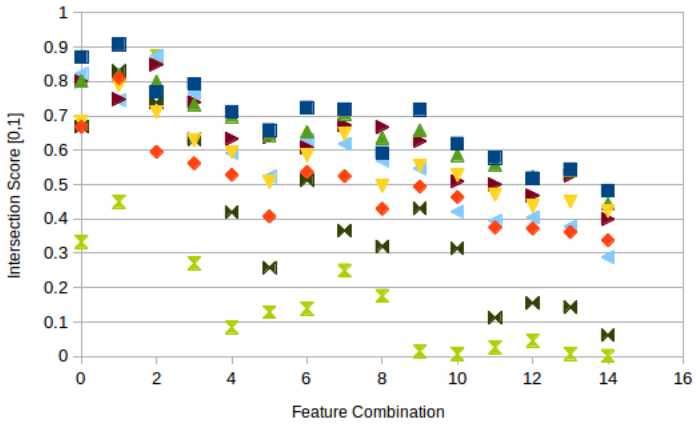
(a) All other hyperparameters were held constant at the following values: $epochs = 180$, $batch\_size = 100$, $learning\_rate = 5e-5$, $hidden\_dimension = 128$

(b) All other hyperparameters were held constant at the following values: $epochs = 250$, $batch\_size = 100$, $learning\_rate = 5e-4$, $hidden\_dimension = 128$

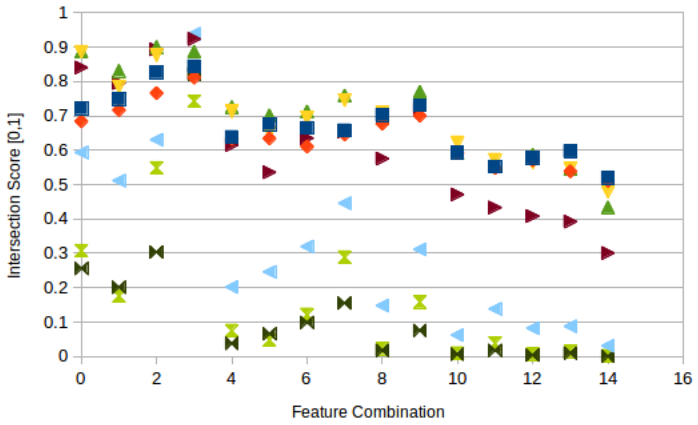
Figure 5.2: Lambda Parameter Search

steps required to complete a full epoch of training.

The values tested for batch size were $\{10, 25, 50, 100, 150, 250, 500, 1000\}$. The intersection vs. parameter setting plots for WGAN-GP and WGAN-GPMI may be seen in Fig. 5.3a and Fig. 5.3b respectively.



(a) All other hyperparameters were held constant at the following values: $epochs = 180$, $learning\_rate = 5e{-}5$, $hidden\_dimension = 128$, $\lambda = 0.1$



(b) All other hyperparameters were held constant at the following values: $epochs = 250$, $learning\_rate = 5e{-}4$, $hidden\_dimension = 128$, $\lambda = 0.3$

Figure 5.3: Batch Size Parameter Search

The overall range and trends of intersection scores are very similar for both WGAN-GP and WGAN-GPMI. It is interesting to observe that for the WGAN-GPMI model that the performance drop after using batch sizes greater than 250 is much greater than that of the WGAN-GP model. This is likely due to additional network generating the mutual information estimate. This network is trained in parallel with the WGAN model, thus earlier steps in training may have poor mutual information estimates. Since larger batch size drives less gradient updates over the same number of epochs there is a larger number

of inaccurate mutual information estimates provided to the WGAN portion of the model. The top selections for the full parameter search are $\{10, 25, 50, 100, 150\}$ for the WGAN model and $\{50, 100\}$ for the WGAN-GPMI model.

Learning Rate

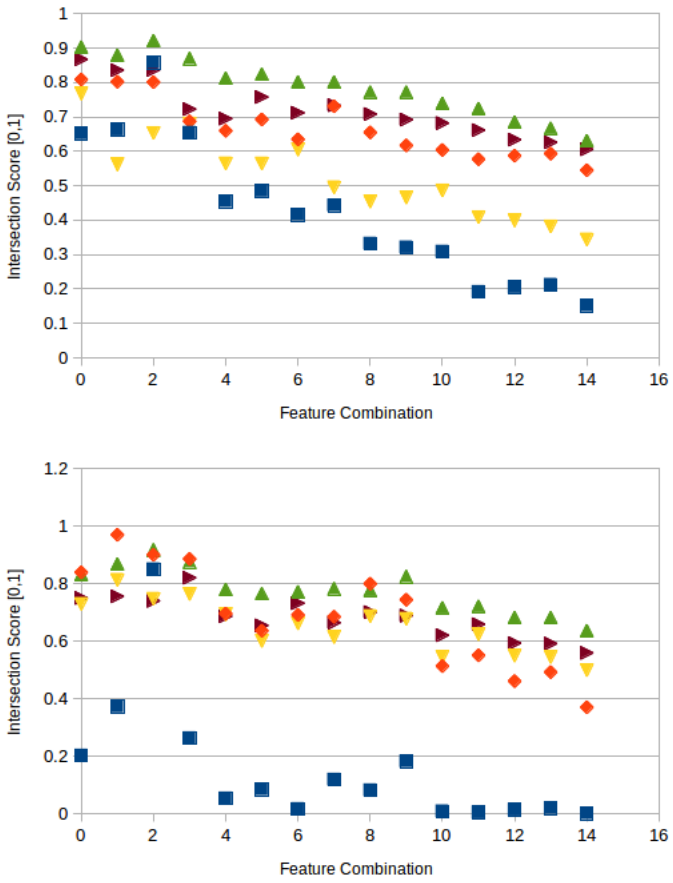
The learning rate of the optimizer defines the base step size, weighted by the gradient of the loss, that is taken when adjusting parameter weights during training. A large learning rate converges quickly, but may overshoot the global optimum and never reach peak performance. A small learning rate won't overshoot the global optimum, however will take significantly longer to converge. Due to the categorical output of alert data and existing difficulty in optimizing GANs, small learning rates were tested. This allowed the network to be able to make fine tuned changes to network weights, as slight changes in output probability create entirely different output values. Additionally, the ADAM optimizer was used, allowing for weight decay over time to modify the learning rate parameter.

The values tested for learning rate were $\{1e-5, 5e-5, 1e-4, 5e-4, 1e-3\}$. The intersection vs. parameter setting plots for WGAN-GP and WGAN-GPMI may be seen in Fig. 5.4a and Fig. 5.4b respectively.

For both of the models tested, the smallest learning rate $1e-5$ performs poorly. Interestingly, in the WGAN-GP model the subsequent three learning rates $\{5e{-}5, 1e{-}4, 5e{-}4\}$ oscillate between performing well and poorly. due to this oscillation, $1e-4$ is dropped, leaving $\{5e-5, 5e-4, 1e-3\}$ for the full parameter search. For the WGAN-GPMI model $\{1e-4, 5e-4, 1e-3\}$ are all used in the full parameter test.

Hidden Dimension

The hidden dimension size determines the number of hidden units available in each hidden layer. Higher hidden dimensions provide more learnable connections to the network allowing the network to learn complex approximations. On the other hand, larger hidden



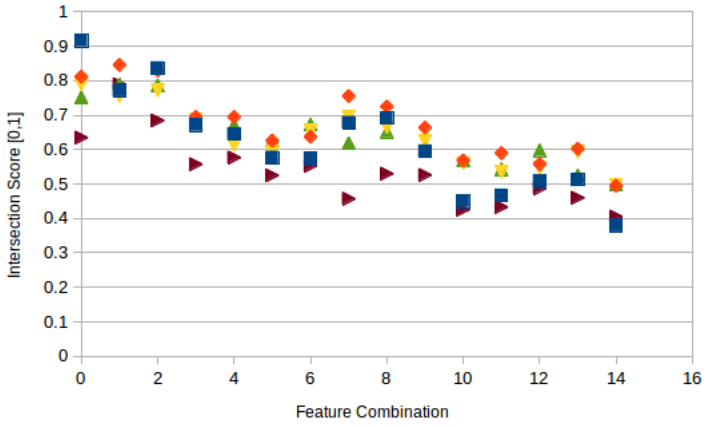
(a) All other hyperparameters were held constant at the following values: $epochs = 180$, $batch\_size = 100$, $hidden\_dimension = 128$, $\lambda = 0.1$

(b) All other hyperparameters were held constant at the following values: $epochs = 250$, $batch\_size = 100$, $hidden\_dimension = 128$, $\lambda = 0.3$

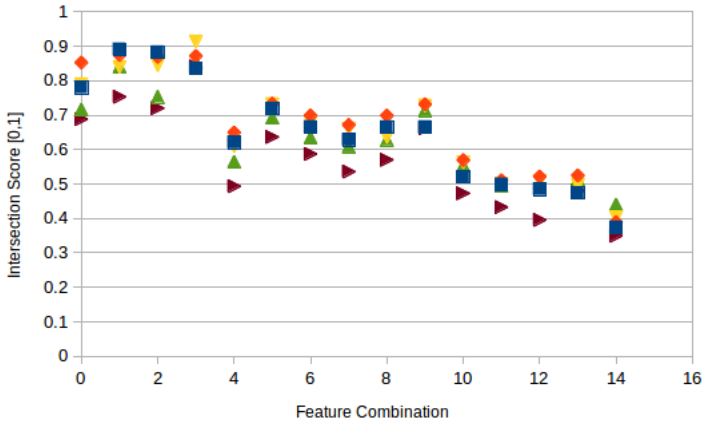
Figure 5.4: Learning Rate Parameter Search

dimension sizes leads to potential overfitting and raises the computational complexity of training the network.

The values tested for hidden dimension were $\{64, 128, 256, 384, 512\}$. The intersection vs. parameter setting plots for WGAN-GP and WGAN-GPMI may be seen in Fig. 5.5a and Fig. 5.5b respectively.



(a) All other hyperparameters were held constant at the following values: $epochs = 180$, $batch\_size = 100$, $learning\_rate = 5e{-}5$, $\lambda = 0.1$



(b) All other hyperparameters were held constant at the following values: $epochs = 250$, $batch\_size = 100$, $learning\_rate = 5e{-}4$, $\lambda = 0.3$

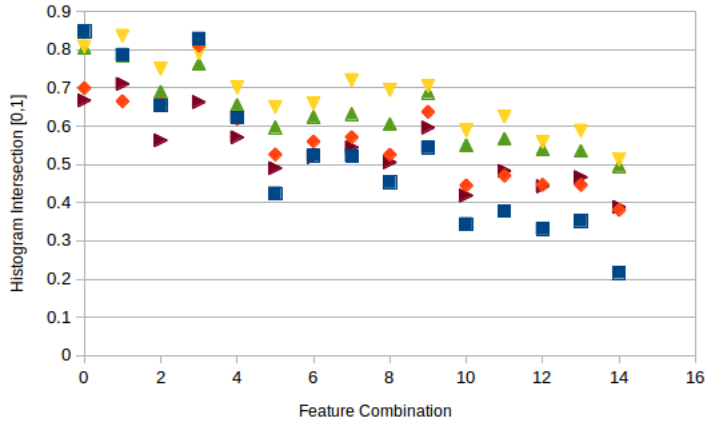
Figure 5.5: Hidden Dimension Parameter Search

The range of Histogram Intersection scores for the hidden parameter search is far smaller than the other hyperparameters. For the WGAN-GP model $\{128, 256, 384\}$ had the highest intersection scores, while $\{64, 128, 256\}$ performed well for the WGAN-GPMI model.

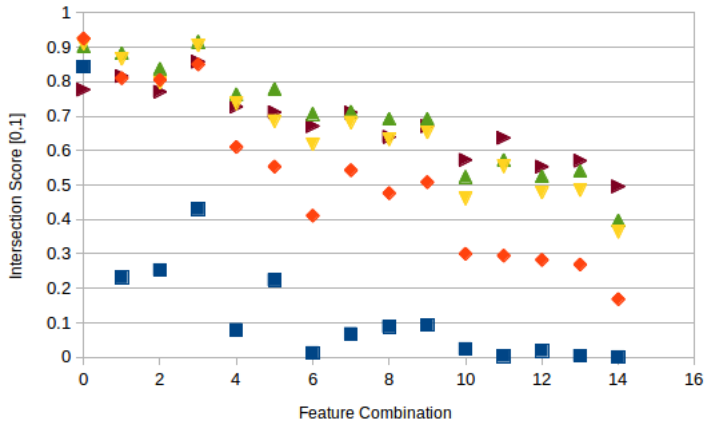
Epochs

The number of epochs determined how many times the network was exposed to the full dataset during training. Using a large number of epochs allows for the network to get more exposure to the ground truth distribution. However using a very large number of epochs can lead to overfitting.

Due to the small size of the CPTC dataset, large values for epochs were tested. These values included $\{50, 100, 150, 200, 250\}$. The intersection vs. parameter setting plots for WGAN-GP and WGAN-GPMI may be seen in Fig. 5.6a and Fig. 5.6b respectively.



(a) All other hyperparameters were held constant at the following values: $batch\_size = 100$, $learning\_rate = 5e{-}5$, $hidden\_dimension = 128$, $\lambda = 0.1$



(b) All other hyperparameters were held constant at the following values: $batch\_size = 100$, $learning\_rate = 5e{-}4$, $hidden\_dimension = 128$, $\lambda = 0.3$

Figure 5.6: Epochs Parameter Search

Increasing the number of epochs is critical for the WGAN-GPMI model to perform well. This makes intuitive sense, as this model requires the optimization of three neural networks. The 30 epochs test case highlights this, as nearly all 3-tuple and 4-tuple

feature combinations have 0 intersection with the ground truth distribution. Comparatively, the WGAN-GP model is able to achieve results within the range of $20 - 30$ percent; still the worst result of the values tested, but significantly better than WGAN-GPMI. For the WGAN-GP model, $\{100, 150, 200\}$ epochs were selected for the full hyperparameter sweep. For the WGAN-GPMI model, $\{150, 200, 250\}$ were selected.

