Can GAN-Generated Network Traffic be used to Train Traffic Anomaly Classifiers?

Pasquale Zingo
Electrical and Computer Engineering
University of Delaware
Newark, DE
patzingo@udel.edu

Andrew Novocin
Electrical and Computer Engineering
University of Delaware
Newark, DE
andynovo@udel.edu

***Abstract*—Recent attempts to introduce the Generative Adversarial Network (GAN) to the computer network traffic domain have shown promise, including several frameworks which generate realistic traffic. This paper presents the 'GAN vs Real (GvR) score', a task-based metric which quantifies how well a traffic GAN generator informs a classifier compared to the original data. This metric is derived from the 'Train-on-Synthetic, Test-on-Real' (TSTR) method, with the added step of comparing the TSTR accuracy to the performance of the same classifier trained on real data and tested on real data. We use this framework to evaluate the B-WGAN-GP model for generating NetFlow traffic records using several stock classifiers. Using GvR we conclude that it is possible to train accurate traffic anomaly detectors with GAN-generated network traffic data.**

***Index Terms*—GAN, IDS, TSTR, NetFlow, Augmentation**

I. Introduction

Labeled computer network traffic plays a crucial role in informing anomaly-based intrusion detection systems (IDS). These systems are a significant component of the cyber infrastructure and provide advantages beyond the signature-based IDS paradigm which is currently dominant. However, the lack of labeled, high-quality data means that anomaly-based IDS underperform signature-based ones. Existing data sets are insufficient for myriad reasons, among them: antiquated data, samples are generated probabilistically, samples are unlabeled (malicious vs normal), or censored for security of the source institution [1]. In the last case, the data sets collected from real world environments suffer due to the need to protect the privacy of the institution and users that generated it. Computer network traffic generation tools exist, but the challenge of determining if the synthetic traffic will inform a model remains.

Generative Adversarial Network (GAN) is a generative deep learning techniques for approximating and sampling the probability distribution which describes a data set. GAN has been successfully implemented in several domains since their introduction in 2014 [2] when it was an early success in deep neural network image generation.

The contributions of this work are:

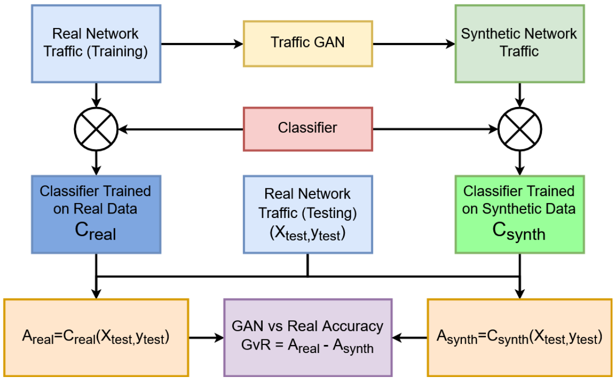
Real Network Traffic (Training)
Traffic GAN
Synthetic Network Traffic
Classifier
Classifier Trained on Real Data
C_{real}
Real Network Traffic (Testing)
(X_{test}, y_{test})
Classifier Trained on Synthetic Data
C_{synth}
$A_{real}=C_{real}(X_{test}, y_{test})$
GAN vs Real Accuracy
$GvR = A_{real} - A_{synth}$
$A_{synth}=C_{synth}(X_{test}, y_{test})$

Fig. 1. How GvR Score is calculated.

- **First generation of labeled traffic records with GAN**: We adapt the B-WGAN-GP to generate NetFlow traffic records which are labeled as malicious or normal. This is a necessary step to using GAN-generated data to train IDS models.
- **Traffic Classifiers can trained from GAN-generated data**: We train classifiers on synthetic traffic which perform within .1% accuracy of the same classifier trained on real data. Another paradigm, GAN-as-augmentation, is explored. Here classifiers are trained on both real data and the output of a GAN trained on the real data, resulting in superior accuracy.
- **A framework for measuring the quality of traffic-GANs**: The application of TSTR and GAN vs Real (GvR) accuracy provides an opportunity to directly compare different frameworks for generating network traffic in terms of task based performance. We hope this will spur new research and lead to competing models for improved traffic GANs.

