2018 IEEE Symposium on Security and Privacy Workshops

Bringing a GAN to a Knife-fight: Adapting Malware Communication to Avoid Detection

Maria Rigaki, Sebastian Garcia
Faculty of Electrical Engineering
Czech Technical University in Prague
Prague, Czech Republic
Email: {maria.rigaki, sebastian.garcia}@fel.cvut.cz

***Abstract*—Generative Adversarial Networks (GANs) have been successfully used in a large number of domains. This paper proposes the use of GANs for generating network traffic in order to mimic other types of traffic. In particular, our method modifies the network behavior of a real malware in order to mimic the traffic of a legitimate application, and therefore avoid detection. By modifying the source code of a malware to receive parameters from a GAN, it was possible to adapt the behavior of its Command and Control (C2) channel to mimic the behavior of Facebook chat network traffic. In this way, it was possible to avoid the detection of new-generation Intrusion Prevention Systems that use machine learning and behavioral characteristics. A real-life scenario was successfully implemented using the Stratosphere behavioral IPS in a router, while the malware and the GAN were deployed in the local network of our laboratory, and the C2 server was deployed in the cloud. Results show that a GAN can successfully modify the traffic of a malware to make it undetectable. The modified malware also tested if it was being blocked and used this information as a feedback to the GAN. This work envisions the possibility of self-adapting malware and self-adapting IPS.**

***Keywords*—GAN, malware, intrusion detection, intrusion prevention, network behavior, network traffic obfuscation**

I. Introduction

A significant part of the most important current malware can be detected in the network using static detection rules and reputation systems. Although these rules have to be created manually, their effectiveness is relatively good. There are also plenty of anomaly detection algorithms that are able to identify anomalies in network traffic. These algorithms are complex, but they have become a standard in modern detection systems. Although new malware is usually not detected because of its unknown characteristics, recent advances in machine learning detection methods show very promising results in this area. Due to the detection systems currently being implemented and the trends in new malware we foresee future malware automatically adapting their network traffic to avoid detection. This adaptation will probably use machine learning techniques. Such adapting malware would have severe consequences to our defensive ability.

The adaptation of malware to changing conditions in detection methods has been so far implemented mostly by static methods. Examples of evasion methods are the Domain Generation Algorithms (DGA), Fast-Flux techniques, and the constant modification of the infrastructure used. Every time an IP address, domain or tool is changed, the Indicators of Compromise (IoC) have to be adapted. These evasion techniques work because the most common detection methods are static and still trying to block IP addresses and domains. The need for a malware to adapt would only be forced by a detection algorithm that does not use static data, but behavioral methods.

With the advent of more complex detection methods, we expect the malware to start adapting. However, this new area of network security is quite unexplored so far. The security community does not have a clear idea of how the new detection methods should work if the malware adapts. It is in this context that we studied adapting malware traffic by using a GAN which instructs the malware how to adapt to a real behavioral detection system. Our hypothesis is that a GAN can help malware to adapt, given the specific traffic that is going to be mimicked. Such adaptation would make a malware not easily detected.

Our proposed method is to use a GAN to learn to imitate Facebook chat traffic and consequently communicate to the malware how to modify its traffic so that it will not be blocked. The GAN is fed with real Facebook chat network flow parameters in order to train for a predefined number of epochs. The trained GAN generator communicates its output parameters to the malware. The malware adapts its traffic according to these parameters and continues its work. The malware traffic never stops in the network, it just changes. In a subsequent phase, the malware detects if it was blocked or not during the last time window and it uses this information to signal the need for additional training and data to the GAN. The detection and blocking of the traffic is done by the Stratosphere IPS system [1], which uses behavioral methods based on machine learning. In order to keep the setup realistic, the feedback signal is a special network measurement done by the malware to know if the Internet access is blocked. There is no signal being generated by the blocking device.

Two very important constraints were set in place during the experiments: First, the malware should be real and perform real malware actions in the network. This was the only way of knowing that the modifications to the traffic did not affect the malware operation. The second constraint was that the blocking of the malware should also be real, because this may perturb its operation.

Results show that starting from a *malware blocked* situation, it is possible to modify the malware behavior to emulate a normal traffic profile, and therefore become unblocked.

DOI 10.1109/SPW.2018.00019

Moreover, it was possible to accomplish these results with a relatively short training regime and without the need for a large amount of data. Our experiments were conducted with only 217 network flows from the normal traffic. After 400 epochs of training the blocking percentage dropped to zero. From the malware perspective, the malicious actions were successful and its operation remain unaffected.