Full Parameter Sweep

Collecting the candidate values from Sections 5.1.1 through 5.1.5, a full parameter search was carried out to test all combinations of these values. The values under test for each model may be seen in Table 5.3, along with the unique number of combinations tested.

Table 5.3: Candidate Parameters for WGAN-GP and WGAN-GPMI

WGAN-GP Parameters							**WGAN-GPMI Parameters**			
Lambda	0.05	0.1	0.2				0.2	0.3	0.4	
Batch Size	10	25	50	100	150		50	100		
Learning Rate	5e-5	5e-4	1e-3				5e-5	1e-4	5e-4	1e-3
Hidden Dimension	128	256	384				64	128	256	
Epochs	100	150	200				150	200	250	
Number of Unique Combinations			405				216			

For each parameter combination tested the intersection of histograms for all feature combinations was computed and tabulated. Given number of unique combinations tested a simple 3 step heuristic was defined to help identify the combinations which performed well in the table. For each combination of features:

1. The highest intersection score achieved was highlighted in yellow.
2. Intersection scores that fell within the 90^{th} percentile were highlighted in green
3. Intersection scores that fell within the 80^{th} percentile were highlighted in red.

This system reduced the search of possible combinations, and allowed for quick visual identification of values which performed well. Identifying the highest intersection score

per feature combination is not sufficient criteria to identify the best hyperparameter combination as scores are very close and can be noisy. Adding in the 90^{th} constraint helps to identify other candidates with high performing intersection scores. Finally, the 80^{th} percentile constraint is meant to act as a lower bound as values which fall within in it are still good, and values which are not highlighted at all should raise questions about model performance. For example, if the individual feature intersections are within the 90^{th} percentile but 3-tuple and 4-tuple combinations are completely not highlighted, the model may have failed to capture high order dependencies in the distribution of the data. Thus, that parameter setting should not be considered for use in future experiments.

A subsection of the highlighted table for the WGAN-GPMI model is given in Fig. 5.7. Note that the row with the red arrow pointing to it performs well, as 6 of it's 15 values are the highest performing intersection scores with an additional 6 in the 90^{th} percentile. The only feature which fails to achieve an intersection score within the 80^{th} percentile is Destination Service. Table 5.4 shows the optimal hyperparameter values selected for each model. Despite only testing on a single target IP address, these parameters were applied to all other target IP addresses and performed well.

Table 5.4: Optimal Hyperparameter Settings

(a)

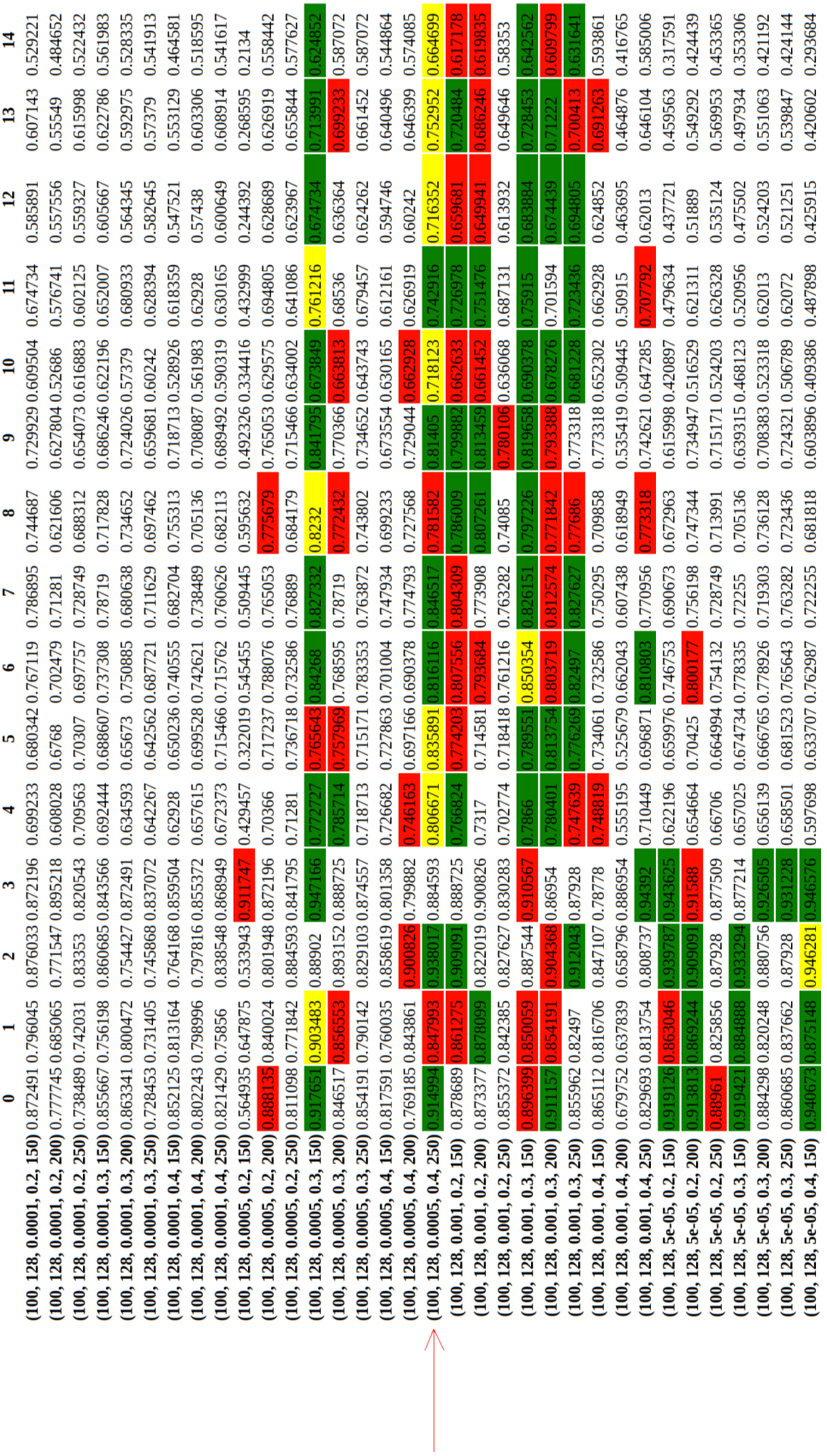
WGAN-GP	
Epochs	150
Batch Size	100
Learning Rate	5e-4
Lambda	0.1
Hidden Dimension	128

(b)

WGAN-GPMI	
Epochs	250
Batch Size	100
Learning Rate	5e-4
Lambda	0.4
Hidden Dimension	128

Alert Fidelity

Two metrics were employed in order to identify the fidelity of alert generation. Each metric was scaled such that it could be used to analyze individual feature performance as well as



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
(100, 128, 0.0001, 0.2, 150)	0.872491	0.796045	0.876033	0.872196	0.699233	0.680342	0.767119	0.786895	0.744687	0.729929	0.609504	0.674734	0.585891	0.607143	0.529221
(100, 128, 0.0001, 0.2, 200)	0.777745	0.685065	0.771547	0.895218	0.608028	0.6768	0.702479	0.71281	0.621606	0.627804	0.52686	0.576741	0.557556	0.55549	0.484652
(100, 128, 0.0001, 0.2, 250)	0.738489	0.742031	0.83353	0.820543	0.709563	0.70307	0.697757	0.728749	0.688312	0.654073	0.616883	0.602125	0.559327	0.615998	0.522432
(100, 128, 0.0001, 0.3, 150)	0.855667	0.756198	0.860685	0.843566	0.692444	0.688607	0.737308	0.78719	0.717828	0.686246	0.622196	0.652007	0.605667	0.622786	0.561983
(100, 128, 0.0001, 0.3, 200)	0.863341	0.800472	0.754427	0.872491	0.634593	0.65673	0.750885	0.680638	0.734652	0.724026	0.57379	0.680933	0.564345	0.592975	0.528335
(100, 128, 0.0001, 0.3, 250)	0.728453	0.731405	0.745868	0.837072	0.642267	0.642562	0.687721	0.711629	0.697462	0.659681	0.60242	0.628394	0.582645	0.57379	0.541913
(100, 128, 0.0001, 0.4, 150)	0.852125	0.813164	0.764168	0.859504	0.62928	0.650236	0.740555	0.682704	0.755313	0.718713	0.528926	0.618359	0.547521	0.553129	0.464581
(100, 128, 0.0001, 0.4, 200)	0.802243	0.798996	0.797816	0.855372	0.657615	0.699528	0.742621	0.738489	0.705136	0.708087	0.561983	0.62928	0.57438	0.603306	0.518595
(100, 128, 0.0001, 0.4, 250)	0.821429	0.75856	0.838548	0.868949	0.672373	0.715466	0.715762	0.760626	0.682113	0.689492	0.590319	0.630165	0.600649	0.608914	0.541617
(100, 128, 0.0005, 0.2, 150)	0.564935	0.647875	0.533943	0.911747	0.429457	0.322019	0.545455	0.509445	0.595632	0.492326	0.334416	0.432999	0.244392	0.268595	0.2134
(100, 128, 0.0005, 0.2, 200)	0.888135	0.840024	0.801948	0.872196	0.70366	0.717237	0.788076	0.765053	0.775679	0.765053	0.629575	0.694805	0.628689	0.626919	0.558442
(100, 128, 0.0005, 0.2, 250)	0.811098	0.771842	0.884593	0.841795	0.71281	0.736718	0.732586	0.76889	0.684179	0.715466	0.634002	0.641086	0.623967	0.655844	0.577627
(100, 128, 0.0005, 0.3, 150)	0.917651	0.903483	0.88902	0.947166	0.772727	0.765643	0.84268	0.827332	0.8232	0.841795	0.673849	0.761216	0.674734	0.713991	0.624852
(100, 128, 0.0005, 0.3, 200)	0.846517	0.856553	0.893152	0.888725	0.785714	0.757969	0.768595	0.78719	0.772432	0.770366	0.663813	0.68536	0.636364	0.699233	0.587072
(100, 128, 0.0005, 0.3, 250)	0.854191	0.790142	0.829103	0.874557	0.718713	0.715171	0.783353	0.763872	0.743802	0.734652	0.643743	0.679457	0.624262	0.661452	0.587072
(100, 128, 0.0005, 0.4, 150)	0.817591	0.760035	0.858619	0.801358	0.726682	0.727863	0.701004	0.747934	0.699233	0.673554	0.630165	0.612161	0.594746	0.640496	0.544864
(100, 128, 0.0005, 0.4, 200)	0.769185	0.843861	0.900826	0.799882	0.746163	0.697166	0.690378	0.774793	0.727568	0.729044	0.662928	0.626919	0.60242	0.646399	0.574085
(100, 128, 0.0005, 0.4, 250)	0.914994	0.847993	0.938017	0.884593	0.806671	0.835891	0.816116	0.846517	0.781582	0.81405	0.718123	0.742916	0.716352	0.752952	0.664699
(100, 128, 0.001, 0.2, 150)	0.878689	0.861275	0.909091	0.888725	0.766824	0.774203	0.807556	0.804309	0.786009	0.799882	0.662633	0.726978	0.659681	0.720484	0.617178
(100, 128, 0.001, 0.2, 200)	0.873377	0.878099	0.822019	0.900826	0.7317	0.714581	0.793684	0.773908	0.807261	0.813459	0.661452	0.751476	0.649941	0.686246	0.619835
(100, 128, 0.001, 0.2, 250)	0.855372	0.842385	0.827627	0.830283	0.702774	0.718418	0.761216	0.763282	0.74085	0.780106	0.636068	0.687131	0.613932	0.649646	0.58353
(100, 128, 0.001, 0.3, 150)	0.896399	0.850059	0.887544	0.910567	0.7866	0.789551	0.850354	0.826151	0.79726	0.819658	0.690378	0.75915	0.683884	0.728453	0.642562
(100, 128, 0.001, 0.3, 200)	0.911157	0.854191	0.904368	0.86954	0.780401	0.813754	0.803719	0.812574	0.771842	0.793388	0.678276	0.701594	0.674439	0.71222	0.609799
(100, 128, 0.001, 0.3, 250)	0.855962	0.82497	0.912043	0.87928	0.747639	0.776269	0.82497	0.827627	0.77686	0.773318	0.681228	0.723436	0.694805	0.700413	0.631641
(100, 128, 0.001, 0.4, 150)	0.865112	0.816706	0.847107	0.78778	0.748819	0.734061	0.732586	0.750295	0.709858	0.773318	0.652302	0.662928	0.624852	0.691263	0.593861
(100, 128, 0.001, 0.4, 200)	0.679752	0.637839	0.658796	0.886954	0.555195	0.525679	0.662043	0.607438	0.618949	0.535419	0.509445	0.50915	0.463695	0.464876	0.416765
(100, 128, 0.001, 0.4, 250)	0.829693	0.813754	0.808737	0.94392	0.710449	0.696871	0.810803	0.770956	0.773318	0.742621	0.647285	0.707792	0.62013	0.646104	0.585006
(100, 128, 5e-05, 0.2, 150)	0.919126	0.863046	0.939787	0.943625	0.622196	0.659976	0.746753	0.690673	0.672963	0.615998	0.420897	0.479634	0.437721	0.459563	0.317591
(100, 128, 5e-05, 0.2, 200)	0.913813	0.869244	0.909091	0.91588	0.654664	0.70425	0.800177	0.756198	0.747344	0.734947	0.516529	0.621311	0.51889	0.549292	0.424439
(100, 128, 5e-05, 0.2, 250)	0.88961	0.825856	0.87928	0.877509	0.66706	0.664994	0.754132	0.728749	0.713991	0.715171	0.524203	0.626328	0.535124	0.569953	0.453365
(100, 128, 5e-05, 0.3, 150)	0.919421	0.884888	0.933294	0.877214	0.657025	0.674734	0.778335	0.72255	0.705136	0.639315	0.468123	0.520956	0.475502	0.497934	0.353306
(100, 128, 5e-05, 0.3, 200)	0.884298	0.820248	0.880756	0.926505	0.656139	0.666765	0.778926	0.719303	0.736128	0.708383	0.523318	0.62013	0.524203	0.551063	0.421192
(100, 128, 5e-05, 0.3, 250)	0.860685	0.837662	0.87928	0.931228	0.658501	0.681523	0.765643	0.763282	0.723436	0.724321	0.506789	0.62072	0.521251	0.539847	0.424144
(100, 128, 5e-05, 0.4, 150)	0.940673	0.875148	0.946281	0.946576	0.597698	0.633707	0.762987	0.722255	0.681818	0.603896	0.409386	0.487898	0.425915	0.420602	0.293684

Figure 5.7: Subsection of WGAN-GPMI Hyperparameter Search Results

m-tuple combinations of features. This allowed for both low and high level performance of the model to be assessed. These metrics were the Histogram Intersection and Jensen Shannon Divergence.