The last two years have seen efforts to apply GANs to the network traffic domain, but computer traffic poses difficulties for the GAN architecture. GANs function on continuous domains where small changes in a sample are insignificant, but this is not the case for network traffic. Depending on the feature representation of the traffic, the almost all the features are binary (in the case of raw packet captures) or categorical

This publication (or program) was made possible by the National Science Foundation EPSCoR Grant No. 1757353 and the State of Delaware.

978-1-7281-8416-6/20/$31.00 ©2020 IEEE

(as is the case for NetFlow representations). In either case, the smallest possible variation in a feature can invalidate the traffic record.

The design of models to generate traffic has been troubled by the problem of evaluation. Traffic GAN evaluation has, to this point, focused on evaluating relative to domain-knowledge coherence of the generated traffic. Questions like "is the synthetic traffic free of artifacts that could not appear in true generated traffic" and "can the generated traffic propagate properly over the network the real data was collected in" are common for evaluation. We believe that this is insufficient, since it's not clear that traffic which can perform well according to the above questions will necessarily do an effective job in communicating the behaviors relevant to detecting malicious traffic, or even whether behaving like traffic that could originate in a real computer network is relevant when determining malicious patterns.

To solve the problem of evaluation, we adapt the Train on Synthetic, Test on Real (TSTR) method [3], which was originally developed to evaluate the performance of continuous time-series GANs for medical data. TSTR is a natural metric for evaluating traffic GANs with respect to their usefulness for training anomaly IDS, as it aligns the task the GAN is evaluated on to the task the GAN will perform once trained. Comparing the TSTR accuracy to that of the same classifier trained on the data used to train the GAN, we obtain a metric which captures the informativeness of the GAN for the classification task, which we have called 'GAN vs Real (GvR) score'.

Section II examines the traffic GANs in other works, other works involving network traffic augmentation, and the TSTR metric. Section III explains the methodology for our experiments, with the results and analysis in section IV. Section V suggests future work in this domain.

II. Literature Survey

A. Traffic GANs

To account for this lack of real-world data sets, several attempts to synthesize network traffic using GANs have been made. These methods have gone some way to recreate traffic that is realistic, but to this date the evaluation of their output has not measured their applicability to training anomaly-based IDS. Some examples of architectures are listed below:

1) B-WGAN-GP & E-WGAN-GP: B-WGAN-GP and E-WGAN-GP are models proposed by Ring et. al to generate NetFlow traffic. B-WGAN-GP was trained on a binary representation on the NetFlow data from week 1 of the CIDDS-001 data set [4], and E-WGAN-GP works on an embedding of the same data using IP2Vec [5] for the categorical features. Both are based on the Wasserstein GAN with Gradient Penalty [6] and use the Two Time-scale Update Rule (TTUR) [7]. Other than the data representation, the models are designed identically and trained for 5 epochs.

Evaluation The authors cite the difficulty of evaluating generated traffic, and used two strategies for measuring the similarity between weeks 2-4 of the original data set, a benchmark, and the output of their GANs. The benchmark used the empirical distribution of the data and sampled each feature independently.

- Euclidean Distances: the Euclidean distance between single features of the empirical distribution of the of weeks 2-4 and the synthetic data sets (baseline and *-WGAN-GP's).
- Domain Knowledge Checks: behaviors expected of network traffic flows were checked for consistency.
  1) No TCP flags for UDP packets
  2) At least one of Source IP and Dest IP must be internal
  3) Normal user behavior using ports 80 and 443 must use TCP
  4) Normal user behavior using ports 53 must use UDP
  5) Only destination IP can be broadcasts
  6) Netbios flows the source must be internal and the dest IP must be external
  7) TCP, UDP, ICMP must not exceed maximum packet size

2) PAC-GAN: PAC-GAN [8] is a model which generates packet-level traffic using a Convolutional Neural Network (CNN) GAN. Packet data was encoded as 28×28 binary matrices, leveraging the raster data structure used in CNN models, which are normally used in the image domain. To account for assumptions of continuity of values and the insignificance of in the image domain that hindered performance for generating packets, the values encoded were duplicated to 'create clusters of similar converted packet byte values.'