The contributions of our work are:

- The use of a GAN to model network traffic behavior that mimics a given traffic profile such as Facebook chat.
- The modification of the behavior of a real malware that mimics a normal traffic profile based on input parameters from the GAN.
- A real-life experiment of traffic blocking using the Stratosphere IPS in a router.
- A novel feedback mechanism of the malware and GAN to adapt depending on the blocking state.

Although our method focuses in modifying malware behavior for the future purpose of improving detectors, it may also be applied as a censorship circumvention method. Censorship circumvention consists in adapting traffic in such a way that censors can not block it. The TOR network has had historically several implementations for this purpose, such as Stegotorus [2], SkypeMorph [3], Dust [4] and ScrambleSuit [5]. The last two actually modify the size and inter-arrival times of packets to resemble random distributions.

Our proposal is, in a sense, similar to Dust and ScrambleSuit anti-censorship methods. The difference would be that our method works in the flow level instead of the packet level, and that it changes the values to mimic real normal applications and not random distributions.

II. Background and Related Work

A. Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a class of generative models mainly focused on the generation of new samples following a distribution learned from a training set [6]. They consist of two neural networks: a generator (G), and a discriminator (D). The generator receives as input a random vector z and it is trained to generate data that is eventually indistinguishable from the real dataset. The discriminator is trained using two batches of data: the training data x sampled from the real data probability distribution p_{data} and the data generated by the generator $G(z)$.

The discriminator loss $J^{(D)}$ is typically computed as a cross entropy loss for a binary classifier with a sigmoid output:

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}} log(D(x)) - \frac{1}{2}\mathbb{E}_z log(1 - D(G(z)))$$

A formulation for the GAN as a zero-sum game would suggest that the generator loss would be computed by

$$J^{(G)} = -J^{(D)}$$

However, this approach does not seem to work well in practice. Instead, a heuristic approach is usually followed for the generator loss:

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z log(D(G(z)))$$

While the majority of the work around GANs is typically about computer vision problems, several researchers have proposed using GANs in other domains. Sequential GANs have been proposed both in discrete [7], [8] and continuous forms [9]. Network traffic parameters as time-series data resemble a continuous sequence.

The convergence of GANs is usually difficult to estimate since the loss functions are not a good indication of the quality of the generator's output. This problem was solved by using an external entity (IPS) that evaluates the results of the generated traffic.

B. Stealthy Malware Communication

Malware typically hides its C2 channels to remain undetected. Popular methods include DGA, Fast-Flux, HTTPS, tunneling, and the abuse of popular services such as Twitter, Gmail or Google Docs [10], [11]. These evasion approaches do not try to mimic other traffic but to use varied, trusted and difficult to find infrastructures.

Another proposed approach for camouflaging malware traffic was to use Format Transforming Encryption (FTE) which uses regular expressions that "convert traffic flows to another application protocol using a regular expression describing the target protocol" [12]. FTE was initially proposed for censorship circumvention [13] but in [12] it was used to modify the traffic of an existing malware named Zeus. Sheridan et al. [14] modeled DNS requests within a network as Poisson distributions and proposed to use a dynamic Poisson distribution model in order to conceal malicious DNS traffic that can be used further as a C2 channel. While effective, the channel efficiency was rather low.

C. Red Team Tools

During a penetration test, red teams often rely on post-exploitation frameworks that allow them to emulate malware behavior in the network and host environments. These frameworks offer some degree of adaptation, usually by the creation of profiles or scripted actions. Kaldera [15] is an adversary emulation framework that utilizes automated planning in order to emulate different post-exploitation attack stages. One of the limitations of the framework is that it does not model a C2 channel. Empire [16] is an open source PowerShell and Python post-exploitation agent that has adaptable communication capabilities. From a network behavior perspective the only configurable parameters offered are the jitter and delay between messages. Cobalt Strike [17] is another commercial framework that offers the possibility to create *Malleable C2 profiles* which define the behavior of the agent. Similar to [16] the only network behavior related parameters are jitter and delay. To the best of our knowledge there is no post-exploitation framework or tool that offers advanced network behavior configuration and the ability to mimic traffic profiles.