Histogram Intersection

The Histogram Intersection was computed for all feature combinations across all 4 IP addresses tested. For each IP, all possible combinations of features were analyzed. This resulted in 4 intersections representing the individual features, 6 representing pairs of features, 4 representing 3-tuples, and a single histogram representing 4-tuple combinations.

Fig. 5.8 steps through these levels of combinations for target 10.0.0.100. The top left plot shows the intersection of Time Bins. The top right plot shows the histogram combination of Time Bins and Destination Service. The bottom right adds the Alert Signature feature to the histogram. And finally, the bottom right shows all 4 features under test as a single joint histogram. It is important to note that as the number of features considered in the combination increases so does the complexity of recreating the data with high fidelity; individual occurrences of features drops, while the number of unique feature values to output rises.

The Histogram Intersection was used to analyze the fidelity of results from both of the models tested. In general, the WGAN-GPMI is able to outperform the standard WGAN-GP model. This suggests that the mutual information constraint results in a model which manages to emulate the ground truth distribution with a higher degree of accuracy than standard models are. Table 5.5 summarizes the results of both models across all 4 target IP addresses. The maximum intersection score for each combination of features is bolded if the given score is at least 0.05 greater than the intersection score of the other model. Additionally, the standard deviation was computed using 1000 unique distributions sampled from the generative model and is shown for each feature combination on each target.

It is interesting to note that the effect of the mutual information constraint varies from

Table 5.5: Histogram Intersection for all Feature Combinations

	Victim Machine IP Address									
	WGAN-GP				WGAN-GPMI					
Features	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**
A	0.697 ± 0.002	0.658 ± 0.007	0.844 ± 0.005	$\mathbf{0.858 \pm 0.009}$	$\mathbf{0.890 \pm 0.006}$	$\mathbf{0.833 \pm 0.005}$	$\mathbf{0.847 \pm 0.006}$	0.808 ± 0.009
D	0.772 ± 0.007	0.660 ± 0.006	0.843 ± 0.005	$\mathbf{0.905 \pm 0.009}$	$\mathbf{0.899 \pm 0.006}$	$\mathbf{0.846 \pm 0.005}$	0.823 ± 0.007	0.827 ± 0.009
S	0.717 ± 0.007	0.867 ± 0.009	$\mathbf{0.846 \pm 0.008}$	0.843 ± 0.009	$\mathbf{0.906 \pm 0.008}$	0.909 ± 0.005	0.755 ± 0.005	0.881 ± 0.010
T	0.814 ± 0.008	0.760 ± 0.007	0.818 ± 0.008	0.723 ± 0.009	$\mathbf{0.892 \pm 0.008}$	$\mathbf{0.815 \pm 0.007}$	0.844 ± 0.008	$\mathbf{0.819 \pm 0.009}$
A,T	0.668 ± 0.007	0.630 ± 0.007	0.741 ± 0.007	0.630 ± 0.008	$\mathbf{0.774 \pm 0.008}$	$\mathbf{0.774 \pm 0.007}$	0.754 ± 0.008	$\mathbf{0.717 \pm 0.010}$
A,S	0.634 ± 0.007	0.590 ± 0.007	$\mathbf{0.774 \pm 0.004}$	0.757 ± 0.008	$\mathbf{0.791 \pm 0.008}$	$\mathbf{0.747 \pm 0.007}$	0.718 ± 0.008	0.771 ± 0.010
S,D	0.698 ± 0.007	0.598 ± 0.007	$\mathbf{0.768 \pm 0.004}$	0.779 ± 0.009	$\mathbf{0.829 \pm 0.007}$	$\mathbf{0.758 \pm 0.005}$	0.715 ± 0.005	0.800 ± 0.010
D,T	0.710 ± 0.007	0.631 ± 0.006	0.768 ± 0.004	0.659 ± 0.009	$\mathbf{0.790 \pm 0.008}$	$\mathbf{0.777 \pm 0.007}$	0.736 ± 0.008	$\mathbf{0.726 \pm 0.009}$
S,T	0.710 ± 0.007	0.702 ± 0.008	0.741 ± 0.007	0.698 ± 0.009	$\mathbf{0.778 \pm 0.008}$	$\mathbf{0.791 \pm 0.006}$	0.701 ± 0.007	$\mathbf{0.782 \pm 0.010}$
A,D	0.693 ± 0.002	0.637 ± 0.006	0.828 ± 0.005	$\mathbf{0.830 \pm 0.008}$	$\mathbf{0.825 \pm 0.008}$	$\mathbf{0.822 \pm 0.006}$	0.820 ± 0.008	0.777 ± 0.010
A,S,T	0.599 ± 0.007	0.558 ± 0.007	0.686 ± 0.007	0.580 ± 0.008	$\mathbf{0.655 \pm 0.008}$	$\mathbf{0.727 \pm 0.006}$	0.683 ± 0.008	$\mathbf{0.632 \pm 0.010}$
A,S,D	0.627 ± 0.007	0.573 ± 0.007	$\mathbf{0.761 \pm 0.004}$	0.734 ± 0.009	$\mathbf{0.733 \pm 0.008}$	$\mathbf{0.737 \pm 0.006}$	0.697 ± 0.008	0.740 ± 0.010
A,D,T	0.653 ± 0.007	0.615 ± 0.006	0.756 ± 0.007	0.612 ± 0.008	$\mathbf{0.715 \pm 0.008}$	$\mathbf{0.731 \pm 0.006}$	0.731 ± 0.007	$\mathbf{0.685 \pm 0.010}$
S,D,T	0.611 ± 0.007	0.569 ± 0.007	$\mathbf{0.690 \pm 0.007}$	0.597 ± 0.008	0.652 ± 0.008	$\mathbf{0.734 \pm 0.007}$	0.632 ± 0.008	0.635 ± 0.010
A,S,D,T	0.584 ± 0.007	0.548 ± 0.007	0.601 ± 0.007	0.571 ± 0.008	$\mathbf{0.607 \pm 0.008}$	$\mathbf{0.718 \pm 0.006}$	0.626 ± 0.008	0.615 ± 0.010

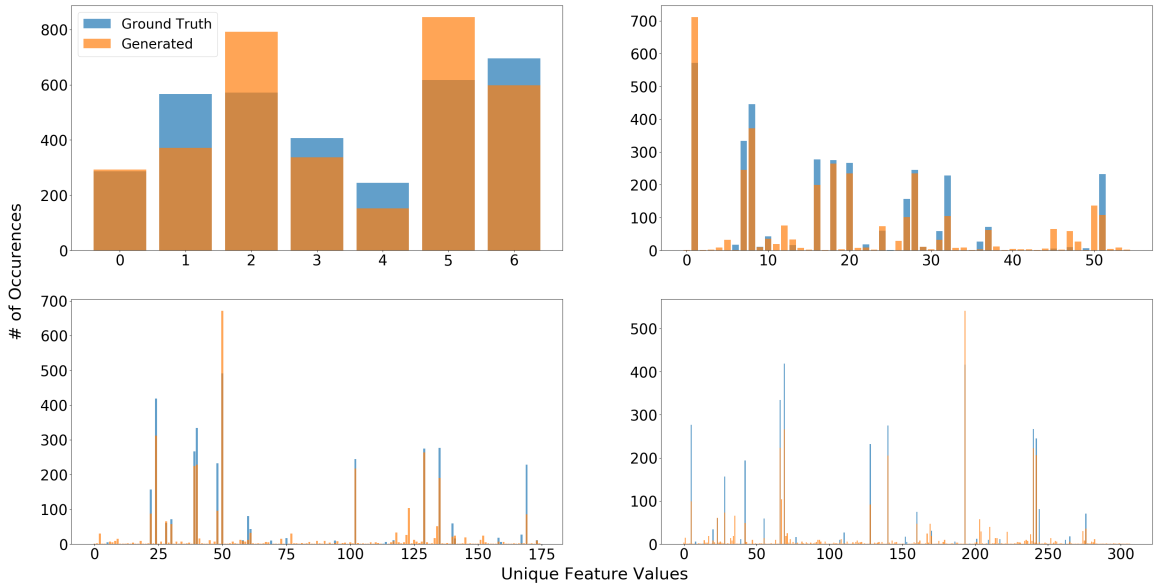


Figure 5.8: Histogram Intersection for M-Tuple Feature Combinations

Target IP address to Target IP address. For example, Target 10.0.0.22 only has small improvements to Histogram Intersection when using the WGAN-GPMI model. In several cases, such as Source IP and Destination Service, the intersection score actually drops. On the other hand Target IP addresses such as 10.0.0.27 see a large benefit from using the mutual information constraint. On average, the Histogram Intersection is 14.63% higher for the WGAN-GPMI model than it's WGAN-GP counterpart. This is result is particularly interesting, as the intent of the mutual information constraint is to improve mode dropping in the generator, not to directly improve Histogram Intersection. It is believed that palliating mode dropping is directly related to increasing Histogram Intersection in many cases because it redistributes output sample entropy across more output values than standard models do when collapsing to a small subset of output values.

Another interesting result of Table 5.5 is that the intersection of histograms is resilient to earlier score bias. Consider the intersection score of Timestamp (T) on victim IP 10.0.0.100. This feature has the highest score of any single feature, potentially leading to the fallacious expectation that any combination with T will also score high. When moving to testing 2-tuple combinations such as Timestamp (T) and Alert Signature (A)

however the intersection drops significantly. This leads to another interesting observation; the Histogram Intersection is monotonically decreasing with respect to it's constituent features. Intuitively this makes sense as adding new features to the combination cannot add any more information than what is provided by the constituent features.

Jensen Shannon Divergence

Due to the shortcomings of Histogram Intersection discussed in Section 5.2 the Jensen Shannon Divergence is also computed for each model. Table 5.6 shows the results of this computation. The minimum divergence for each feature combination is highlighted if the value is at least 0.01 less than the corresponding value for the other model. Additionally, the standard deviation of divergences was computed for each feature combination and target, for both models, using 1000 distributions sampled from the generative model.

Note that the Jensen Shannon Divergence does not have an upper bound like the Histogram Intersection does, however it's result can be interpreted as the amount of information required, in nats, to get from one distribution to another. For disparate distributions, the divergence is high, as a large amount of information is required to get from one distribution to the other. Histograms that have a similar probability distribution have a low amount of divergence, as less information is required to get from one distribution to another; the Jensesn Shannon Divergence follows a reverse trend from the Histogram Intersection.

One of the largest benefits of the Jensen Shannon Divergence is that it imposes a nonlinear penalty based off the difference in histogram probability distributions. Directly applied to each of the models presented here, a large penalty is incurred if the model fails to output a value that occurs with non-zero probability in the ground truth. This benefit is especially apparent for Target IP 10.0.0.100 where the Jensen Shannon Divergence shows a greater improvement for 4-tuple results when using the WGAN-GPMI model than the Histogram Intersection does. This will be further highlighted in Section 5.4 where the number of output modes is shown to increase as much as two fold when using the WGAN-GPMI model.