Evaluation PAC-GAN was evaluated by attempting to send the decoded packets generated by the model and measuring the Success Rate and Byte Error of the packets.

B. Train on Synthetic, Test on Real

TSTR was introduced by Hyland et. Al [3] to evaluate the performance of GANs generating time series data. The algorithm works by training classifier models on data produced by the GAN and evaluating based on the performance of the classifier on the real data.

Algorithm 1: TSTR: Train on Synthetic, Test on Real (modified for unconditional GAN)

```
train, test = split(data);
discriminator, generator = train_GAN(train);
synthetic_features, synthetic_labels =
  generator.generate_synthetic();
classifier = train_classifier(synthetic_features,
  synthetic_labels);
predictions = classifier.predict(test_features);
TSTR_score = score(predictions, labels)
```

This has several nice features:

- The TSTR value after training a model can inform a hyperparameter search and comparisons between traffic GAN architectures.

- Comparing the synthetically trained classifier to a classifier trained on the original data set gives some indication as to the success the GAN generator had in generalizing the data set. If the GAN-trained classifier significantly under performs the real-trained classifier, then we can conclude the traffic GAN has not generalized well. If the classifiers perform similarly, or if the GAN-trained classifier performs better, we conclude that the GAN is generalizing well.

III. Methodology

Our experiments use the B-WGAN-GP model from [9], a variant of WGAN-GP trained for generating network traffic. All hyperparameters were inherited from [9], and the model was implemented in the torchGAN framework [10]. Our experiment also used NetFlow [11] records from the Coburg Intrusion Detection Data Set (CIDDS) [4] following the binary preprocessing scheme from [9]. See details of the preprocessing in table VII.

From week 1 of the CIDDS data set 5 training sets of 150,000 randomly selected samples were created. Each training set was balanced by class, and each with corresponding, disjoint, balanced testing sets of 300,000 samples. A GAN was trained on each of the training sets, and at certain points in the training (1,5,10, 25, 50, 75, and 100 epochs) the generator was sampled with size 150,000. For each of the generated samples and each of the real training sets, TSTR classifiers were trained, with accuracy calculated on the testing set corresponding to the real training set which they are derived from. For each of the testing points for the GANs, 15 samples, and so 15 classifiers are created, the average accuracy of which is presented in the tables below (I - VI).

Four classifiers were chosen for use in the TSTR regime: Decision Tree, Gaussian Naive Bayes, Adaboost and Random Forest. These were chosen from [12] which demonstrated their applicability for IDS and for reproducibility. All classifiers are used from the scikit-learn python library [13] with default parameters.

Two classifiers are trained and scored for each hyper parameterization: in the first, the classifier is trained only on the synthetic data sampled from the GAN. In the second, a training set of the synthetic data and the training set used to train the GAN it was generated from are combined into a composite training set. The first case corresponds to replacing the traffic data set with a GAN, where the second tests the utility of using the GAN as a data augmentation method.

IV. Results and Discussion

Table I contains average performance of TSTR classifiers trained on the output of B-WGAN-GP. The results corresponding exactly to the hyperparameters prescribed by [9] are in the row for 5 epochs. The hyperparameters of B-WGAN-GP were originally picked using less sophisticated metrics (average values and domain knowledge checks), and we recorded performance at several points in training up to 100 epochs to find if the previous hyperparameters were adequate. We found that classifiers performed better when trained on synthetic data generated from longer trainings of the GAN, however only the case using Random Forest classifier performed better after 75 epochs. Further, none of the classifiers outperform the baseline case trained on the original data set.

