Malicious and Benign URL Dataset Generation Using Character-Level LSTM Models

Spencer Vecile, Kyle Lacroix, Katarina Grolinger, Jagath Samarabandu
Department of Electrical and Computer Engineering
Western University
London, Canada
svecile@uwo.ca, klacroi6@uwo.ca, kgroling@uwo.ca, jagath@uwo.ca

***Abstract*—As technologies advance, so do the attacks on them. Cybersecurity plays a significant role in society to protect everyone. Malicious URLs are links designed to promote scams, attacks, and frauds. Companies often have web filtering algorithms that will blacklist specific URLs as malicious; however, due to privacy concerns, they will not give outside entities access to their cybersecurity data. Unfortunately, this lack of data creates a dire need for more data in cybersecurity research and machine learning applications. This paper proposes using machine learning to generate new synthetic URLs characteristically indistinguishable from the data they replace. To do this two character-level long short-term memory (LSTM) models were trained, one to generate malicious URLs and one to generate benign URLs. To assess the quality of the synthetic data two tests were performed. (1) Classify the URLs into malicious and benign to ensure the characteristics of the original data were preserved. (2) Use the Levenstein ratio to check the similarity between the real and synthetic URLs to ensure sufficient anonymization. The results from the classification test show that the synthetic data classifier only slightly underperformed the real data classifier; however, with having accuracy, precision, recall, sensitivity, and specificity above 99%, it can be concluded that the characteristics of the malicious and benign URLs were preserved. The Levenstein ratio tests showed a mean of 67% and 79% similarity for the benign and malicious URLs, respectively. In the end, the character-level LSTM model successfully generated an anonymized, synthetic dataset, that was characteristically similar to the original, which could pave the way for the publication of many more datasets in this way.**

***Index Terms*—LSTM, Deep Learning, Dataset Generation, Dataset Anonymization, Malicious URLs, Cybersecurity**

I. Introduction

When doing machine learning research or applying it to a problem there is always one constant regardless of the project or application; a need for vast amounts of data. This acquisition of data is a challenging process both in time and resources for many reasons but mainly because data contains valuable information that is not often given away for free. This case is especially true for private companies whose data can contain trade secrets or they may be in the business of data like social media companies. If they were to publicize these datasets first, they would have to ensure all private and privileged information was removed by having a data analyst go over the entire dataset which for a large dataset with billions of data points is just not feasible or would be incredibly costly. Now think about a world where these datasets could be sanitized not by a data analyst but by a machine learning algorithm. This is the end goal of this research and as a proof of concept, we will be applying this concept to network data, specifically, malicious and benign URLs.

Malicious URLs are links designed to promote scams, attacks, and fraud [1]. Victims are induced to click on a malicious URL that initiates the download of ransomware, viruses, trojans, or other types of malware [1]. A successful attack may compromise the target device or, in the case of a business, even its entire network. Before now, companies used rule-based network intrusion detection systems (IDS) [2] to detect attacks coming from malicious URLs and that worked fine but could quickly fail when the rules are subverted by a skilled attacker.

An IDS can be improved by applying machine learning to this task [3], but it must be trained on a good dataset for it to be effective. A good dataset has the following three characteristics: up to date, labeled, and contains realistic user behavior [4]. Realism is especially important as experimental results have shown that classification performance is significantly reduced when an IDS is used in a network environment that is significantly different than where its training data was extracted [5]. This should not be a problem as there is a lot of cybersecurity data globally, but most of it is held by private companies that are unwilling to release it for confidentiality reasons [6]. Companies often have web filtering algorithms that will blacklist specific URLs as malicious [7]; however, it is hard for outside entities to create predictive models to classify new and unseen URLs as malicious without access to these lists. Unfortunately, this lack of data creates a dire need for more data for cybersecurity research and machine learning applications. When companies are willing to give access to their network data, the data is most of the time heavily anonymized [4].

This is a significant problem because to train and evaluate supervised learning algorithms, high-quality, realistic datasets are required. By creating a character-level long short-term memory (LSTM) model [8] that will generate realistic, labeled URLs, this paper tackles issues related to realism and labeling. Intrusion detection systems can then be trained and evaluated by using this synthetic data. This paper proposes using machine learning to generate new synthetic URLs that are characteristically indistinguishable from the data they replace. Using this method, the new data will be created with the

2022 IEEE Conference on Dependable and Secure Computing (DSC) DOI: 10.1109/DSC54232.2022.9888835

same characteristics as the original without compromising the privacy of the original data provider. Eventually, this method could allow a corperation to outsource much of their cybersecurity development using sanitized datasets created through our method.