Table 5.6: Jensen Shannon Divergence (nats) for all Feature Combinations

	Victim Machine IP Address									
	WGAN-GP				WGAN-GPMI					
Features	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**
A	0.111 ± 0.002	0.088 ± 0.003	0.033 ± 0.002	0.024 ± 0.002	$\mathbf{0.017 \pm 0.001}$	$\mathbf{0.031 \pm 0.002}$	0.031 ± 0.002	0.033 ± 0.002
D	0.052 ± 0.002	0.086 ± 0.003	0.032 ± 0.002	0.012 ± 0.002	$\mathbf{0.006 \pm 0.001}$	$\mathbf{0.026 \pm 0.002}$	0.041 ± 0.002	0.021 ± 0.002
S	0.085 ± 0.002	0.022 ± 0.002	0.029 ± 0.001	0.017 ± 0.002	$\mathbf{0.010 \pm 0.001}$	$\mathbf{0.014 \pm 0.001}$	0.037 ± 0.002	0.011 ± 0.002
T	0.049 ± 0.002	0.047 ± 0.003	0.028 ± 0.002	0.060 ± 0.003	$\mathbf{0.009 \pm 0.001}$	$\mathbf{0.030 \pm 0.002}$	0.019 ± 0.002	$\mathbf{0.023 \pm 0.002}$
A,T	0.150 ± 0.002	0.128 ± 0.003	$\mathbf{0.063 \pm 0.003}$	0.131 ± 0.004	$\mathbf{0.077 \pm 0.003}$	$\mathbf{0.067 \pm 0.003}$	0.080 ± 0.003	$\mathbf{0.071 \pm 0.003}$
A,S	0.145 ± 0.002	0.150 ± 0.003	0.070 ± 0.002	0.071 ± 0.003	$\mathbf{0.059 \pm 0.003}$	$\mathbf{0.076 \pm 0.003}$	$\mathbf{0.067 \pm 0.003}$	0.063 ± 0.003
S,D	0.107 ± 0.002	0.141 ± 0.003	0.071 ± 0.002	0.054 ± 0.003	$\mathbf{0.036 \pm 0.002}$	$\mathbf{0.063 \pm 0.003}$	0.078 ± 0.003	$\mathbf{0.041 \pm 0.003}$
D,T	0.121 ± 0.002	0.126 ± 0.003	$\mathbf{0.063 \pm 0.003}$	0.126 ± 0.004	$\mathbf{0.069 \pm 0.003}$	$\mathbf{0.063 \pm 0.003}$	0.091 ± 0.003	$\mathbf{0.063 \pm 0.003}$
S,T	0.102 ± 0.002	0.082 ± 0.003	$\mathbf{0.073 \pm 0.003}$	0.077 ± 0.003	$\mathbf{0.049 \pm 0.002}$	$\mathbf{0.053 \pm 0.003}$	0.096 ± 0.004	$\mathbf{0.039 \pm 0.003}$
A,D	0.142 ± 0.002	0.110 ± 0.003	0.044 ± 0.002	0.054 ± 0.003	$\mathbf{0.060 \pm 0.003}$	$\mathbf{0.050 \pm 0.002}$	0.047 ± 0.002	0.055 ± 0.003
A,S,T	0.189 ± 0.002	0.181 ± 0.004	$\mathbf{0.119 \pm 0.003}$	0.193 ± 0.005	$\mathbf{0.146 \pm 0.004}$	$\mathbf{0.110 \pm 0.003}$	0.158 ± 0.004	$\mathbf{0.136 \pm 0.004}$
A,S,D	0.173 ± 0.002	0.169 ± 0.004	0.080 ± 0.003	0.100 ± 0.004	$\mathbf{0.102 \pm 0.003}$	$\mathbf{0.092 \pm 0.003}$	0.081 ± 0.003	$\mathbf{0.082 \pm 0.004}$
A,D,T	0.173 ± 0.003	0.141 ± 0.004	$\mathbf{0.072 \pm 0.003}$	0.157 ± 0.004	$\mathbf{0.113 \pm 0.004}$	$\mathbf{0.081 \pm 0.003}$	0.095 ± 0.004	$\mathbf{0.090 \pm 0.004}$
S,D,T	0.174 ± 0.003	0.178 ± 0.004	$\mathbf{0.114 \pm 0.003}$	0.197 ± 0.005	$\mathbf{0.141 \pm 0.003}$	$\mathbf{0.106 \pm 0.003}$	0.160 ± 0.003	$\mathbf{0.132 \pm 0.004}$
A,S,D,T	0.209 ± 0.003	0.194 ± 0.004	$\mathbf{0.121 \pm 0.003}$	0.213 ± 0.005	$\mathbf{0.178 \pm 0.004}$	$\mathbf{0.122 \pm 0.003}$	0.164 ± 0.004	$\mathbf{0.150 \pm 0.005}$

Alert Dependencies

Interactions between NIDS alert features provide important contextual information about cyber attacks. For example, knowing what machine an attack will occur on can reduce the number of possible attack vectors on that machine. Capturing these feature dependencies is a key aspect of understanding the quality of the generative model. To verify these dependencies the drop in Histogram Intersection is noted, weighted conditional entropy is computed according to the equations provided by (4.2) and (4.4), and joint entropy is computed using (4.3) and (4.5) to provide a baseline. Conditional probability tables are also generated to provide detailed inspection of specific feature value relationships. Appendix A contains an alternative method to verifying feature dependency through the usage of a Support Vector Machine. These metrics were all computed for both WGAN-GP and WGAN-GPMI models.

WGAN-GP Feature Dependency Performance

By viewing the difference in Histogram Intersection between $m \pm 1$ tuples of features, feature dependencies can be inferred. Large drops indicate a low amount of dependence between the lower levels of feature combinations and the higher level. Small drops when adding a new feature to the joint distribution indicate that there is dependence between the lower level combination and the newly added feature. By applying the graph structure given in Fig. 4.1 to NIDS alert features, Histogram Intersection scores can be noted in nodes of the graph while edges represent conditional and joint entropy. Fig. 5.9 shows the dependency graph for Target IP 10.0.0.27 from the CPTC'17 dataset. These graphs were constructed for the other three target IPs tested from CPTC '17 and are included in the Appendix B

Stepping through the graph, nodes along the outer edges represent a single feature histogram. The values given are the Histogram Intersection between the ground truth and

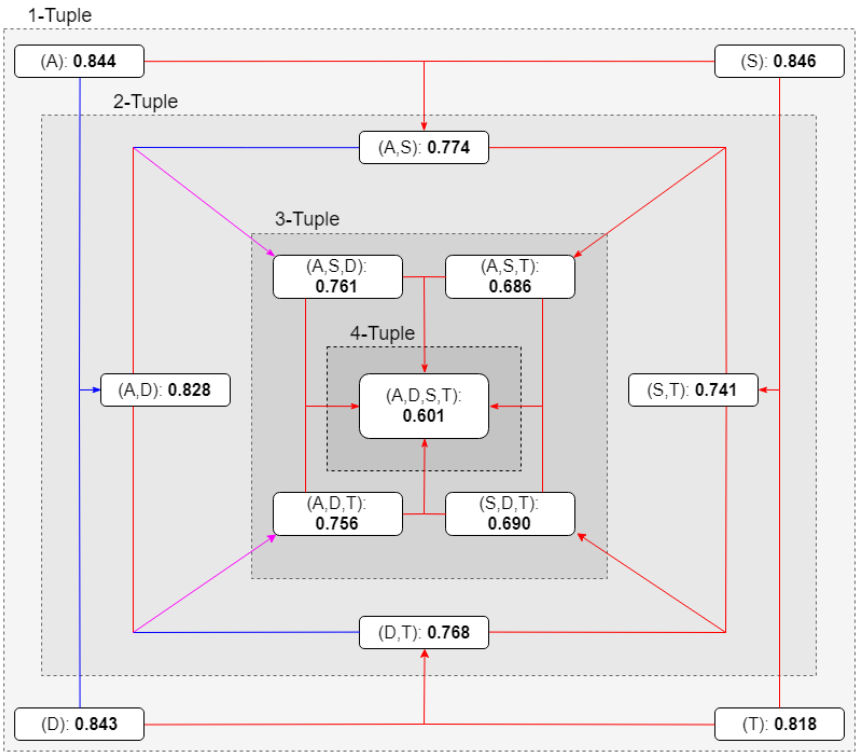


Figure 5.9: Target 10.0.0.27 Alert Dependency Graph: WGAN-GP Results

generated data distribution. As edges are traversed and meet, they are combined and arrive at the next node as a combination of the features considered. This process continues inwards, adding one more unique feature to each joint distribution, until all four features are considered at once.

It is important to note that for the 2-tuple combinations there are actually 6 possible distributions, however only 4 may be visualized while maintaining planarity of the graph; the analysis of these values is still included in the conditional and joint probability tables.

In order to highlight drops in intersection score, the following schema was applied to the graph: Lines are color coded such that blue lines indicate feature unions which result in less than 5% difference between histogram scores. Conversely, lines which are red indicate a difference that is greater than 5%. Lines which are purple indicate a unidirectional dependence relationship between the feature tuples. This occurs when one feature is a good predictor of another ($<$ 5% intersection difference) but the opposite is not true ($>$ 5% in-

tersection difference). Lines which are blue or red exhibit bidirectional dependence; both lines exhibit the same intersection difference. All dashed lines are bounding boxes added to clearly segment the varying m-tuple histograms.

To verify that the graph correctly highlights feature dependencies Table 5.7 was computed. This table considers all unique permutations of features for all m-tuples across all 4 target IPs. The conditional entropy is then compared between the ground truth and generated data distributions. Additionally, the joint entropy was computed in Table 5.8 to provide a baseline representation of the amount of randomness in the feature distributions. This table contains calculations only from 2-tuple, 3-tuple, and 4-tuple combinations as there is no direct comparison to draw between conditional and joint entropy for single feature histograms.

Several instances of feature dependency show good agreement between Fig. 5.9 and Table 5.7. For example, the drop in interesection score between A,S and A,S,T is low and the conditional entropy of the ground truth distribution T|A,S is low.

To further study specific feature-value relationships between the ground truth and generated data distributions conditional probability tables may be output for each feature permutation. In order to make these tables human-readable the following color schema is applied: cells which have probability 0 are highlighted in red; cells with probability 1 are highlighted in blue. This two step process highlights both the sparsity of output probabilities as well as cases where the input results in a deterministic output. Specific examples of these tables for the ground truth distribution, WGAN-GP generated distribution, and WGAN-GPMI generated distribution will be covered in 5.3.2

WGAN-GPMI Feature Dependency Performance

All of the aforementioned plots and tables were created in duplicate in order to test the effect of Mutual Information maximization on the model's ability to capture feature dependencies.

Table 5.7: Weighted Normalized Conditional Entropy Values for all Target IPs: WGAN-GP Result

	Victim Machine IP Address								
	Ground Truth Results				Generated Results				
Features	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**
A|T	0.244	0.238	0.153	0.333	0.260	0.288	0.196	0.539
T|S	0.593	**0.463**	0.515	0.695	0.706	**0.528**	0.613	0.560
T|A	0.330	0.339	0.695	0.246	0.567	0.536	0.824	0.359
S|T	0.262	0.252	0.263	0.405	0.220	0.211	0.274	0.485
S|A	0.800	0.752	0.831	0.711	0.280	0.340	0.572	0.367
D|S	0.346	0.445	0.253	0.278	0.558	0.378	0.203	0.658
A|D	0.080	**0.222**	0.070	0.288	0.140	**0.109**	0.074	0.181
T|D	0.479	0.346	0.655	0.383	0.581	0.542	0.822	0.426
D|T	0.287	0.234	0.152	0.379	0.382	0.276	0.176	0.614
A|S	0.346	**0.385**	0.271	0.475	0.422	**0.390**	0.233	0.620
S|D	0.822	0.779	0.856	0.785	0.309	0.346	0.567	0.379
D|A	0.006	**0.246**	0.006	0.016	0.240	**0.091**	0.082	0.160
S|A,D	0.799	0.747	0.829	0.705	0.278	0.327	0.562	0.342
D|S,T	0.171	0.118	0.013	0.149	0.347	0.257	0.143	0.539
S|D,T	0.107	0.025	0.101	0.024	0.158	0.190	0.246	0.335
T|A,D	0.316	0.335	0.650	0.246	0.541	0.515	0.815	0.341
A|S,D	0.069	0.206	0.056	0.245	0.124	0.399	0.066	0.165
A|S,T	0.131	**0.117**	0.013	0.130	0.223	**0.267**	0.165	0.468
A|D,T	0.038	**0.018**	0.001	0.004	0.102	**0.083**	0.061	0.126
D|A,S	0.005	**0.243**	0.005	0.012	0.239	**0.081**	0.070	0.147
T|S,D	0.393	0.340	0.587	0.348	0.496	0.399	0.573	0.396
D|A,T	0.004	0.238	0.003	0.007	0.214	0.071	0.069	0.147
S|A,T	0.211	0.178	0.100	0.228	0.151	0.186	0.253	0.324
T|A,S	0.365	**0.312**	0.561	0.302	0.494	**0.396**	0.578	0.330
A|S,D,T	0.041	**0.172**	0.028	0.195	0.097	**0.076**	0.051	0.110
D|A,S,T	0.002	0.232	0.002	0.004	0.212	0.067	0.053	0.134
S|A,D,T	0.209	0.167	0.498	0.222	0.467	0.182	0.240	0.299
T|A,S,D	0.362	0.302	0.558	0.294	0.147	0.381	0.566	0.312

Table 5.8: Normalized Joint Entropy Values for all Victim IPs: WGAN-GP Result

	Victim Machine IP Address								
	Ground Truth Results				Generated Results				
Features	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**
A,D	0.715	0.652	0.469	0.747	0.492	0.398	0.191	0.553
A,S	0.704	0.683	0.674	0.749	0.462	0.531	0.456	0.652
A,T	0.783	0.781	0.839	0.750	0.617	0.603	0.648	0.638
D,T	0.811	0.807	0.858	0.767	0.682	0.592	0.608	0.672
S,D	0.764	0.711	0.686	0.738	0.553	0.533	0.444	0.652
S,T	0.816	0.801	0.842	0.773	0.672	0.642	0.665	0.593
A,D,T	0.805	0.775	0.839	0.750	0.603	0.538	0.547	0.595
A,S,D	0.735	0.683	0.674	0.749	0.508	0.478	0.389	0.579
A,S,T	0.780	0.750	0.810	0.740	0.595	0.595	0.595	0.620
S,D,T	0.806	0.775	0.810	0.748	0.646	0.595	0.581	0.647
A,S,D,T	0.803	0.750	0.810	0.740	0.605	0.563	0.553	0.611

First, the graph structure and highlighting in Fig. 5.9 was applied to the histogram intersection scores of the WGAN-GPMI model in Fig. 5.10. Similar to the WGAN-GP model results, Appendix B contains the results for the other three target IP addresses tested. Note that from this figure alone it is impossible to make a claim that feature dependencies are more fully captured by the WGAN-GPMI model. However by computing the conditional entropy in Table 5.9 it is apparent that the WGAN-GPMI model more closely imitates the entropy of the ground truth. In fact, several of the small valued m-tuples such as A|T, T|D, and D|S,T all have identical conditional entropy values to the ground truth distribution. Additionally, values that are within 10% of the higher entropy value are highlighted.