TABLE I
Synthetic Scores - Batch Size 64

Epoch	Gaussian NB	Decision Tree	AdaBoost	RF
1	0.536608	0.533106	0.569171	0.552432
5	0.716322	0.744971	0.831313	0.841452
10	0.916622	0.901186	0.978351	0.983134
25	0.965019	0.950287	**0.993959**	0.996504
50	0.972796	0.961122	0.993785	0.997674
75	**0.982439**	**0.961640**	0.991756	0.997558
100	0.977779	0.953080	0.990084	**0.998027**
Real	0.987079	0.999385	0.999537	0.999859

TABLE II
Synthetic Scores - Batch Size 16

Epoch	Gaussian NB	Decision Tree	AdaBoost	RF
1	0.805310	0.739964	0.849059	0.860568
5	0.903028	0.886949	0.977499	0.988512
10	0.966408	0.918425	0.989231	0.995717
25	0.981385	0.942773	0.990096	0.996806
50	0.971112	0.935907	0.989436	0.997822
75	0.976903	**0.946419**	**0.989662**	0.998068
100	**0.981731**	0.937509	0.988838	**0.998104**
Real	0.987079	0.999385	0.999537	0.999859

TABLE III
Synthetic Scores - Batch Size 512, ncritic 1

Epoch	Gaussian NB	Decision Tree	AdaBoost	RF
1	0.493889	**0.544396**	0.494187	0.509598
5	0.480510	0.542926	0.535742	0.534275
10	0.491778	0.485282	0.544150	**0.545224**
25	0.500263	0.495227	**0.551541**	0.498054
50	0.500000	0.506098	0.489463	0.496035
75	**0.505158**	0.507686	0.487027	0.477967
100	0.500906	0.504745	0.506213	0.499927
Real	0.987079	0.999385	0.999537	0.999859

TABLE IV
Synthetic Scores - Batch Size 512, ncritic 10

Epoch	Gaussian NB	Decision Tree	AdaBoost	RF
1	0.441684	0.517180	0.447907	0.470960
5	0.495769	0.465338	0.482221	0.507133
10	0.560044	0.535549	0.612845	0.587122
25	0.649308	0.576659	0.636749	0.632172
50	0.748698	0.616770	0.776213	0.791516
75	0.854834	0.794225	0.853399	0.860014
100	**0.888097**	**0.886342**	**0.956066**	**0.958225**
Real	0.987079	0.999385	0.999537	0.999859

The authors of [9] chose 64 for the training batch size, which is minuscule compared to the memory available on modern GPUs. To explore the significance of batch size for

TABLE V
COMPOSITE SCORES - BATCH SIZE 64

Epoch	Gaussian NB	Decision Tree	AdaBoost	RF
1	0.699359	**0.999283**	0.983256	0.999840
5	0.978069	0.999196	0.996960	0.999809
10	0.985415	0.999031	0.996171	0.999850
25	0.982564	0.999042	0.997483	0.999855
50	0.981506	0.999084	**0.997776**	**0.999872**
75	0.983506	0.999093	0.997506	0.999872
100	**0.983788**	0.999217	0.997244	0.999872
Real	0.987079	0.999385	0.999537	0.999859

TABLE VI
COMPOSITE SCORES - BATCH SIZE 16

Epoch	Gaussian NB	Decision Tree	AdaBoost	RF
1	0.985411	0.999213	0.997232	0.999800
5	0.982340	0.999084	0.996446	0.999880
10	**0.983972**	0.999010	0.996578	0.999880
25	0.982949	0.998986	0.996875	0.999886
50	0.981905	0.999020	0.997084	**0.999894**
75	0.983053	**0.999081**	**0.997252**	0.999891
100	0.983200	0.999039	0.997071	0.999891
Real	0.987079	0.999385	0.999537	0.999859

generating synthetic network traffic, additional runs for batch sizes of 16 and 512 samples were computed. In table III, which is trained at batch size 16, note that the models achieve near-baseline performance by 25 epochs, and then do not consistently improve across the classifiers.