II. Related Work

A. Network Traffic Data Generation with Generative Adversarial Networks

An article by Cheng et al. discussed the use of Generative Adversarial Networks (GAN) [9] for the creation of IP packet-layer network traffic data [10]. The GAN has proven highly successful over the past few years with its ability to create highly realistic yet artificial images, text, audio, and video data [11]. Convolutional neural networks (CNN) GANs are used in this study. CNN GANs use CNN models for both the discriminator and generator. The researchers developed a new technique to encode network data specifically for use in CNN GANs. There are two steps to the encoding scheme: 1) converting packet byte values into subranges of sequential values, and 2) duplicating and mapping these converted values into a CNN input square matrix multiple times. Generating individual traffic types can lead to a success rate of up to 99%, whereas the generation of different traffic mixes produces a success rate of up to 88% [10]. This study shows excellent results using GANs to generate data so we attempted a replication of their work as a starting point. This also showed good generation results sometimes in terms of correctly formed packets but, we also found that it suffered from mode collapse causing most of the generated packets to be exactly the same. Due to this, it was decided to not use the GAN architecture but instead use a character-level LSTM model that is often used in text generation.

B. Long Short-Term Memory Rap Lyric Generation

Another study by Potash et al. took a closer look at using an LSTM [12] to generate language, specifically rap lyric generation [13]. LSTM models are best suited for classification, processing, and predicting in the case of time-series data. Their goal was to develop a model to generate lyrics similar to a given rapper's style but not identical to existing lyrics. This is called ghostwriting. The researchers compared the LSTM model to an n-gram model. They evaluated the models on both similarities of style and novelty. They used a similarity algorithm to compare produced lyrics with all verses from the same artist in order to assess the novelty of generated lyrics. Using an LSTM model, the researchers were able to generate lyrics without templates or constraints while also providing full verses instead of single lines. According to the results, the LSTM model excels at producing lyrics that also reflect the rhyming style of the target artist. In this study, LSTM is used to create data, which advances the proof of concept.

III. Method

A. Character-Level Long Short-Term Memory Models

Two character-level LSTM models were coded[1] and trained in Pytorch, one to generate malicious URLs and one to generate benign URLs. The LSTM models consist of an embedding layer, an LSTM layer, a dropout layer, and a fully connected layer. The embedding layer embeds each of the integers in an encoded training sample into a vector that is the length of the character dictionary. The LSTM has two layers and a hidden size of 512. The dropout layer helps with overfitting and has a probability of 0.5. Finally, a fully connected layer is at the end of the model and takes the hidden size of 512 in, and has an output the same size as the character dictionary. This is because each of the output neurons contains the probability of each character in the character set being the next character in a sequence.

B. Dataset

In our experiment, the URL dataset we used was ISCX-URL2016 from the Canadian Institute for Cybersecurity [14]. It contains 35,377 benign URLs, 12,000 spam URLs, 11,566 malware URLs and 9,965 phishing URLs. The spam, malware, and phishing URLs are concatenated together to create the malicious URL dataset containing 33,531 samples.

C. Preprocessing

Before the dataset can be used in the character-level LSTM model, it must be preprocessed into something the model can use. The first step is to get all the possible characters in the list of URLs and make a dictionary that encodes the characters to integers and another that decodes the integers back to characters. The dictionary must also have the 0th entry as the special padding token. This padding token is used to pad the URLs in a batch since the URLs will vary in length and LSTMs require a uniform length throughout a batch. It will also contain a special start token to indicate to the LSTM model the start of a sequence and a special end token that the LSTM model places at the end of a sequence to indicate the end of the generated sequence. The dictionary is then used to encode the entire set of URLs into integer representations. An example of this encoding can be seen in Fig. 1.

D. Training

The goal of training is to train the character-level LSTM models to generate data with the same characteristics as the original URL data but not generate the same data. The first step is splitting the data into train, validation, and test sets; 75%, 10%, and 15%, respectively. The training set was much larger since these models require a large amount of data to train. There is no test set because you can't test character-level LSTM models by comparing inputs to generated sequences. Their only input is the start token, then they generate URLs one character at a time based on the probability of what the model thinks the next character should be. Next, the batches

[1]Code available at https://github.com/svecile/URL-Dataset-Generation

must be created. In a batch of size one, take a sample and make two arrays, one for input and one for target, each being of size one longer than the length of the sample (length of n+1). The input array has the start token placed in the first position of the array and then each character of the sample in the rest of the cells. The target array has the characters of the sample placed in the first n cells and the end token placed in the last cell (n+1). This creates an offset between the input and target arrays. This is because the LSTM is trying to predict the next character; therefore, if the input is the start token, then the next character (target) is the first character in the URL and onward [15].

The next part of the batching process deals with the URL samples all having different lengths. Since they have varying lengths to do batched training, you must find the lengths of all the samples, then sort them by longest to shortest, and then pad all the shorter sequences with the special padding character [16]. The special padding token is ignored in the loss function; and therefore, it does not contribute to the loss. An example of the batching and encoding process can be seen in Fig. 1.

The training function uses cross-entropy loss and Adams optimization with a learning rate of 0.001, beta one of 0.9, and beta two of 0.999. It has a batch size of 128, and to ensure that we do not run into the exploding gradient problem with really long URLs, the gradient norm is clipped with a max norm of 5. Every epoch, a validation step is included, and if the validation loss doesn't improve after three epochs, then training will be stopped.