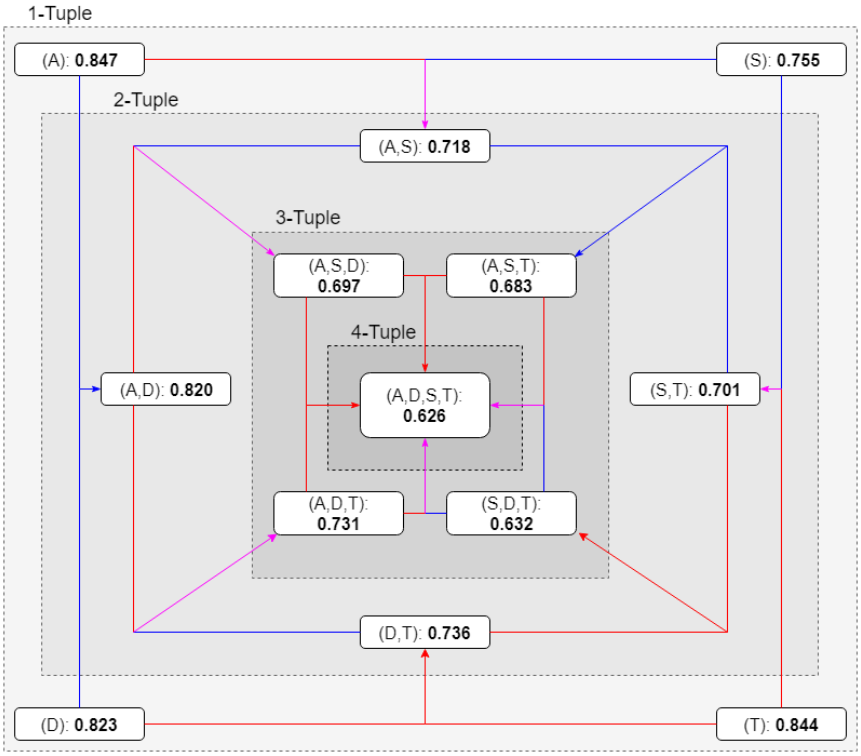


Figure 5.10: Target 10.0.0.27 Alert Dependency Graph: WGAN-GPMI Results

Despite identical conditional entropy values existing between the ground truth and generated distribution it is important to note that this does not guarantee that the distributions are identical. Fundamentally, two different distributions can have the same entropy. Additionally, the conditional entropy calculation used is weighted by the probability that a given

Table 5.9: Weighted Normalized Conditional Entropy Values for all Target IPs: WGAN-GPMI Result

	Victim Machine IP Address									
	Ground Truth Results				Generated Results					
Features	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**		
A|T	**0.244**	**0.238**	**0.153**	**0.333**	**0.244**	**0.238**	**0.153**	**0.334**		
T|S	**0.593**	**0.463**	**0.515**	**0.695**	**0.593**	**0.463**	**0.516**	**0.695**		
T|A	**0.330**	**0.339**	**0.695**	**0.246**	**0.330**	**0.339**	**0.695**	**0.246**		
S|T	**0.262**	0.252	**0.263**	**0.405**	**0.263**	0.186	**0.252**	**0.406**		
S|A	0.800	0.752	0.831	0.711	0.229	0.239	0.526	0.222		
D|S	0.346	0.445	0.253	0.278	0.509	0.385	0.207	0.558		
A|D	0.080	0.222	0.070	0.288	0.149	0.026	0.007	0.097		
T|D	**0.479**	**0.346**	**0.655**	**0.383**	**0.479**	**0.346**	**0.655**	**0.383**		
D|T	**0.287**	**0.234**	**0.152**	**0.379**	**0.287**	**0.234**	**0.152**	**0.379**		
A|S	0.346	**0.385**	0.271	0.475	0.403	**0.376**	0.214	0.543		
S|D	0.822	0.779	0.856	0.785	0.436	0.260	0.474	0.301		
D|A	0.006	0.246	0.006	0.016	0.055	0.009	0.048	0.000		
S|A,D	0.799	0.747	0.829	0.705	0.228	0.226	0.474	0.222		
D|S,T	**0.171**	**0.118**	**0.013**	**0.149**	**0.171**	**0.118**	**0.013**	**0.149**		
S|D,T	**0.107**	**0.025**	**0.101**	**0.024**	**0.107**	**0.025**	**0.101**	**0.024**		
T|A,D	**0.316**	**0.335**	**0.650**	**0.246**	**0.316**	**0.335**	**0.650**	**0.246**		
A|S,D	0.069	0.206	0.056	0.245	0.018	0.003	0.007	0.055		
A|S,T	0.131	**0.117**	**0.013**	**0.130**	0.112	**0.117**	**0.013**	**0.131**		
A|D,T	**0.038**	**0.018**	**0.001**	**0.004**	**0.038**	**0.018**	**0.001**	**0.004**		
D|A,S	0.005	0.243	0.005	0.012	0.054	0.000	0.000	0.000		
T|S,D	0.393	0.340	0.587	0.348	0.176	0.144	0.334	0.170		
D|A,T	0.004	0.238	0.003	0.007	0.044	0.006	0.000	0.000		
S|A,T	0.211	0.178	**0.100**	0.228	0.055	0.012	**0.100**	0.019		
T|A,S	0.365	0.312	0.561	0.302	0.170	0.144	0.328	0.089		
A|S,D,T	0.041	0.172	0.028	0.195	0.005	0.003	0.000	0.001		
D|A,S,T	0.002	0.232	0.002	0.004	0.044	0.000	0.000	0.000		
T|A,D,S	0.209	0.167	0.498	0.222	0.157	0.144	0.328	0.089		
S|A,T,D	0.362	0.302	0.558	0.294	0.055	0.004	0.100	0.019		

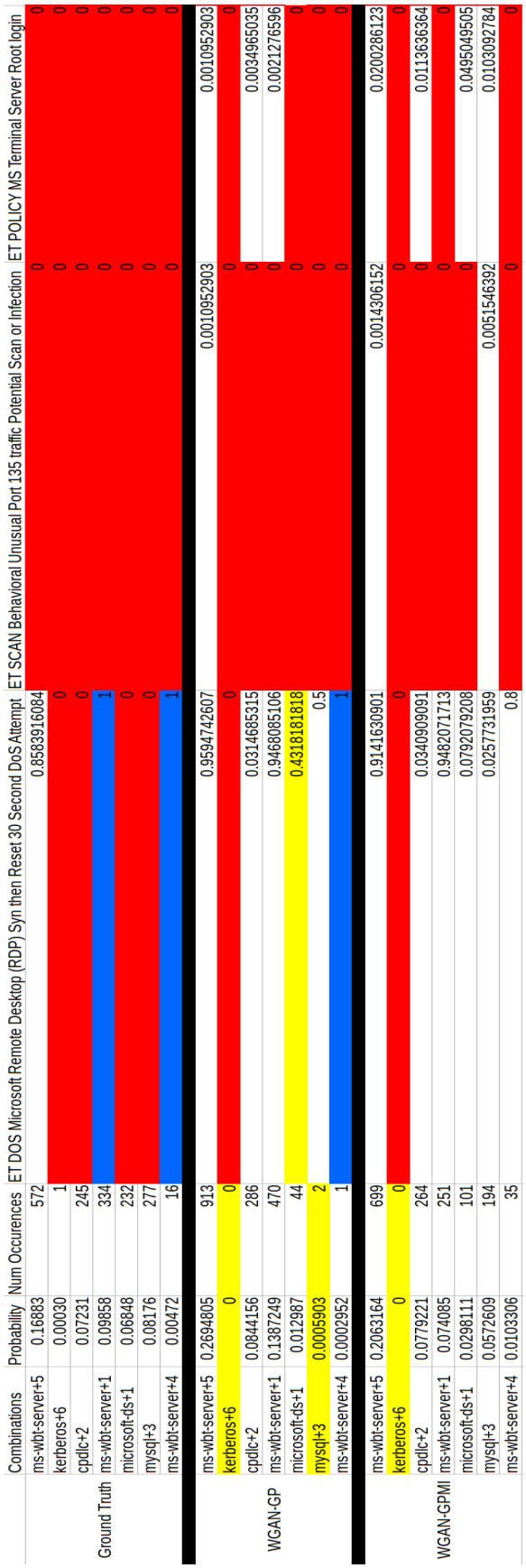
Table 5.10: Normalized Joint Entropy Values for all Victim IPs: WGAN-GPMI Result

	Victim Machine IP Address							
	Ground Truth Results				Generated Results			
Features	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**
A,D	0.715	0.652	0.469	0.747	0.532	0.457	0.192	0.453
A,S	0.704	0.683	0.674	0.749	0.609	0.548	0.367	0.566
A,T	0.783	0.781	0.839	0.750	0.693	0.616	0.665	0.554
D,T	0.811	0.807	0.858	0.767	0.733	0.611	0.619	0.572
S,D	0.764	0.711	0.686	0.738	0.657	0.552	0.353	0.567
S,T	0.816	0.801	0.842	0.773	0.722	0.589	0.656	0.618
A,D,T	0.805	0.775	0.839	0.750	0.651	0.555	0.577	0.521
A,S,D	0.735	0.683	0.674	0.749	0.591	0.491	0.327	0.499
A,S,T	0.780	0.750	0.810	0.740	0.678	0.579	0.597	0.564
S,D,T	0.806	0.775	0.810	0.748	0.712	0.582	0.588	0.578
A,S,D,T	0.803	0.750	0.810	0.740	0.680	0.557	0.568	0.547

input conditioning value occurs. This allows inputs that occur frequently to have more influence over the conditional entropy value than inputs that occur rarely. Given a very small chance of an input occurring it's contribution to the conditional entropy calculation could become negligible. In order to better understand each model, the conditional probability tables showing inputs and their respective output probability were generated. Figure 5.11 shows subsections of the conditional probability table's from the ground truth, WGAN-GP generated, and WGAN-GPMI generated distributions.

The figure shows the conditional probability table for A|D,T. All of the tables are formatted such that the input feature combination is given in the first column, the probability of that input occurring is given by the second column, the third column indicates the number of times that input occurred, and the remaining columns indicate the output values and their probability of occurring. This allows for identification of the influence each input combination would have on the conditional probability calculation. Cells that are highlighted red have no probability of being output given the input conditioning values. Cells that are highlighted blue indicate cells that are deterministically output given specific input conditioning values.

Note that the ground truth distribution exhibits deterministic behavior for many of the input feature combinations. Capturing these cases of deterministic behavior is especially important for inputs which occur frequently. For example, successfully learning the dependency between ms-wbt-server and time bin 1 predicting the value of the correct alert signature is important to model accuracy. The WGAN-GP generated distribution exploits this and generates many more samples than the ground truth contains, while the WGAN-GPMI distribution performs far closer to the real probability of occurrence. Additionally, several input combinations which the models fail to generate are highlighted in the table. Note that the input combination of kerberos+6 only occurs once in the ground truth, making it nearly impossible for either model to learn to recreate this value. But other cases such as mysql+3 occur 277 times in the ground truth distribution but only twice in the WGAN-GP



	Combinations	Probability	Num Occurences	ET DOS Microsoft Remote Desktop (RDP) Syn then Reset 30 Second DoS Attempt	ET SCAN Behavioral Unusual Port 135 traffic Potential Scan or Infection	ET POLICY MS Terminal Server Root login
Ground Truth	ms-wbt-server+5	0.16883	572	0.8583916084	0	0
	kerberos+6	0.00030	1	0	0	0
	cpdlc+2	0.07231	245	0	0	0
	ms-wbt-server+1	0.09858	334	1	0	0
	microsoft-ds+1	0.06848	232	0	0	0
	mysql+3	0.08176	277	0	0	0
	ms-wbt-server+4	0.00472	16	1	0	0
WGAN-GP	ms-wbt-server+5	0.2694805	913	0.9594742607	0.0010952903	0.0010952903
	kerberos+6	0	0	0	0	0
	cpdlc+2	0.0844156	286	0.0314685315	0	0.0034965035
	ms-wbt-server+1	0.1387249	470	0.9468085106	0	0.0021276596
	microsoft-ds+1	0.012987	44	0.4318181818	0	0
	mysql+3	0.0005903	2	0.5	0	0
	ms-wbt-server+4	0.0002952	1	1	0	0
WGAN-GPMI	ms-wbt-server+5	0.2063164	699	0.9141630901	0.0014306152	0.0200286123
	kerberos+6	0	0	0	0	0
	cpdlc+2	0.0779221	264	0.0340909091	0	0.0113636364
	ms-wbt-server+1	0.074085	251	0.9482071713	0	0
	microsoft-ds+1	0.0298111	101	0.0792079208	0	0.0495049505
	mysql+3	0.0572609	194	0.0257731959	0.0051546392	0.0103092784
	ms-wbt-server+4	0.0103306	35	0.8	0	0

Figure 5.11: Comparison of Conditional Probability Tables

Model. WGAN-GPMI does a far better job of recreating this input, as it occurs 194 times in the generated distribution.

Another interesting observation from these probability tables is that sometimes only a subset of the input conditioning values affects the output value. For example, if the given Destination Service value is *ms-wbt-server* then the output Alert Signature is *ET DOS Microsoft RDP Syn.* for *all* Time Bin values except 5.