In table III, it is clear that the GAN fails to capture any patterns between the features and label of the traffic, and all classifiers perform as random on average (testing sets are balances, so a 50% accuracy implies no knowledge of the task). We noted that the loss of the discriminator was several orders of magnitude greater than that of the generator when training at this batch size, and an additional test which over trained the discriminator (in the torchGAN architecture, set ncritic hyperparameter to 10 vs 1 in table III), the results of which are in table IV. Here, the performance is not as good as in I or II, but it does give hope that B-WGAN-GP could be trained successfully at a higher batch size. Training the GAN at batch size 16 over 150,000 samples for 100 epochs took over 4 hours on an Nvidia Tesla K80 GPU, vs approximately 10 minutes at 512 samples per batch. Tables V and VI use the same synthetic traffic generated for I and II, respectively, concatenated with the real traffic used to train the GAN which generated it. Here we see that when the GAN is not sufficiently trained the performance of the the classifier is hindered by the synthetic traffic. Furthermore, for the default parameterization of the classifiers as was used, only Random Forest classifier outperforms the baseline model.

Given the results above, we address the adequacy of GAN-generated network traffic for training traffic anomaly detectors.

A. GAN as Traffic Data Set Replacement

The results of tables V-IV indicate that GANs can effectively capture and replicate the patterns between features of a NetFlow record and its class. For the original parameters given by Ring in [9], the Random Forest GvR score is -15.84% (TSTR accuracy - real accuracy), but at 100 epochs the GvR achieves -0.17% accuracy. A GvR of 0 would imply a reconstruction of the original data with all patterns relevant to the task captured as well as the original data set.

We considered that this performance could be explained if GAN had memorized samples from its training set (and if the performance seen at lower batch size/later epochs indicated overfitting and memorization), but after comparing the synthetic data sets to the data sets GANs were trained on, we found that while these values did correlate, even at 100 epochs in the worst case only around 1% of the samples generated were exact copies of training samples.

Algorithm 2: GvR: GAN vs Real score

```
train, test = split(data);
discriminator, generator = train_GAN(train);
synthetic_features, synthetic_labels =
  generator.generate_synthetic();
synth_classifier = train_classifier(synthetic_features,
  synthetic_labels);
synth_score = synth_classifier.score(test_features,
  test_labels);
real_classifier = train_classifier(train_features,
  train_labels);
real_score = synth_classifier.score(test_features,
  test_labels);
TSTR_score = synth_score - real_score
```

B. GAN as data augmentation

The results of tables V and VI show that for only one case, Random Forest classifier with batch size 16, a small but positive GvR score of 0.002% was achieved, though it is too small of a performance improvement over the baseline to present 'GAN as data augmentation' as a viable technique.

V. FUTURE WORK

- **Differential Privacy** [14] is a better regime for comparing if the output of a traffic GAN betrays details about the original dataset, and this analysis must be conducted before this technique can safely be applied to vulnerable data sets.
- **Anonymizing GAN Architectures** such as AnomiGAN [15] use strategies for preserving differential privacy in GAN generation, which could be implemented into a traffic GAN architecture to protect the original data set.
- **Stronger IDS surrogate:** In this work, stock classifiers were used as a baseline calculation of GvR in place of state-of-the-art anomaly IDS algorithms. The GvR metric may be more informative if domain specific classifiers are used.
- **Discriminator Loss Explosion:** In our experiments, discriminator loss frequently exceeded generator loss by