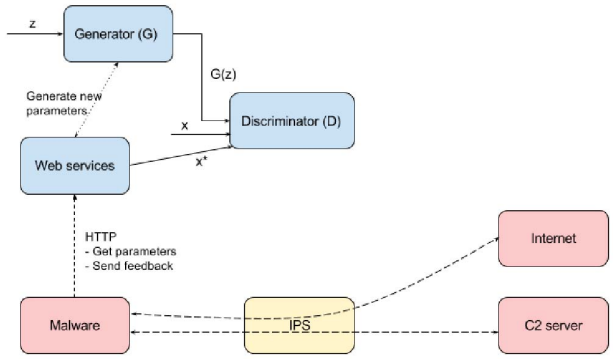

Fig. 1. Network experiments setup. The GAN is implemented independently and communicates with the malware through a web service. The malware gets the parameters and modifies its traffic in real time. The C2 channel should be maintained and should be operational. The IPS blocks all the traffic that does not look like Facebook chat.

III. Generation Method

A. Threat Model

There are three main components in our threat model: (1) the detector, (2) the malware, and (3) the C2 server.

The detector is based on the Stratosphere Linux IPS software, deployed in a Turris router [18]. It inspects all the network traffic between the infected host and the Internet and decides if the traffic of each IP address is allowed to pass or not. It is programmed to block all the traffic that does *not* match the normal traffic profile we have chosen to mimic. In our work this is Facebook messaging traffic, i.e. Facebook chat. The detector implements a behavioral detection, which means that it knows how the Facebook chat traffic looks like and it is configured to only let that traffic through. It does not perform Deep Packet Inspection (DPI) or any other signature based detection.

B. GAN

Both the generator and the discriminator are Recurrent Neural Networks (RNNs). More specifically, they use a Long-Short Term Memory (LSTM) architecture [19]. For the real data, one-side label smoothing is applied [20]. The GAN components were implemented with the Keras Framework [21].

A web service component is used to facilitate the communication between the GAN and the malware. This web service exposes two API calls: *get_params* and *feedback*. The first API call loads the saved generator model and produces new traffic parameters which are then send back to the malware as a JSON object. The generated parameters are also saved to a local file database along with their generation timestamp.

The *feedback* API call loads both the saved generator and discriminator models, and based on the feedback received (blocked traffic or not) it adds the parameters of the previous time window to the current dataset. Subsequently, another training round of 5 epochs is performed using the augmented dataset.

C. Malware

To evaluate our method we used the open source Remote Access Trojan (RAT) called Flu [22]. Flu was modified to receive the input from the GAN generator and adapt its network behavior. Flu consists of a client that is written in C# and a C2 server that is written in PHP. The C2 server was deployed in Amazon Web Services (AWS). The infected client was deployed in a Windows 7 virtual machine in the Stratosphere laboratory network at the CTU University.

The communication between client and server is established over HTTP. The Flu client performs the following actions in sequence:

1) Checks if the server is online.
2) Sends a heartbeat message with a unique identifier.
3) Retrieves a XML file from a pre-configured URL path. The XML file contains a command id and a list of commands that the client needs to execute.
4) Executes any ”fresh” commands in the XML file.

These steps are performed sequentially and between each step there is a delay of three seconds. This makes the communication channel highly periodic and therefore predictable.

We modified the Flu client to alter its network behavior. The modifications affect the timing, duration and request sizes of each flow. These features were adapted based on the parameters provided by the generator. Before any communication between the client and the server, Flu interrogates the GAN using the HTTP API and retrieves three parameters: **total byte size**, **duration** of the next network flow, and **time delta** between the current flow and the next one.

The size of each HTTP request is computed by taking into account the minimum size of the original request, the size of the TCP three-way handshake and the expected HTTP response size from the server. The HTTP request body is padded in order to match the total byte size value that was generated by the GAN. The duration of the network flow is controlled at the TCP level by keeping the connection open for the time indicated by the duration value. The time-delta parameter defines the amount of time Flu needs to wait before initiating a new request towards the server. In case the time-delta is smaller than the duration of the flow a new flow and therefore a new request is initiated before the duration of the ongoing request elapses. This is accomplished by the use of a new thread that is responsible for fetching new parameters from the GAN and launching a new request towards the server.

Finally, during the start-up of the client a five minute timer is initiated, which is responsible for triggering the checks on whether or not the client is blocked.

D. Intrusion Prevention System

An important part of the methodology was to evaluate our proposal in a real security environment. For this, a real behavioral Intrusion Prevention System was needed. The Stratosphere Linux IPS (slips) system [1] was chosen because it models behaviors in the network and uses machine learning algorithms to detect those behaviors in the network.