Output Modes Captured

Finally, to assess output modes captured by each model, two values were collected; the number of values that existed in the ground truth that were not generated by the model and the number of values generated by the model that never occurred in the ground truth. Like all previous metrics, these values were collected for each unique m-tuple combination of features.

Table 5.11 shows the number of output modes missed by the generative model. The bottom two rows identify the number of unique output modes consisting of all 4 features for each target IP and the percentage of modes dropped. Note that this table shows the direct benefit of mutual information maximization, as the number of output modes missed by the model decreases across the board for the WGAN-GPMI model. Some of the target IP addresses recover more modes than others when moving to the WGAN-GPMI model; 10.0.0.100, as well as 10.0.0.22, halve the number of output modes dropped. On the other hand, 10.0.0.27 and 10.0.99.143 only see a minor improvement when adding in the mutual information constraint.

One potential explanation for this would be that the missing output modes occur with such low probability that even with the mutual information constraint on the generator the model does not receive sufficient gradient feedback to learn to output these values. Methods to recreate rare samples remains an ongoing challenge that is incredibly important to the field of Cyber Security. It is possible that critical actions, such as data exfiltration

may only generate a small number of alerts rarely over the course of an attack. Being able to model such behaviors would be extremely beneficial to proactive Cyber Defense, where critical vulnerabilities are identified and patched. Additionally a means to identify the type of behavior associated with the additional output modes captured would provide contextual information to what type of network behaviors are most recoverable from data driven models such as GANs.

Table 5.12 shows the number of noisy outputs by the generative model that did not occur in the ground truth. The bottom two rows of this table shows the number of unique modes and the ratio of noisy 4-tuple samples over the number of true combinations. For target IP 10.0.99.143, the target IP tested with the lowest number of alerts, the WGAN-GPMI model appears to create less noisy outputs. Additionally 10.0.0.27 also sees a significant drop in the number of noisy outputs generated when using the WGAN-GPMI model. Interestingly, these are also the two IP addresses which did not experience a large decrease in the number of output modes dropped.

Interestingly, these output modes aren't inherently wrong since the individual feature value do exist in the ground truth dataset. However, there should be no gradient feedback to encourage the generation of these combinations of feature values since they don't occur in the ground truth data.

Generality of Models

One of the most powerful characteristics of Machine Learning models is their ability to adapt well to new datasets from the same domain. To ensure that the models presented here maintain generality we retrain each on the CPTC'18 dataset. This dataset continues to make use of Suricata alerts, however includes different feature values and is based off an attack on a completely new network architecture. Additionally, the number of alerts for CPTC'18 is far greater, with tested target's containing between 7475 and 9850 alerts. The same features and preprocessing steps identified in Section 4.1 were applied to alerts from

Table 5.11: Output Modes Dropped

	Target IP Address (WGAN-GP)				Target IP Address (WGAN-GPMI)				
M	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**	
1	5	4	2	1	1	4	2	1	
	1	0	0	1	2	6	1	3	
	0	1	0	1	0	1	0	0	
	10	5	3	3	1	0	0	1	
2	17	11	4	4	2	7	1	2	
	6	2	2	3	2	6	0	2	
	11	9	3	2	4	10	2	5	
	13	9	3	7	1	2	1	2	
	7	6	2	4	6	11	2	4	
	10	7	3	3	3	8	2	3	
3	14	11	3	7	9	13	3	8	
	20	13	8	9	5	10	3	7	
	18	13	5	4	5	11	2	5	
	16	12	7	9	6	12	3	4	
4	21	15	9	10	10	14	4	8	
# Alerts	3388	3166	2974	2182					
# Unique Modes	32	27	22	27					
% Modes Dropped	0.6563	0.5556	0.4091	0.3704	0.3125	0.5185	0.1818	0.2963	

Table 5.12: Noisy Generator Output Counts

	Target IP Address (WGAN-GP)				Target IP Address (WGAN-GPMI)			
M	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**
1	0	0	0	5	0	0	0	0
	2	0	0	24	0	0	0	0
	11	0	0	17	0	0	0	0
	0	0	0	30	0	0	0	1
2	70	23	29	95	31	23	13	20
	22	0	0	60	11	12	12	4
	12	1	1	68	39	24	12	21
	10	9	3	71	20	14	11	8
	20	5	1	94	14	16	26	3
	19	2	1	58	23	20	22	10
3	131	112	46	192	103	60	63	50
	119	97	36	240	83	40	34	40
	49	23	6	187	117	57	52	57
	54	46	17	174	84	57	68	40
4	168	235	76	321	213	97	107	98
# Alerts	3388	3166	2974	2182				
# Combinations	286200	185976	128520	189658				
# Unique Modes	32	27	22	27				
Noise Ratio	5.250	8.704	3.455	11.889	6.656	3.593	4.864	3.630

the four IP addresses that contained the most number of alerts. These IP addresses included 10.0.1.46, 10.0.1.5, 10.0.0.24, and 10.0.0.22.

Table 5.13: Histogram Intersection for all Feature Combinations: CPTC'18

	Victim Machine IP Address							
	WGAN-GP				WGAN-GPMI			
Features	**10.0.1.46**	**10.0.1.5**	**10.0.0.24**	**10.0.0.22**	**10.0.1.46**	**10.0.1.5**	**10.0.0.24**	**10.0.0.22**
A	0.752	**0.810**	0.801	0.815	**0.852**	0.765	**0.825**	**0.863**
D	0.764	**0.954**	0.800	0.820	**0.859**	0.909	**0.918**	**0.874**
S	0.744	0.740	0.789	0.812	**0.844**	**0.785**	**0.872**	**0.867**
T	0.782	0.718	0.851	0.811	**0.826**	**0.766**	**0.928**	**0.857**
A,T	0.764	**0.954**	0.800	**0.820**	**0.859**	0.909	**0.918**	0.874
A,S	0.744	0.740	0.789	0.812	**0.844**	**0.785**	**0.872**	**0.867**
S,D	0.782	0.718	0.851	0.811	**0.826**	**0.766**	**0.928**	**0.857**
D,T	0.679	0.701	0.784	0.793	**0.746**	**0.744**	**0.898**	**0.811**
S,T	0.667	0.734	0.766	0.789	**0.753**	**0.773**	**0.848**	**0.816**
A,D	0.646	0.644	0.774	0.796	**0.694**	**0.646**	**0.862**	**0.807**
A,S,T	0.679	0.701	0.784	0.793	**0.746**	**0.744**	**0.898**	**0.811**
A,S,D	0.667	0.734	0.766	0.789	**0.753**	**0.773**	**0.848**	**0.816**
A,D,T	0.646	0.644	0.774	0.796	**0.694**	**0.646**	**0.862**	**0.807**
S,D,T	0.536	0.617	0.750	**0.769**	**0.580**	0.616	**0.820**	0.762
A,S,D,T	0.536	**0.617**	0.750	**0.769**	**0.580**	0.616	**0.820**	0.762

After training each model the Histogram Intersection was computed for each generated distribution, as shown in Table 5.13. Given the results, both models perform well at

generating alerts which emulate the CPTC'18 data. Note that the prior observations regarding mutual information maximization increasing Histogram Intersection also hold true when applied to a new dataset. Additionally, the methods presented for identifying feature dependencies and output mode dropping are fully applicable to this new dataset.

Chapter 6

Conclusions and Future Work

Conclusion

This work shows the potential for using Generative Adversarial Networks as a means to generate artificial network intrusion detection alerts. Given the stochastic nature of alerts, rarity of known malicious samples, and rapidly growing need for alert data to train Machine Learning algorithms, GANs provide a means to create alerts that traditional statistic models cannot compare to.

Additionally, two intuitive metrics are defined for analyzing alert generation fidelity; Histogram Intersection and Jensen Shannon Divergence. Direct visualization of the Histogram Intersection illustrates the model's output distribution across m-tuple combinations of features, showing the intricacy of learning to output realistic alert data. Jensen Shannon Divergence is employed to confirm relationships identified by Histogram Intersection and highlight crucial differences in the probability distribution of the generative model.

Using these metrics and information theoretic calculations, feature dependencies are revealed and examined to provide a deeper understanding of both the power and shortcomings of GANs applied to NIDS alert generation. Joint entropy is computed for the ground truth distribution and used as a baseline score identifying the challenge of modeling the data distribution. Then conditional entropy is employed on both the ground truth and generated data distributions to identify feature dependencies and how well the generative network models these dependencies. In order to view low level feature-value dependencies, condi-

tional probability tables are generated and highlighted to show sparsity and determinism.

Finally, it is shown that using mutual information is an effective means of regularizing the generator network, improving exploration of the ground truth feature space. Not only do the metrics measuring alert fidelity improve, but output mode collapse is shown to decrease. This is a direct result of the generator more closely modeling the ground truth data distribution.

Future Work

Though this work shows the promise of GANs in NIDS alert generation the results are far from perfect. Modeling the distribution of alerts is still a monumental challenge for several reasons: The lack of known malicious alert data makes it challenging to train GANs. Current Machine Learning methods in the field of Cyber Security use sequences of alerts in order to model temporal relationships in an attack sequences. And the quality of results requires improvement before being ready for application to other Cyber Security Machine Learning systems, especially with regards to guiding alert generation to target specific types of attacker behavior.

Multi-Alert Generation and Analysis

Altering the models presented by this work to generating multiple temporally related alerts at once would greatly increase the utility of generator outputs. Current works applying Machine Learning methods to Cyber Security problems almost exclusively use Recurrent Neural Network structures for attack classification and prediction. It is well established that cyber attacks have intricate behavioral patterns consisting of multiple actions; several alerts may be all related to a single goal or exploit used by the attacker. Modifying the models presented to output chains of alerts would allow for inspection of attacker behaviors and long term strategies.

Two means of modifying these models are presented. First, each of the neural networks

presented could be modified to use a Recurrent Neural Network structure. Particularly, Long Short Term Memory cells have been effective at learning complex temporal relationships and can adaptively select when to forget prior information. In order to train a model consisting of LSTM Neural Networks, the ground truth data would have to be segmented into contiguous chains of alerts. Additionally, alert chain generation would have be cut manually, as there is no natural concept of what type of alert marks the end of a chain (whereas in other tasks like sentence generation a period represents the end).

The second modification to network structure would be to implement Convolutional Neural Networks for each model. In order to make this network structure work, packets would have to be structured into 2D arrays consisting of contiguous alerts. One example of this would be for each packet to have the prior 2 packets preprended to the alert and the subsequent 2 packets appended. This would create an $2N + 1$ x M array where $2N + 1$ is the number of alerts considered in each input to the network and M is the number of features given by each alert. This network would also have the drawback of being a fixed size input and output.

Regardless of the means used to generate chains of alerts, new analysis options are immediately available when considering alert sequences. Sequences of alerts could be modeled as Markov Chains allowing for longest common subsequence to be identified and direct comparison of state transition probabilities. This allows for a rich understanding of the temporal performance of the GAN and how it's results differ from ground truth sequences. Additionally alert sequences could be viewed as a graph allowing for visualization of alerts with low connectivity/deterministic attack behavior, and feature value changes over time.

Improving Generation through Reinforcement Learning

Methods of establishing generator loss in a generative adversarial networks fail to capture discrete features which have interdependencies. This avenue of future work proposes a way to model generator loss as a game and apply a reinforcement learning inspired solution to

it.

Consider the generator to be an agent, whose action space is comprised of all the potential output features it may generate. Then, consider the following hierarchical reward scheme: For each feature that is output correctly reward the generator with 2^1 'points'. For each pair of features reward the generator with 2^2 'points'. And so on and so forth where the maximal reward is 2^n 'points'. The n-many tuple that is correct can be considered the order of the output (e.g.) The above examples would be first order, second order, and finally n-th order. This can be established as a sum of the correct combinations as follows: Given A is the action space consisting of all unique features in the dataset and s is a sample action generated by the neural network drawn from A.

$$L = \sum_{k=1}^{n} \sum_{s_{\binom{n}{k}} \in A_{\binom{n}{k}}} 2^k \tag{6.1}$$

This formulation has the following benefits:

1. Encourages a generator which captures inter-feature dependencies during training
2. Can be trained without the discriminator network, lowering the potential intractability of complex networks
    - Note that this comes at the expense of a loss function with O(n!) runtime, arguably intractible for actions with a high rank
3. Can be modified to target only a specific combination of features
4. Increases reward size non-linearly to as to account for the significant difficulty increase in getting n-many tuples correct.

Another benefit of formulating loss in this manner is that it allows for easy expandability. Two examples are as follows:

1. The loss may be modified to provide a reward proportional to the probability of the selected character being generated.

2. The loss may be modeled as a n-th order Markov Source to capture temporal structure within the data.

In order to account for the distribution of alert feature combinations in the loss function, each feature predicted that is in the Alert Space would have a weighted reward value by the probability of that feature value occurring as shown below: Let F be a subset of features from the Alert Space A and F_v represents a specific value for the feature (f). Then weight the loss of the generated sample by the probability p_{v_n} of that specific feature value combination occurring.

$$p_{v_n} = \frac{|F_{v_n}|}{|F_n|} \tag{6.2}$$

$$L = \sum_{k=1}^{n} \sum_{s_{\binom{n}{k}} \in A_{\binom{n}{k}}} (2^k) p(v_1) \tag{6.3}$$

This can be further expanded to account for the impact of other features on the target prediction feature. Conceptually, if trying to compute the loss of feature A given feature B as a prior, the conditional probability of A may be computed. This would further encourage inter-feature dependency as the loss function would account for decreases in the entropy of one feature value given another.