TABLE VII
NETFLOW VARIABLES AND BINARY ENCODINGS

Attribute	**Type**	**Example**	**Binary Encoding**	
			Attr.	Value
Date first seen	timestamp	2020-07-04 17:28:21.123	isMonday	0
			isTuesday	0
			isWednesday	0
			isThursday	0
			isFriday	0
			isSaturday	1
			isSunday	0
			daytime	$\frac{62901}{86400} = 0.728$
Duration	continuous	1.503	norm_dur	$\frac{1.503 - dur_{min}}{dur_{max} - dur_{min}}$
Transport Protocol	categorical	TCP	isTCP	1
			isUDP	0
			isICMP	0
Source IP Address	categorical	192.168.210.5	ip_1 to ip_8	1,1,0,0,0,0,0,0
			ip_9 to ip_16	1,0,1,0,1,0,0,0
			ip_17 to ip_24	1,1,0,1,0,0,1,0
			ip_25 to ip_32	0,0,0,0,0,1,0,1
Source Port	categorical	445	pt_1 to pt_8	0,0,0,0,0,0,0,1
			pt_9 to pt_16	1,0,1,1,1,1,0,1
Destination IP Address	categorical	192.168.220.16	ip_1 to ip_8	1,1,0,0,0,0,0,0
			ip_9 to ip_16	1,0,1,0,1,0,0,0
			ip_17 to ip_24	1,1,0,1,1,1,0,0
			ip_25 to ip_32	0,0,0,1,0,0,0,0
Destination Port	categorical	445	pt_1 to pt_8	1,1,1,0,0,1,0,1
			pt_9 to pt_16	1,1,0,1,1,1,0,0
Bytes	numeric	144	byt_1 to byt_8	0,0,0,0,0,0,0,0
			byt_9 to byt_16	0,0,0,0,0,0,0,0
			byt_17 to byt_24	0,0,0,0,0,0,0,0
			byt_25 to byt_32	1,0,0,1,0,0,0,0
Packet	numeric	1	pkt_1 to pkt_8	0,0,0,0,0,0,0,0
			pkt_9 to pkt_16	0,0,0,0,0,0,0,0
			pkt_17 to pkt_24	0,0,0,0,0,0,0,0
			pkt_25 to pkt_32	0,0,0,0,0,0,0,1
TCP flags	binary/categorical	.A..S.	isURG	0
			isACK	1
			isPSH	0
			isRES	0
			isSYN	1
			isFIN	0

several orders of magnitude. Experiments with ncritic in table IV encourage that balancing the losses may yield more informative synthetic data sets.

- **Memorization-Averse Loss:** To prevent dataset memorization, a custom loss function for the generator could be constructed to discourage memorization, and so improve security of the architecture.

REFERENCES

[1] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers Security*, vol. 86, 06 2019.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[3] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional gans," 2017.

[4] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, "Creation of flow-based data sets for intrusion detection," *Journal of Information Warfare*, vol. 16, pp. 40–53, 2017.

[5] M. Ring, A. Dallmann, D. Landes, and A. Hotho, "Ip2vec: Learning similarities between ip addresses," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017, pp. 657–666.

[6] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 5767–5777. [Online]. Available: http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans.pdf

[7] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in neural information processing systems*, 2017, pp. 6626–6637.

[8] A. Cheng, "Pac-gan: Packet generation of network traffic using generative adversarial networks," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2019, pp. 0728–0734.

[9] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *CoRR*, vol. abs/1810.07795, 2018. [Online]. Available: http://arxiv.org/abs/1810.07795

[10] A. Pal and A. Das, "Torchgan: A flexible framework for gan training and evaluation," 2019.

[11] B. Claise, "Cisco systems netflow services export version 9," Oct 2004. [Online]. Available: https://tools.ietf.org/html/rfc3954

[12] R. K. Vigneswaran, R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security," in *2018 9th International*

Conference on Computing, Communication and Networking Technologies (ICCCNT), 2018, pp. 1–6.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] C. Dwork, "Differential privacy," in *Automata, Languages and Programming*, M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–12.

[15] H. Bae, D. Jung, H.-S. Choi, and S. Yoon, *AnomiGAN: Generative Adversarial Networks for Anonymizing Private Medical Data*, 01 2020, pp. 563–574.