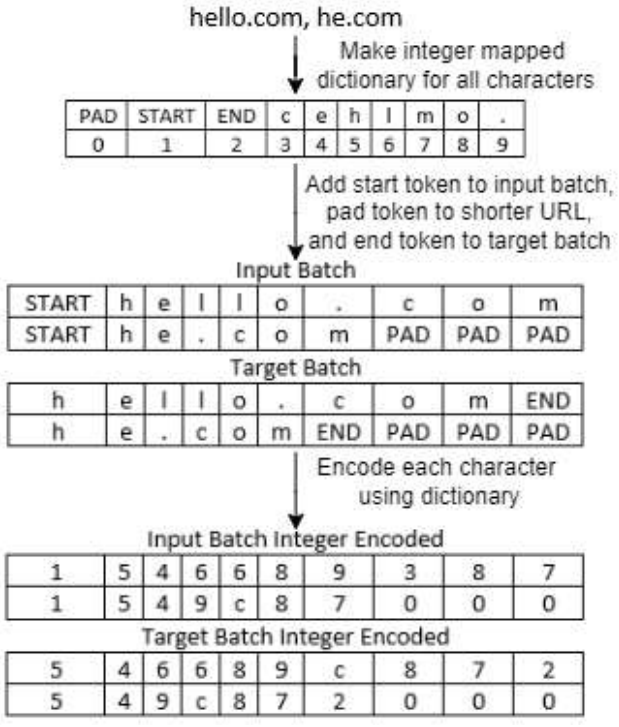

Fig. 1. Example of encoding one batch that includes two URLs hello.com and he.com.

E. Generating URLs

To evaluate the quality of the character-level LSTM models, two synthetic URL datasets will have to be generated. One is made up of synthetic malicious URLs generated by the LSTM model trained on the real malicious URLs. The second is made up of synthetic benign URLs generated by the LSTM model trained on the real benign URLs. To do this, two custom functions were used, one called 'predict' and one called 'sample'.

The predict function takes in a character, and a hidden state passes these to the model's forward function and obtains probabilities raw output and a new hidden state. The hidden state will be returned to the calling function in order to be passed to predict again if it needs another character. The raw output is given to the Softmax function, which outputs the probability that each character in the character dictionary should be the next character. This probability distribution is then passed to a random choice function that randomly selects one of the characters considering the Softmax probabilities. This random choice function can be used to add more variation (anonymization) to the model by making it output more than one choice and randomly selecting from those. This function then returns the chosen character and hidden state.

The 'sample' function calls the 'predict' function. The sample function takes in an LSTM model as input and outputs a fully formed URL string. It first passes the start token to the model and obtains a prediction for the next character, which is added to a list. It then continues to call the predict function passing in the most recently predicted character until the network predicts the special end token, which tells the algorithm the URL is fully formed and it stops predicting and returns the URL string.

To illustrate the generation process think back to the example of hello.com and he.com in Fig. 1. Now if an LSTM was trained on these two URLs it would definitely overfit and we would most likely get the same URLs back when we generate samples. If we wanted it to generate a URL string, first, the 'sample' function shown in Fig. 2 would be called and it would pass the start token to the trained LSTM's predict function as can be seen in Fig. 3, which would put the start tag through the LSTM and the output would be a probability distribution like the one shown in step 1 of Fig. 3 and a hidden state which is the memory of the network. The distribution is the Softmax probability that each of the characters in the dictionary should be the next character in the URL string. In this case, the probability that h is next is 1 because in our two training samples "h" is the only character that ever comes after the start tag and the LSTM would have learned this. The random choice function would see this, select it as the next character, and "h" along with the hidden state would be passed back to the sample function.

Now "h" is concatenated to the generated URL string and the first hidden state and "h" are passed back to the LSTM's predict function. Next, in step 2 of Fig. 3 it can be seen that "e" has a probability of 1 because "e" is the only thing that ever

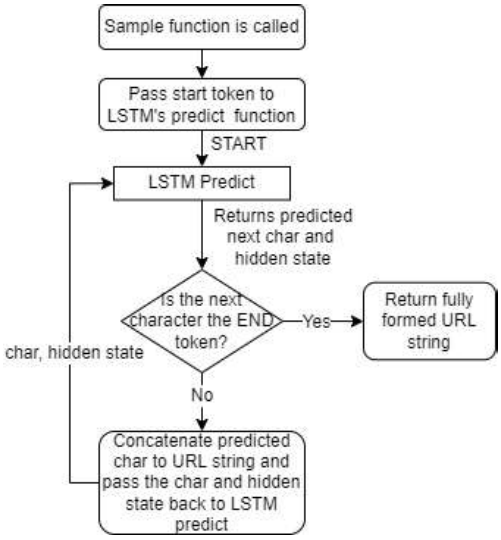

Fig. 2. Sample function flow diagram.