The core method consists of reading the network flows from the traffic to model and extracting three features for each

flow: its duration, its size and its periodicity. The periodicity is a ratio between the time differences between the last three flows. Each combination of features for each flow is assigned a state. Since flows are sequential, states are sequential. From these transition states, a Markov model is created, that is the final model of the traffic behavior. These models are stored on disk. During detection, the unknown traffic is converted to a sequence of states. Slips then computes the probability that each unknown sequence of states matches the generated stored model. Slips is integrated with the *iptables* Linux firewall to block the IP addresses that do not match a normal model or that match a malware model.

IV. Experiments

A. Initial Data Collection

The data used for training the GAN and for generating of the detection models in Slips were network traces gathered while two users were using Facebook chat. The communication between the two users was intermittent as we tried to simulate normal user behaviors. The messages included text, images, links and documents. The traffic capture was performed in the router and it lasted approximately one day.

B. Data Pre-processing and Feature Selection

The network captures were converted to bidirectional flows using the Argus suite [23]. They resulted in 217 distinct flows of Facebook chat traffic. The features used for the GAN training and also as input for the malware traffic modification were the **duration of the network flow**, the **total number of bytes in the flow** and the **inter-flow time** calculated from the time-stamp of each flow. Flows were sorted in time. The data were converted to time series to be used as input for the discriminator.

C. GAN Hyperparameters

LSTM hidden layers The generator and discriminator are unidirectional LSTM networks. Both networks have a depth of one, each with 128 hidden units and a sequence length of 6.

Training The LSTMs are trained using batch gradient descent. It uses the Adam optimizer with a learning rate 0.001. The discriminator was trained for three epochs for every one epoch of training for the generator.

Latent input variable (z): A sequence of 16 random normal vectors with $\mu = 0$ and $std = 1.0$.

D. Time Windows

The width of the time window was set to five minutes in both the detector and the malware. The detector collects traffic for the duration of the time window and it takes a decision on whether the traffic matches the stored Facebook model. From the malware perspective the time window was used to check whether or not it got blocked by the detector. This was accomplished by sending an HTTP request to the well known stable internet service www.google.cz. The result was used to signal a positive or negative feedback to the GAN and trigger additional training.

TABLE I. Percentage of unblocking actions per number of training epochs in five runs

# Epochs	Mean (%)	Std (%)
0	0.0	0.0
50	16.88	3.26
100	67.37	13.89
200	67.54	6.84
300	61.39	7.84
400	63.42	10.37

TABLE II. Percentage of blocking actions per number of training epochs in five runs

# Epochs	Mean (%)	Std (%)
0	0.0	0.0
50	44.44	11.18
100	16.72	14.15
200	3.16	2.64
300	0.08	1.23
400	0.0	0.0

It is worth noting that the time window was a known parameter in the malware configuration. The reason for this decision was twofold: the size of the time window needed to be set in a value that would permit meaningful detection but at the same time would not delay the experiment significantly; in addition, developing a technique to identify the decision time window from the malware perspective was considered as a separate topic worthy of its own research.

E. Evaluation

The evaluation was done by first pre-training the GAN and subsequently using only the generator to get new parameter sequences for the malware. Then, the channel efficiency was computed, mainly in terms of number of flows per time window. Finally, the original dataset was augmented based on the external feedback signal. The evaluation focused on three questions: (a) can the GAN mimic the traffic profile of Facebook chat, (b) what is the C2 channel efficiency in terms of the amount of undetected flows and (c) can the feedback loop be used as a way to augment the dataset in case we have very little data to begin with?

Our experiments included two verification steps to guarantee the correctness of the approach. First, the Flu malware kept connecting to its C2 server and receiving orders. Second, the Facebook traffic was replayed continually to see if Slips blocked it or not.

1) Network Traffic Profile Generation: In this experiment the GAN is trained for a number of different epochs and the trained generator is used as a source of network parameters. Then, we let the malware communicate with the C2 server for a fixed amount of time (four hours) and recorded the amount of times the traffic was blocked by the detector.