An example of this would be computing the probability of a specific value for Alert Signature given a specific Destination Service value. The number of possible alert signatures is greatly reduced based on the vulnerabilities in the targeted Destination Service. On the other hand, it is still important to value results which occur in the global action space even if they don't occur with the specific IP value given; the intuition behind this is that the generated should also be encouraged to explore the action space, not just exploit it. To do this the joint probability should be considered equally with the naive probability proposed above.

$$L = \sum_{k=1}^{n-1} \sum_{s_{\binom{n}{k}} \in A_{\binom{n}{k}}} (2^k) p(v_1) + \sum_{k=1}^{n-1} \sum_{s_{\binom{n}{k}} \in A_{\binom{n}{k}}} (2^k) p(v_1|v_2) \tag{6.4}$$

Such a conditional probability model can also be used to model temporal relations within the data. This is accomplished by using the previous alerts feature value as a prior for the current generated feature. This in turn can be extended to an n-th order markov model where n previous alerts act as priors for the current alert being generated. This may be represented as shown below:

$$L = \sum_{k=1}^{n-1} \sum_{s_{\binom{n}{k}} \in A_{\binom{n}{k}}} (2^k) p(v_1|v_2, v_3, ..., v_n)) \tag{6.5}$$

Bibliography

[1] Hyrum S. Anderson, Anant Kharkar, Bobby Filar, and Phil Roth. Evading machine learning malware detection. In *Proceedings of Blackhat*, Mandalay Bay, Las Vegas, NV, July 22-27 2017.

[2] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, pages 214–223, Sydney, NSW, Australia, August 6-11 2017.

[3] Ishmael Belghazi, Sai Rajeswar, Aristide Baratin, R. Devon Hjelm, and Aaron C. Courville. MINE: mutual information neural estimation. In *Proceedings of International Conference on Machine Learning*, Stockholmsmssan, Stockholm Sweden, July 10-15 2018.

[4] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, Vol. 56(Num. 15), October 2012.

[5] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *Proceedings of IEEE Symposium on Security and Privacy*, volume abs/1608.04644, San Jose, CA, May 22-24 2017.

[6] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 29, pages 2172–2180. Curran Associates, Inc., 2016.

[7] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-

image translation. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, Utah, USA, June 18-22 2018.

[8] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository - kdd cup dataset. "`http://archive.ics.uci.edu/ml`", 2017.

[9] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proccedings of AAAI Conference on Artificial Intelligence*, New Orleans, Louisiana, USA, February 2-7 2018.

[10] Haitao Du and Shanchieh Jay Yang. Probabilistic inference for obfuscated network attack sequences. In *Proceedings of IEEE/ISIF International Conference on Dependable Systems and Networks*, Atlanta, GA, June 23-26 2014.

[11] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on machine learning models. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, Utah, USA, June 18-22 2018.

[12] Isaac Faber and Giovanni Malloy. Deep security: Cyber security threat behavior classification. "`http://cs230.stanford.edu/projects_spring_2018/reports/8285947.pdf`", 2018.

[13] D. S. Fava, S. R. Byers, and S. J. Yang. Projecting cyberattacks through variable-length markov models. *IEEE Transactions on Information Forensics and Security*, Vol. 3(Num. 3):359–369, September 2008.

[14] Pavel Filonov, Fedor Kitashov, and Andrey Lavrentyev. RNN-based early cyber-attack detection for the tennessee eastman process. In *Proceedings of ICML Time Series Workshop*, Sydney, Australia, August 11 2017.

[15] Pavel Filonov, Andrey Lavrentyev, and Artem Vorontsov. Multivariate industrial time series with cyber-attack simulation: Fault detection using an LSTM-based predictive data model. In *Proceedings of NIPS Time Series Workshop*, Barcelona, Spain, December 9 2016.

[16] Yang Gao, Rita Singh, and Bhiksha Raj. Voice impersonation using generative adversarial networks. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2506–2510, Calgary, Alberta, Canada, April 15-20 2018.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Proceedings of Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[18] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of The IEEE International Conference on Learning Representations ICLR*, San Diego, California, USA, May 7-9 2015.

[19] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Proceedings of Advances in Neural Information Processing Systems 30*, pages 5769–5779, Long Beach, California, USA, December 4-9 2017.

[20] Liang Hu, Zhen Zhang, Huanyu Tang, and Nannan Xie. An improved intrusion detection framework based on artificial neural networks. In *Proceedings of 2015 11th International Conference on Natural Computation (ICNC)*, pages 1115–1120, Zhangjiajie, China, August 15-17 2015.

[21] Weiwei Hu and Ying Tan. Black-box attacks against RNN based malware detection algorithms. In *Proccedings of AAAI Conference on Artificial Intelligence*, New Orleans, Louisiana, USA, February 2-7 2018.

[22] Stefano Iannucci, Hisham Kholidy, Amrita Dhakal Ghimire, Rui Jia, Sherif Abdelwahed, and Ioana Banicescu. A comparison of graph-based synthetic data generators for benchmarking next-generation intrusion detection systems. In *Proceedings of IEEE Cluster Conference*, Hawaii, USA, September 5-8 2017.

[23] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (To Appear)*, volume abs/1812.04948, Long Beach, California, USA, June 15-21 2019.

[24] Alexander D Kent. Cybersecurity Data Sources for Dynamic Network Research. In *Dynamic Networks in Cybersecurity*. Imperial College Press, June 2015.

[25] Jihyun Kim, Jaehyun Kim, Huong Le Thi Thu, and Howon Kim. Long short term memory recurrent neural network classifier for intrusion detection. In *Proceedings of International Conference on Platform Technology and Service*, Jeju, Korea, February 15-17 2016.

[26] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume abs/1609.04802, Honolulu, Hawaii, USA, July 22-25 2017.

[27] Ming Li. Attack graph analysis based on markov decision process, 2017.

[28] Zilong Lin, Yong Shi, and Zhi Xue. IDSGAN: generative adversarial networks for attack generation against intrusion detection. *CoRR*, abs/1809.02077, 2018.

[29] R.P Lippmann, D.J Fried, I. Graf, J.W Haines, K.R Kendall, D McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M.A. Zissman. Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation. In *Proceedings of DARPA Information Survivability Conference and Exposition*, Hilton Head, SC, USA, January 25-27 2000.

[30] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. Palais des Congrs Neptune, Toulon, France, April 24-26 2017.

[31] John McHugh. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information System Security*, 3(4):262–294, November 2000.

[32] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[33] Steven Noel and Sushil Jajodia. Advanced vulnerability analysis and intrusion detection through predictive attack graphs. *Critical Issues in C4I, AFCEA Solutions Series. International Journal of Command and Control*, 2009.

[34] Ian Perry, Lutzu Li, Christopher Sweet, Shanchieh Jay Yang, and Ahmet Okutan. Differentiating and predicting cyberattack behaviors using lstm. In *IEEE Conference on Dependable and Secure Computing*, Kaohsiung, Taiwan, December 10-13 2018.

[35] Xinzhou Qin and Wenke Lee. Attack plan recognition and prediction using causal networks. In *Proceedings of 20th Annual Computer Security Applications Conference*, pages 370–379, Tucson, AZ, December 6-10 2004.

[36] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *Proceedings of The*

IEEE International Conference on Learning Representations ICLR, Caribe Hilton, San Juan, Puerto Rico, May 2-4 2016.

[37] Maria Rigaki and Sebastin Garca. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In *Proceedings of IEEE Security and Privacy Workshops (SPW)*, San Francisco, California, USA, May 24 2018.

[38] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Proceedings of Conference on Neural Information and Processing Systems*, volume abs/1701.00160, pages 2234–2242, Barcelona, Spain, December 2016.

[39] Yun Shen, Enrico Mariconti, Pierre Antoine Vervier, and Gianluca Stringhini. Tiresias: Predicting security events through deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 592–605, New York, NY, USA, 2018. ACM.

[40] Peter Siska, Marc Ph. Stoecklin, Andreas Kind, and Torsten Braun. A flow trace generator using graph-based traffic classification techniques. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, IWCMC '10, pages 457–462, New York, NY, USA, 2010. ACM.

[41] Joel Sommers and Paul Barford. Self-configuring network traffic generation. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 68–81, New York, NY, USA, 2004. ACM.

[42] Ralf C Staudemeyer. Applying long short-term memory recurrent neural networks to intrusion detection. *South African Computer Journal*, 56, 2015.

[43] Salvatore Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip Chan. Cost-based modeling for fraud and intrusion detection: Results from the jam project. In

Proceedings of DARPA Information Survivability Conference, volume 2, pages 130 – 144 vol.2, Hilton Head, SC, USA, January 25-27 2000.

[44] Hui Su, Xiaoyu Shen, Pengwei Hu, Wenjie Li, and Yun Chen. Dialogue generation with gan. In *Proccedings of AAAI Conference on Artificial Intelligence*, New Orleans, Louisiana, USA, February 2-7 2018.

[45] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proceedings of International Conference on Learning Representations*, Banff, Canada, April 14-16 2014.

[46] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, CISDA'09, pages 53–58, Piscataway, NJ, USA, 2009. IEEE Press.

[47] Joe Touch, Eliot Lear, Allison Mankin, Markku Kojo, Kumiko Ono, Martin Stiemerling, Lars Eggert, Alexey Melnikov, Wes Eddy, Alexander Zimmermann, Brian Trammell, Jana Iyengar, Allison Mankin, Michael Tuexen, Eddie Kohler, and Yoshifumi Nishida. Service name and transport protocol port number registry, 2018.

[48] Aaron Tuor, Samuel Kaplan, Brian Hutchinson, Nicole Nichols, and Sean Robinson. Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. In *Proceedings of AI for Cyber Security Workshop at AAAI 2017*, volume abs/1710.00811, San Francisco, California USA, 2017.

[49] Kalyan Veeramachaneni, Ignacio Arnaldo, Vamsi Korrapati, Constantinos Bassias, and Ke Li. AI2 : Training a big data machine to defend. In *Proceedings of IEEE 2nd International Conference on Big Data Security on Cloud*, pages 49–54, Beijing, China, June 2016.

[50] L. Wang, A. Liu, and S. Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications*, 29(15):2917 – 2933, 2006.

[51] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. In *Proceedings of International Joint Conference on Artificial Intelligence*, Stockholmsmssan, Stockholm, Sweden, July 13-19 2018.

[52] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. pages 2242–2251, Venice, Italy, October 22-29 2017.

Appendices

Appendix A

SVM Feature Dependency Experiment

One additional way to confirm the dependencies highlighted through drop in histogram intersection and conditional entropy is to test with a simple model for separation. To do so, a SVM with the RBF Kernel function was trained using all unique permutations of features. The model was given 1 to $n-1$ features as conditioners and an expected output feature for all possible input-output pairs.

To further prove that the dependencies identified by the GAN exist, a SVM was fit to the $1|2$-combination distributions for each of the target machines from the CPTC'17 dataset. The accuracy of this fit was tabulated for all four victim IP addresses tested in Table A.1.

Table A.1: SVM Prediction Accuracy For 3-Combination Feature Values Assorted Victim IPs

	Machine IP Address			
Prediction|Features	**10.0.0.100**	**10.0.0.27**	**10.0.0.22**	**10.0.99.143**
D|A,T	0.958	0.591	0.949	0.908
D|A,S	0.962	0.616	0.970	0.977
D|S,T	0.790	0.541	0.879	0.794
A|D,T	0.911	0.490	0.929	0.472
A|S,D	0.852	0.516	0.889	0.486
A|S,T	0.749	0.440	0.811	0.344
S|A,T	0.719	0.742	0.539	0.702
S|D,T	0.736	0.814	0.525	0.729
S|A,D	0.962	0.616	0.970	0.977
T|A,S	0.411	0.387	0.215	0.459
T|S,D	0.408	0.424	0.161	0.514
T|A,D	0.178	0.208	0.189	0.436

Note that the order of the combinations is the same as that given in Table 5.7 and held

constant for all victim IPs. Though some general trends exist, such as alert signature and destination port category being poor predictors of timestamp, there is variation between the different victims. Fig. A.1 shows that regardless of what feature dependencies exist for a given victim there is a strong negative correlation between accuracy of an SVM predictor and the conditional entropy.