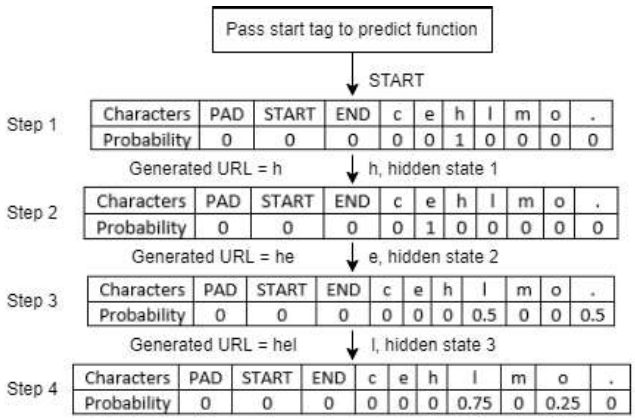

Fig. 3. Prediction function example.

comes after ”h” in our two training samples. So ”e” is chosen as the next character and it along with the new hidden state is passed back to the sample function. The sample function concatenates ”e” to ”h” and our URL is now ”he”. Next, ”e” and the hidden state are then passed back to predict and the probability distribution in step 3 of Fig. 3 shows that ”l” and ”.” both have a 0.5 probability this is because in hello.com ”l” comes after ”e” and in he.com ”.” comes after e. This is where the random choice function comes in and will pick one or the other adding variability to the generated samples. In this example, it picks ”l” as the next character. Now the URL string is ”hel” so ”l” is passed back to the predict function and the probability distribution may look like step 4 of Fig. 3. In this case ”l is predicted with a probability of 0.75 and ”o” has a probability of 0.25. This case may arise because both ”l” and ”o” can possibly come after ”l” in hello.com. The reason why ”l” has a higher probability than ”o” is because the LSTM’s hidden state (memory) is allowing it to remember the letters that came before the ”l” which were ”h” and ”e”. It would know that there should probably be two ”l’s” before it predicts ”o”. This process continues until the LSTM predicts the end token should be next at which time the sample function shown in Fig. 2 knows that the string is fully formed and to stop predicting and return it.

F. Evaluation Method

This research had two goals: to create data that was characteristically similar to the original in terms of classification and for that data to be different enough (anonymized) from the original data; therefore, it could be made publicly available[2]. To evaluate these requirements, two tests were performed. The first test attempted to classify the URLs into malicious and benign to satisfy the first requirement. The second test checked the similarity between the real and synthetic URLs to satisfy the anonymization constraint. For these tests to be performed, 50,000 malicious and 50,000 benign URL samples were generated. Then any URLs with illegal characters were removed, and any duplicates within each set were removed. This resulted in 48,890 synthetic benign samples and 40,541 synthetic malicious samples.

G. Classifier

The classifier is another LSTM model but modified to do binary classification. This test for characteristic preservation will involve giving the network a list of malicious URLs (class 1) and a list of benign URLs (class 0) and asking it to classify them into one of the two classes. For this test, two classifiers are trained one on the real benign and malicious datasets and one on the synthetic benign and malicious datasets. These two classifiers are then tested on both the real and the synthetic data. So overall there will be four parts to this test: train on real, test on real (TRTR), train on synthetic, test on synthetic (TSTS), train on real, test on synthetic (TRTS), and train on synthetic, test on real (TSTR). The performance of these classifiers is then compared and if the classifiers have similar performance, we can say the real data’s characteristics have been preserved in the synthetic dataset. Several metrics were used to compare the performance, including accuracy, recall, precision, sensitivity, specificity, and a confusion matrix. The classifier was trained with a learning rate of 0.001, binary cross-entropy with logits loss, a batch size of 128, and a two-layer LSTM with a hidden size of 512. It was allowed to go for a maximum of 50 epochs, but early stopping was set to 3; therefore, training will stop if the validation loss doesn’t improve. The data split used was 75% train, 10% validation, and 15% test when doing TRTR and TSTS. When TRTS is performed the entire synthetic dataset is used to test it since the classifier has not seen any of this data during training there will be no leakage. The same is true for TSTR where the entire real dataset will be used for testing since it was not used for training.

[2]The trained models can not be released as they would contain too much information about the real dataset.

H. Levenshtein Similarity Test

The Levenshtein ratio was used to evaluate how similar a synthetic URL is to a URL in the real dataset. Using Levenshtein distance [17], one can measure the difference between two sequences. The Levenshtein distance between words represents the number of single-character edits (insertions, deletions, and substitutions) necessary to convert one word into another. The Levenshtein ratio uses the Levenshtein distance to calculate the percent similarity between two strings or, in our case, two URLs. For example, two strings are exactly the same if they have a Levenshtein ratio of 100%. Using this ratio, every synthetic malicious URL is compared to every real malicious URL. Every synthetic benign URL is compared to every real benign URL, and the highest similarity ratio is recorded. These are then plotted on two histograms to visually represent the two similarity distributions. Then based on a user's similarity tolerance, packets above a certain similarity ratio can be removed.

IV. Results

A. Malicious URL LSTM model

The first model that was assessed was the malicious URL LSTM. The training took place over 18 epochs and the model converged to a training loss of about 0.6 and a validation loss of around 0.8. A visual representation of this can be found in Fig. 4.