The detector makes three types of decisions at the end of each time window: (1) if it gets more than three flows from the malware it decides to block the traffic or not. If there are less than three flows it just logs the traffic without blocking. The threshold of three flows was chosen to avoid blocking the malware perpetually due to SYN packet re-transmission

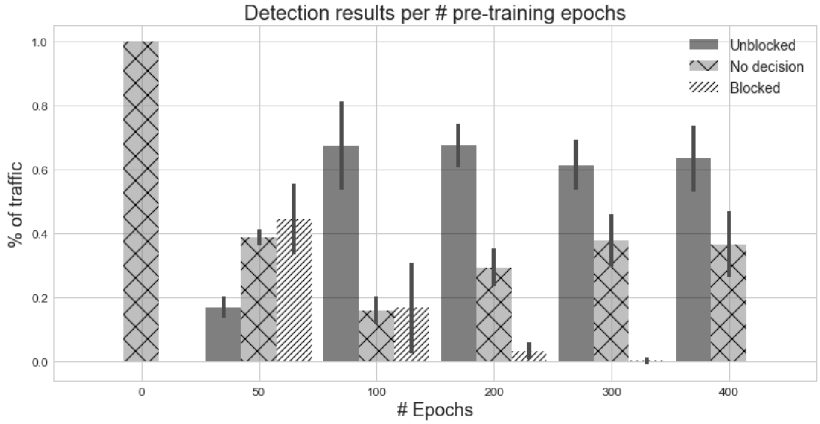

Fig. 2. Detection results for each of the pre-trained models. The x-axis represents the number of training epochs and the y-axis the average percentage of traffic that was either blocked, unblocked or unblocked due to no decision.

from the TCP stack which occurs when the malware is already blocked. Each experiment was repeated 10 times. The mean and standard deviations of the results are shown in tables I and II for the unblocking and blocking actions respectively.

Figure 2 shows clearly that as the number of training epochs increases, blocking actions go to zero. However, the unblocking actions are relatively stable.

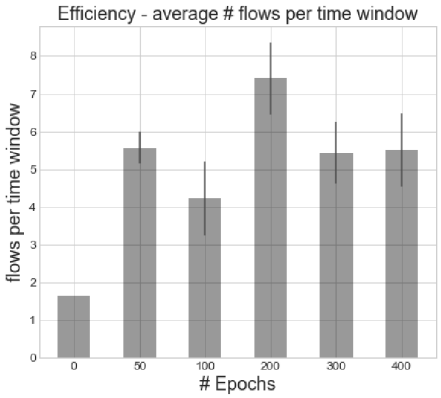

Fig. 3. Number of flows per time window for each of the pre-trained models. The x-axis represents the number of training epochs and the y-axis the average number of flows per time window of five minutes.

2) Communication Channel Efficiency: By default the C2 channel in Flu has a three second periodicity. This means 75-100 flows in a five-minute time window. Forcing the malware to behave as a web service that exhibits intermittency, influences the amount of traffic between client and server. However, Figure 3 shows that the average number of flows per time window is higher than 5 for the best performing generators. This implies that the server and the client communicate on average at least once per minute, which makes this a feasible channel for a C2. If someone operates a large number of clients it is perfectly acceptable to experience small delays between sending a command and receiving the results.

3) Feedback Loop: We ran experiments where we selected a random subset of the dataset of 35 data points as a pre-training set and used the detector's decisions as a way to augment the dataset with new data. The results were promising and led to a gradually better performance, but they were not statistically significant. This indicates that for this particular setup the training data requirement was not high and good results were obtained even with a subset of the data.

V. Analysis of results

The results suggest that it is possible to use GANs for network traffic mimicking. After enough number of training epochs we managed to reduce the number of blocking actions to zero even with a relatively small dataset. The percentage of unblocking actions was 63.42% while the rest of the time the detector allowed the traffic to go through due to the number of network flows being below the detection threshold. This indicates that the approach can be used for practical purposes. Under these circumstances the malware can keep behaving like Facebook forever and not be blocked. One of the most important constraints of our experiments was that the communication between malware and C2 server would continue unimpeded. Indeed, the most successful GAN models (produced after 300-400 epochs of training) generated more than one C2 flow per minute. While there is room for improvement, this result is acceptable from a deployment perspective.

Finally, the idea of using implicit network feedback in the form of blocking or unblocking detection seems promising and merits further exploration. The experiments were successful in that additional data and training improved our results over time, but we attribute the improvement more to the additional training than to the data augmentation.

VI. Conclusion

The automatic adaptation of malware to blocking conditions is an important concern for current and future security mechanisms. To better understand the adaptability limits of malware, and the future needs for the blocking systems of our community, we implemented a GAN that modifies the behavioral patterns of a specific malware in the network in order to mimic Facebook chat.