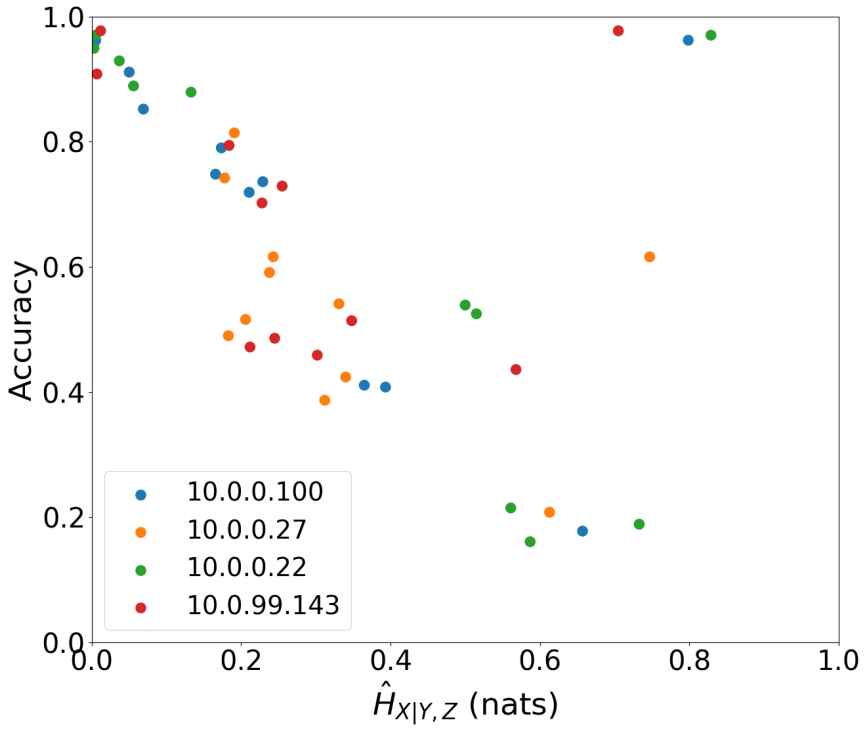


Figure A.1: SVM Accuracy Plotted Against Conditional Entropy

Appendix B

Alert Dependency Plots

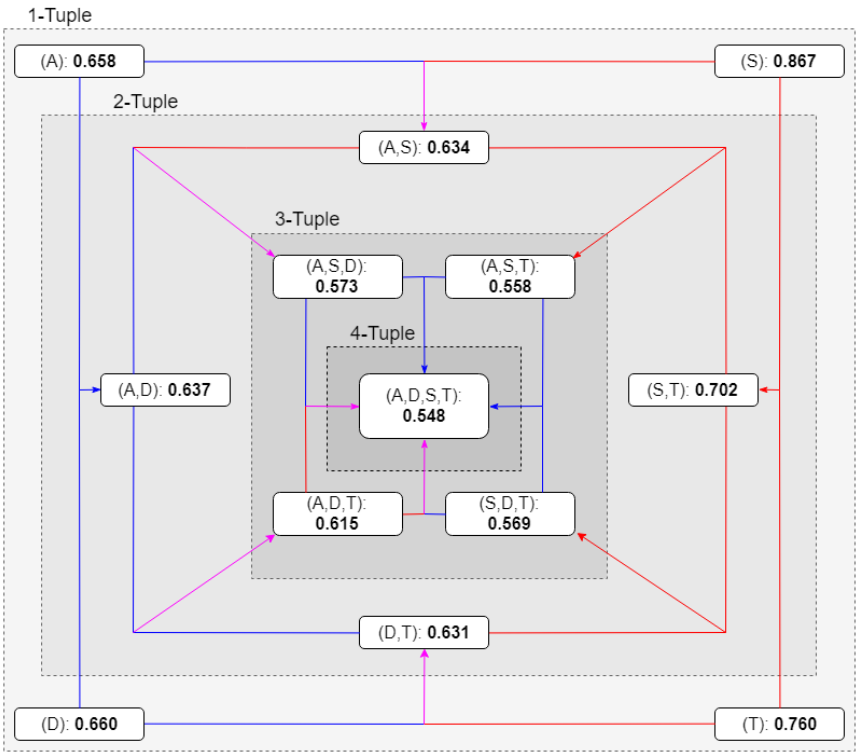


Figure B.1: Target 10.0.0.22 Alert Dependency Graph: WGAN-GP Result

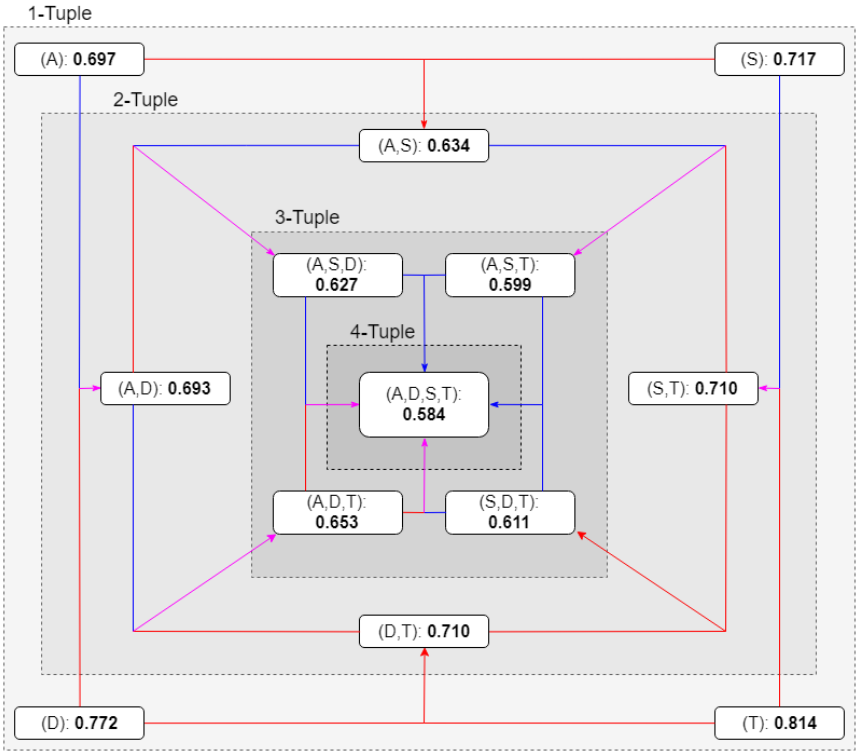


Figure B.2: Target 10.0.0.100 Alert Dependency Graph: WGAN-GP Result

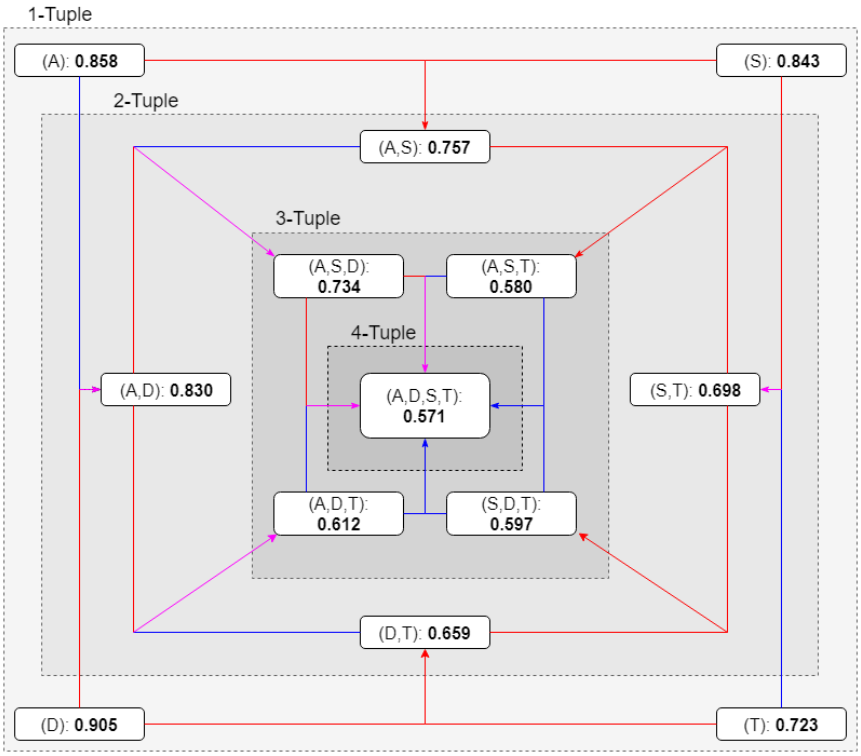


Figure B.3: Target 10.0.99.143 Alert Dependency Graph: WGAN-GP Result

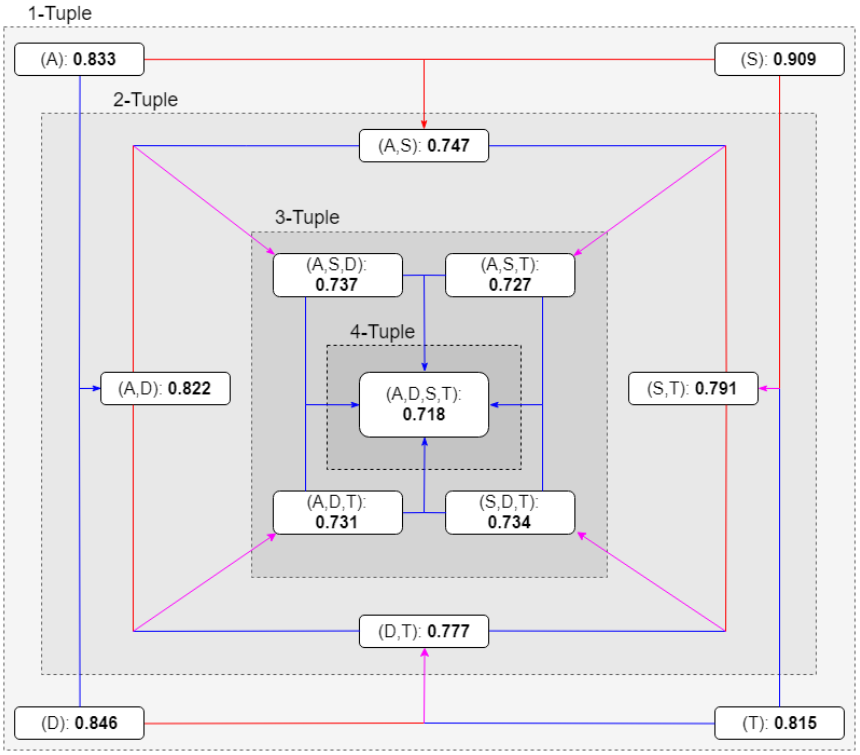


Figure B.4: Target 10.0.0.22 Alert Dependency Graph: WGAN-GPMI Result

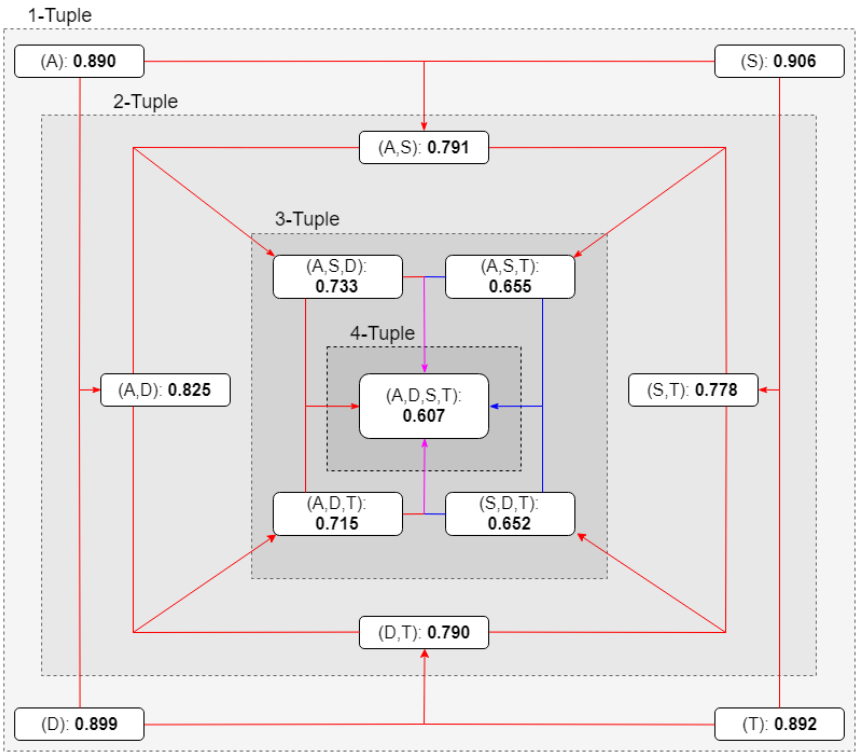


Figure B.5: Target 10.0.0.100 Alert Dependency Graph: WGAN-GPMI Result

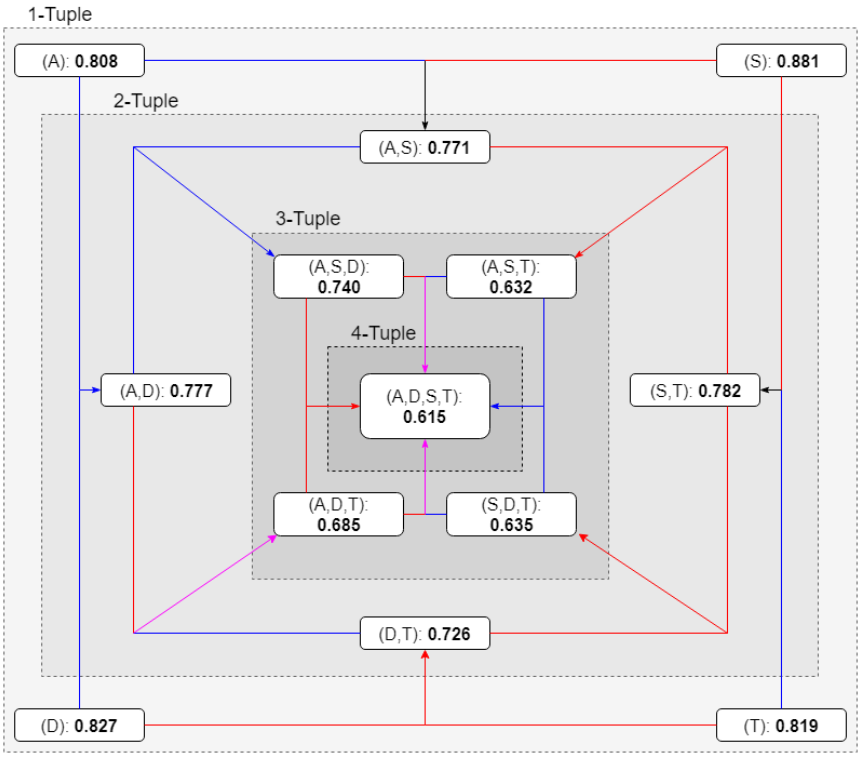


Figure B.6: Target 10.0.99.143 Alert Dependency Graph: WGAN-GPMI Result

Appendix C

Alert Dependency Plots

The Common Port Services listing provided by IANA is available at the following url: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml Note that these mappings are periodically updated and are only common mappings/suggestions. Individual corporations may opt to configure their own service mapping.