The Levenshtein ratio histogram shown in Fig. 5 shows how similar a synthetic malicious URL is to URLs in the real dataset. Looking at the histogram, it can be seen that most of the data points are between 0.5-1.0. The average value for this histogram is 0.78913. This indicates that on average a synthetic data point is 78.913% similar to a data point in the real dataset. The maximum value that the histogram got was 1.0. This indicates that when the model created data, it created some copies of the real data points. The minimum value the histogram got was 0.15.

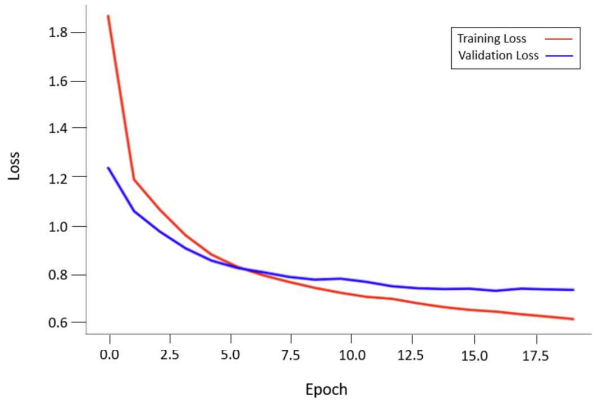

Fig. 4. Malicious URL LSTM model training graph.

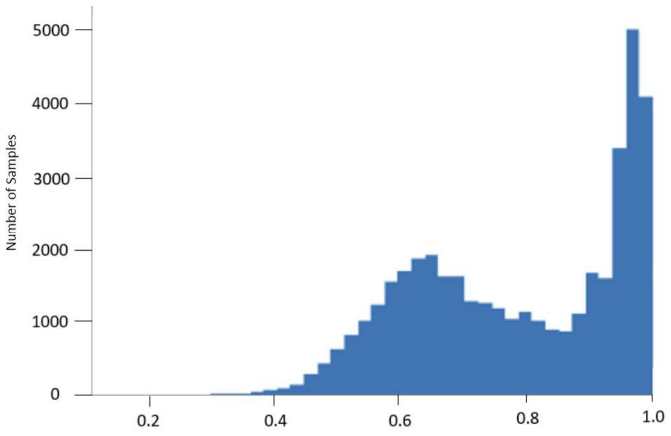

Fig. 5. Similarity of generated malicious URLs to real URLs used in training.

B. Synthetic Malicious URL Samples

Below are some examples of synthetic malicious URLs that were created by the LSTM. As you can see the URLs have proper top-level domain suffixes and are formatted correctly. Some include http and some are https. Some include the www. and some do not but all this is good as it shows the model has good diversity and that it is learning the URL syntax.

- http://amazon.co.uk/s/ref=sr_nr_n_2_2/202-2513040-1206232?ie=UTF8&rh=n:195522
- http://9779.info/%E8%A1%8D%E7%BA%B8%E8/
- http://www.accontamparoeb.com/wp-content/plugins/content/tmpl/ch/modules/mod_memage.css
- http://www.freatusa.com/ilap/images/nmh851/aol?products_readedpartnerId=2&nid=05
- https://spreadsheets.google.com/spreadsheet/viewform?formkey=dGJViHLWECjGUJMD3G2QC3==/

C. Benign URL LSTM model

The benign URL LSTM was the next model to be evaluated. The training period was 27 epochs. In training, the model's training loss converged to a value near 0.8 and its validation loss converged to a value of about 1. This is illustrated in Fig. 6.

According to the histogram shown in Fig. 7, the majority is around 60% similar. This histogram has an average value of 0.6719. As a result, on average, a synthetic data point has a 67.19% similarity to a real data point. A maximum value of 1.0 was obtained in the histogram. The minimum value the histogram got was 0.16.

D. Synthetic Benign URL Samples

Below are some examples of synthetic benign URLs that were created by the LSTM. Again the URLs have various schemes, subdomains, second-level domains, and top-level domains and are formatted correctly. Often, the domains in the synthetic samples are real however the subdirectory part of the URLs often leads nowhere which is the anonymization part of the algorithm working.