Our proposed method consists of a GAN that was trained with real Facebook chat traffic and directly communicates

with a malware sample. The malware adapts the behavioral characteristics of its traffic according to the GAN parameters. The malware has a C2 channel on the internet that has to be kept operative. All the traffic is monitored by the Stratosphere Linux IPS system in a router that blocks all the traffic that does not behave like Facebook chat traffic. The malware also monitors whether it is being blocked and uses this information as a feedback signal to improve the GAN models.

The results are promising, showing that it is in fact possible to improve the chances of the malware not being blocked by mimicking Facebook chat traffic. Moreover, it is shown that a small amount of data for training the GAN is enough for this use case.

Our future work will include certain improvements. First, we want the malware traffic to not only look like Facebook to the algorithm, but also to a human. Therefore we plan to use HTTPS connections and probably some mimicking of Facebook certificates and content. In addition we would like to test the generalization of the approach using different traffic profiles. From the GAN perspective we plan to explore alternative architectures such as using Convolutional Neural Networks in the discriminator as well as looking into other ways to take advantage of the feedback capabilities of our setup. We will also test the GAN against different types of network detectors and combinations of them, including an improved Stratosphere Linux IPS. From an implementation perspective we would like to combine the malware and the GAN in order to avoid the need of having a separate deployment of a web service to facilitate communication between the two.

Among the possible implementations of our work are: a standalone testing tool against Intrusion Detection Systems, an external framework for adapting any tool with pre-defined behaviors, and as a tool for censorship circumvention. We believe that further research in this area may prove substantial for the security community.

ACKNOWLEDGMENT

The authors thank Ondrej Lukas for his implementation of the Slips system in the Turris Routers. This research was partially supported by the Czech TACR project no. TH02010990.

REFERENCES

[1] S. Garcia, "Modelling the Network Behaviour of Malware To Block Malicious Patterns . the Stratosphere Project : a Behavioural Ips," in *Virus Bulletin*, no. September, 2015, pp. 1–8. [Online]. Available: https://www.virusbtn.com/pdf/conference_slides/2015/Garcia-VB2015.pdf

[2] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh, "Stegotorus: a camouflage proxy for the tor anonymity system," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 109–120.

[3] H. Mohajeri Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "Skypemorph: Protocol obfuscation for tor bridges," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 97–108.

[4] B. Wiley, "Dust: A blocking-resistant internet transport protocol," *Technical rep ort. http://blanu. net/Dust. pdf*, 2011.

[5] P. Winter, T. Pulls, and J. Fuss, "Scramblesuit: A polymorphic network protocol to circumvent censorship," in *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. ACM, 2013, pp. 213–224.

[6] 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.

[7] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient." in *AAAI*, 2017, pp. 2852–2858.

[8] K. Lin, D. Li, X. He, M.-t. Sun, and Z. Zhang, "Adversarial ranking for language generation," in *Advances in Neural Information Processing Systems*, 2017, pp. 3158–3168.

[9] O. Mogren, "C-rnn-gan: Continuous recurrent neural networks with adversarial training," *arXiv preprint arXiv:1611.09904*, 2016.

[10] P.-M. Bureau and C. Dietrich, "Hiding in plain sight - advances in malware covert communication channels," 2015. [Online]. Available: https://www.youtube.com/watch?v=3o4sVGmf4Dk

[11] A. Lehtiō, "C&C-As-A-Service: Abusing Third-party Web Services As C&C Channels," in *Virus Bulletin*, 2015. [Online]. Available: https://www.virusbulletin.com/uploads/pdf/conference_slides/2015/Lehtio-VB2015.pdf

[12] X. Zhong, Y. Fu, L. Yu, R. Brooks, and G. K. Venayagamoorthy, "Stealthy malware traffic-not as innocent as it looks," in *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*. IEEE, 2015, pp. 110–116.

[13] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Protocol misidentification made easy with format-transforming encryption," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 61–72.

[14] S. Sheridan and A. Keane, "Improving the stealthiness of dns-based covert communication," in *European Conference on Cyber Warfare and Security*, 2017.

[15] A. Applebaum, D. Miller, B. Strom, C. Korban, and R. Wolf, "Intelligent, automated red team emulation," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 2016, pp. 363–373.

[16] "Empire - powershell and python post-exploitation agent." [Online]. Available: https://github.com/EmpireProject/Empire

[17] "Cobalt strike," https://www.cobaltstrike.com/.

[18] CZ.NIC, "Turris Project." [Online]. Available: https://www.turris.cz/en/

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[21] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[22] "Flu project," https://github.com/fluproject/flu, 2017.

[23] "Argus - auditing network activity," https://qosient.com/argus/, 2017.