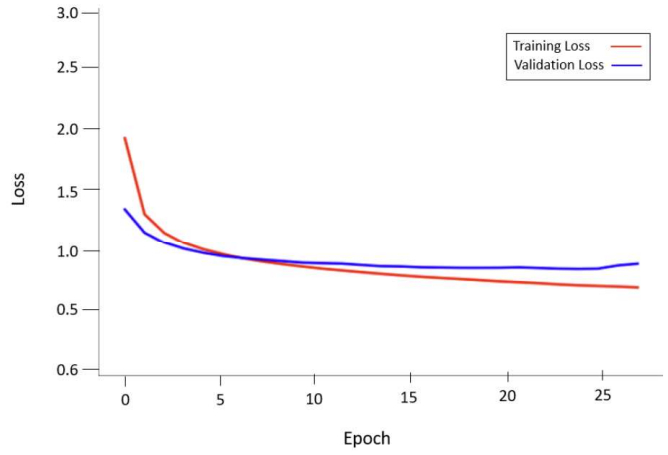

Fig. 6. Benign URL LSTM Training Graph

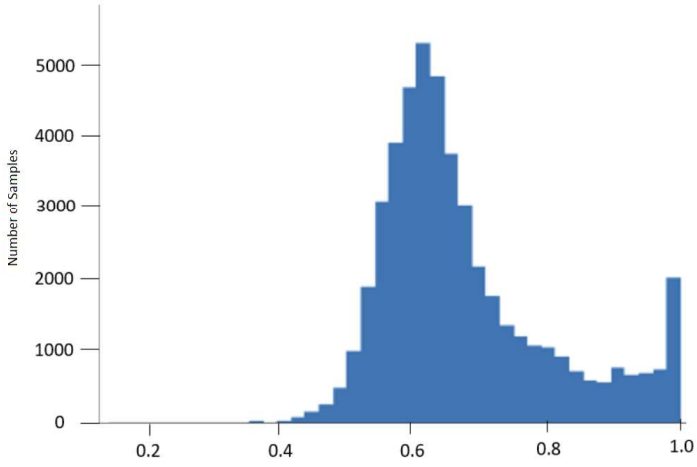

Fig. 7. Similarity of generated benign URLs to real URLs used in training.

- https://medium.com/dan-sanchez/duke-your-sony-couple-to-drab-temple-fakes-happening-with-mars-possibless
- https://www.gov.uk/government/organisations/review-in-detective-second-pental-illegal-goods
- http://kickass.to/bundles-player-2-killing-kongs-season-6-week-192s-song-crash/
- https://wordpress.org/showcase/electronics-dad-100-onlines-mailcategory/%5%35
- http://google.com/big/pep-watch-the-way-world-constructions-for-a-san/

E. Classifier Trained on Real Data

The classifier trained on real data was the next model to be evaluated. There were 25 epochs during the training period for this model. The model showed a very low training loss, converged to around 0.005. Additionally, the validation loss was extremely low, converged to about 0.01. This can be seen in Fig. 8. After the model was trained, the model was tested twice, once using the real test set and once using the synthetic dataset. The results of the test sets can be seen in Table 1.

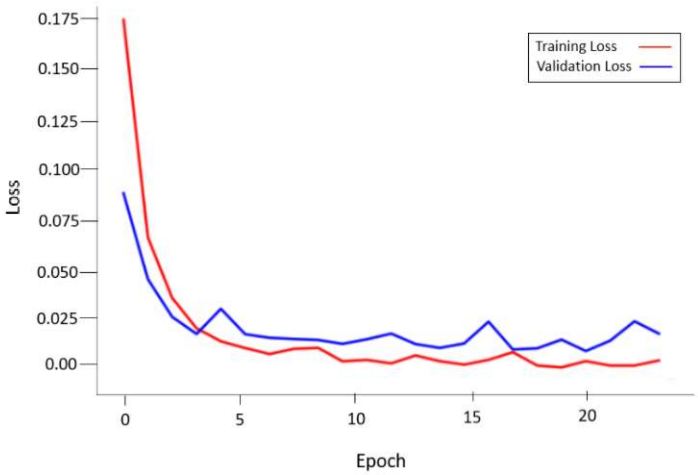

Fig. 8. Classifier trained on real data training graph.

TABLE I
CLASSIFIER TRAINED ON REAL DATA TEST RESULTS

Metrics	TRTR Values	TRTS Values
Accuracy	0.9966	0.9873
Recall	0.9967	0.9952
Precision	0.9951	0.9772
Sensitivity	0.9967	0.9952
Specificity	0.9965	0.9807

F. Classifier Trained on Synthetic Data

Next, the synthetic data classifier was evaluated. There were 10 epochs during the training period for this model. During training, the model showed a low loss, convergent to about 0.1. Furthermore, the validation loss was also relatively low, convergent to about 0.15. This can be seen in Fig. 9. After the model was trained, the model was tested using the generated test set. The results of the test sets can be seen in Table 2.

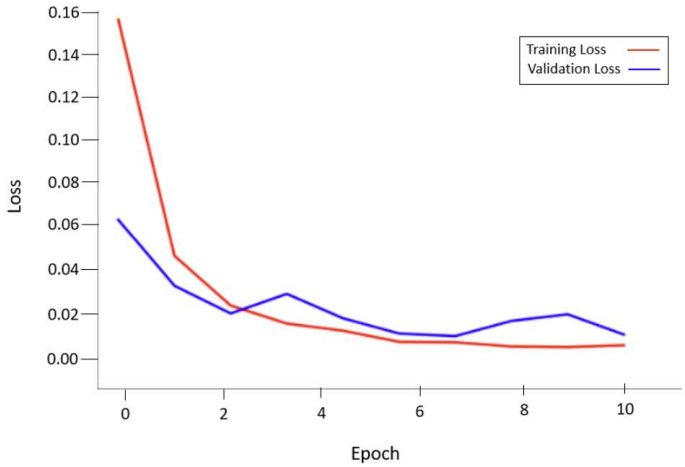

Fig. 9. Classifier trained on synthetic data training graph.

TABLE II
CLASSIFIER TRAINED ON SYNTHETIC DATA TEST RESULTS

Metrics	TSTS Values	TSTR Values
Accuracy	0.9957	0.9906
Recall	0.9948	0.9857
Precision	0.9958	0.9913
Sensitivity	0.9948	0.9857
Specificity	0.9966	0.9940

G. Confusion Matrices

The confusion matrix provides a visual representation of how a classification algorithm performs on its test set and what types of errors it is making. The confusion matrix provides count values for the number of correct and incorrect predictions. Ideally, numbers should appear on the diagonal and zeros elsewhere. Numbers outside the diagonal indicate that the model's performance is not perfect.

Starting with the classifier trained and tested on the real dataset in Fig. 10, it can be seen that the model does well overall. Looking at the diagonal on the matrix, there are only a few mislabeled data (30 out of 8,960). This indicates that the classifier can identify which real URL links are benign or malicious with high accuracy. Next, in Fig. 11 you can see how the classifier trained on the real data performs when classifying the synthetic dataset. It only misclassified 1136 out of 89,344 synthetic samples showing it can accurately distinguish between the malicious and benign samples even when it was not trained on the same dataset.

The confusion matrix of the classifier trained and tested on the synthetic dataset shown in Fig. 12 can be seen to be very similar to Fig. 10 only mislabeling 56 out of 13,312 samples. As with the confusion matrix of the real dataset, only a few data points have been mislabeled in this dataset. Finally, in Fig. 13 you can see how the classifier trained on the synthetic data performs when classifying the real data. It only misclassified 562 out of 59,904 real samples showing its ability to classify the malicious and benign samples even when the dataset is different from the one it was trained on.

The fact that all confusion matrices have a very small amount of false positives and false negatives shows that the performance of the two classifiers is similar suggesting that the synthetic dataset accurately captured the characteristics of the real dataset.

V. DISCUSSION/CONCLUSION

The results from the TRTR and TSTS classification tests show that the synthetic data classifier only slightly underperformed the real data classifier; however, with having accuracy, precision, recall sensitivity, and specificity above 99%, it's easy to conclude their performance is similar, and excellent. However, these metrics alone can't prove that characteristic preservation was achieved. The TRTR is mainly to show the best performance that could be achieved using only real data and the rest of the tests should strive to be as close to this as possible. The results of TSTS being very close to this TRTR is very good but the data could be in a format that is only

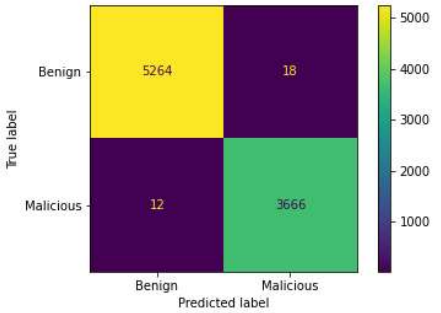

Fig. 10. TRTR classifier confusion matrix.

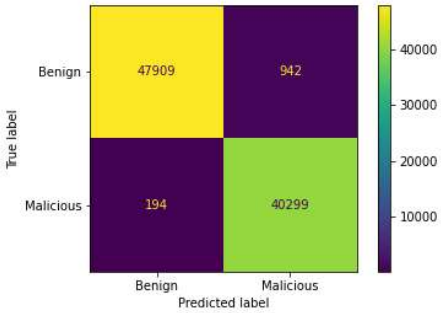

Fig. 11. TRTS classifier confusion matrix.

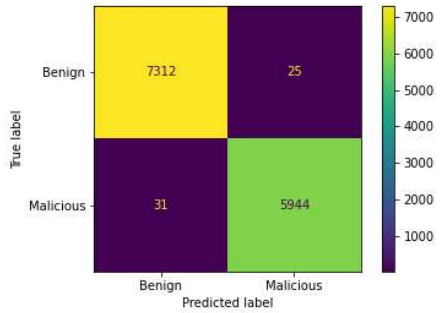

Fig. 12. TSTS classifier confusion matrix.

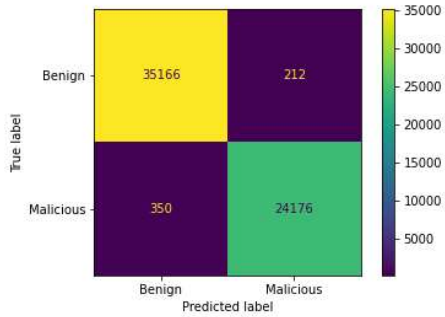

Fig. 13. TSTR classifier confusion matrix.

usable by character level LSTM models or there could be mode collapse in the model. This is where TRTS and TSTR come in.

TRTS is important because it shows that when a classifier only sees real data during training it is still able to classify the synthetic data with very similar accuracy. This means that the syntax used in the real URLs was captured by the LSTM that generated the synthetic data and the data it created should be usable by any model in the same way the original data was used. This is shown in Table 1 where All metrics are only different by approximately 0.01-0.02 which is a great result. However, this test still wouldn't pick up mode collapse in the model but it is still a good result that shows some characteristic preservation.

Finally, and most importantly is TSTR, as it will be able to detect mode collapse in the model and it represents our main use case where a company could outsource cybersecurity development using the synthetic dataset to alleviate privacy concerns. In Table 2 you can see that again the TSTR is only off from TRTR by about 0.01 in all metrics which is an excellent result. This result shows that there was no mode collapse in the model since there is enough diversity in the synthetic training set that the model was able to classify almost all of the real data correctly. This means that a company could use the synthetic dataset to develop a machine learning model and be confident that when it is deployed back at the company that hired them, on real network data, the results should be almost the same. TSTR combined with TSTS and TSTR and compared to TRTR show that the characteristics of what makes a malicious URL malicious and a benign URL benign were preserved in the generated dataset.

One can visually verify the characteristic preservation by looking at the examples in sec 4 B and D. You may also notice that the last example in sections 4 A and B are both google websites however they have different subdomains. The classification model can correctly classify these but defacement and typosquatting URLs were not used in the training data so it will be left to future work to see if the algorithm can deal with these situations.

The Levenstein ratio tests showed a mean of 67% similarity for the benign URLs and a mean of 79% similarity for the malicious URLs. 67% for the benign URLs shows a sufficient level of anonymization, with some still being exact copies of the real ones; however, these can be discarded from the final set. The malicious URLs had more similarity with a mean of 79%, but this still should be sufficient anonymization. However, the malicious dataset had a significant amount more samples with 100% similarity. We think this can be attributed to the datasets used to train the character-level LSTM models. The benign LSTM model's dataset had around half a million unique URLs initially, and then after duplicates and domain-only URLs were removed, it was down to a little over 35,000. When manually inspecting the dataset, it seemed to have more variance than the malicious dataset. On the other hand, the malicious dataset had many URLs with the same base URL (e.g. www.google.ca/) but slightly different queries or paths added to the base URL (e.g. https://www.google.ca/search?q=hello, https://www.google.ca/search?q=goodbye). We think this caused some overfitting and caused the LSTM model to generate many URLs with the same base URL but different paths attached, which increased the mean similarity. This problem could be solved by finding a dataset with more variance in the malicious URLs. Despite this, the character-level LSTM model successfully generated an anonymized synthetic dataset that was characteristically similar to the original. It is our hope that this research paves the way for the publication of many more datasets in this way; giving researchers better access to high-quality, real-world data for use in their research and companies easier access to outsourcing cybersecurity development.

REFERENCES

[1] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious urls," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 1245–1254.

[2] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[3] H. Kaur, G. Singh, and J. Minhas, "A review of machine learning based anomaly detection techniques," *arXiv preprint arXiv:1307.7286*, 2013.

[4] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, vol. 1, pp. 108–116, 2018.

[5] S. Mahdavifar and A. A. Ghorbani, "Application of deep learning to cybersecurity: A survey," *Neurocomputing*, vol. 347, pp. 149–176, 2019.

[6] S. Molnár, P. Megyesi, and G. Szabó, "How to validate traffic generators?" in *2013 IEEE International Conference on Communications Workshops (ICC)*. IEEE, 2013, pp. 1340–1344.

[7] M. S. I. Mamun, M. A. Rathore, A. H. Lashkari, N. Stakhanova, and A. A. Ghorbani, "Detecting malicious urls using lexical analysis," in *International Conference on Network and System Security*. Springer, 2016, pp. 467–482.

[8] K. Hong, "Programming a poet: Poetry text generation using lstm," 2020.

[9] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[10] 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)*. IEEE, 2019, pp. 0728–0734.

[11] A. Aggarwal, M. Mittal, and G. Battineni, "Generative adversarial network: An overview of theory and applications," *International Journal of Information Management Data Insights*, p. 100004, 2021.

[12] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv preprint arXiv:1402.1128*, 2014.

[13] P. Potash, A. Romanov, and A. Rumshisky, "Ghostwriter: Using an lstm for automatic rap lyric generation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1919–1924.

[14] M. S. I. Mamun, M. A. Rathore, A. H. Lashkari, N. Stakhanova, and A. A. Ghorbani, "Url dataset (iscx-url2016)," 2016. [Online]. Available: https://www.unb.ca/cic/datasets/url-2016.html

[15] C. Lara, "Character-level lstm in pytorch," https://github.com/LeanManager/NLP-PyTorch/blob/master/Character-Level%20LSTM%20with%20PyTorch.ipynb, 2019.

[16] H. Trivedi, "Minimal tutorial on packing and unpacking sequences in pytorch," https://github.com/HarshTrivedi/packing-unpacking-pytorch-minimal-tutorial, 2019.

[17] G. Navarro, "A guided tour to approximate string matching," *ACM computing surveys (CSUR)*, vol. 33, no. 1, pp. 31–88, 2001.