Evading a Machine Learning-based Intrusion Detection System through Adversarial Perturbations

Torgeir Fladby
Oslo Metropolitan University
Oslo, Norway

Hårek Haugerud
Oslo Metropolitan University
Oslo, Norway

Stefano Nichele
Oslo Metropolitan University
Oslo, Norway

Kyrre Begnum
Oslo Metropolitan University
Oslo, Norway

Anis Yazidi
anisy@oslomet.no
Oslo Metropolitan University
Oslo, Norway

ABSTRACT

Machine-learning based Intrusion Detection and Prevention Systems provide significant value to organizations because they can efficiently detect previously unseen variations of known threats, new threats related to known malware or even zero-day malware, unrelated to any other known threats. However, while such systems prove invaluable to security personnel, researchers have observed that data subject to inspection by behavioral analysis can be perturbed in order to evade detection .

We investigated the use of adversarial techniques for adapting the communication patterns between botnet malware and control unit in order to evaluate the robustness of an existing Network Behavioral Analysis solution. We implemented a packet parser that let us extract and edit certain properties of network flows and automated an approach for conducting a grey-box testing scheme of Stratosphere Linux IPS. As part of our implementation, we provided several techniques for providing perturbation to network flow parameters, including a *Simultaneous Perturbation Stochastic Approximation* method, which was able to produce sufficiently perturbed network flow patterns while adhering to an underlying objective function.

Our results showed that network flow parameters could indeed be perturbed to ultimately enable evasion of intrusion detection based on the detection models that were used with the Intrusion Detection System. Additionally, we demonstrated that it was possible to combine evading detection with techniques for optimization problems that aimed to minimize the magnitude of perturbation to network flows, effectively enabling adaptive network flow behavior.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**;

KEYWORDS

Machine Learning, Intrusion Detection, Adversarial Techniques

ACM Reference Format:
Torgeir Fladby, Hårek Haugerud, Stefano Nichele, Kyrre Begnum, and Anis Yazidi. 2020. Evading a Machine Learning-based Intrusion Detection System through Adversarial Perturbations. In *International Conference on Research in Adaptive and Convergent Systems (RACS '20), October 13–16, 2020, Gwangju, Republic of Korea.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3400286.3418252

RACS '20, October 13–16, 2020, Gwangju, Republic of Korea

ACM ISBN 978-1-4503-8025-6/20/10…$15.00
https://doi.org/10.1145/3400286.3418252

1 INTRODUCTION

The immense growth of the Internet has over the past three decades laid the foundation for massive technological development in almost all modern industries. However, as digital solutions open up for opportunities for citizens and organizations, the increased accessibility of intellectual property introduces an arsenal of new risks. Advanced persistent threats (APTs) are more frequently targeting critical infrastructure such as government systems, power grids, mobile communication systems, critical manufacturing facilities, water supplies and supply chain systems [10]. As businesses and organizations migrate services and infrastructure to the cloud, and more
a responsible overview over data and communications in internal networks becomes increasingly difficult [17]. As a result, companies and nation states are forced to take the risk introduced by increased attack surface, adaptive adversaries and response-driven cyber security into consideration when conducting their daily business.

The efficiency of Intrusion Detection Systems (IDSs) rely on several parameters, including positioning in the network, configuration of preprocessors, event management and more importantly the techniques used for detection. A combination of signature detection, anomaly detection and a variety of Machine Learning(ML)-based approaches are commonly used in modern Intrusion Detection and Prevention (IDPS) [1, 2, 9, 12]. An example of a system which provides ML-based Network Behavioral Analysis (NBA) is Stratosphere Linux IPS (Slips), an open source implementation of an NBA that uses Markov chains to generate models of malicious network traffic over time. Slips provides IDS functionality by comparing these models with live network traffic and determines whether the recorded network flow behavior matches that of a malicious profile.

The paper that this article is based upon aimed to evaluate Slips' behavioral analysis capabilities with respect to its robustness against adaptive network flow behavior, and furthermore observed

the degree of which malicious network traffic had to be altered to in order to evade detection by Slips. The results of this article may incentivize further research in creating even more robust NBA solutions for intrusion detection.

This article will highlight some of the related work which this work is based upon, including a brief description of Slips, a core technology used in this research. The approach to the experiments conducted is then accounted for, along with the idea of manipulating network flows - "flow perturbations" - and the mathematical procedures used in this process. Furthermore, the experiments setup is described before a subset of results and the conclusion is discussed.

2 RELATED WORK

Techniques using machine learning to detect previously unseen malware have been studied extensively throughout the last century; however, it is only in recent years that adversarial machine learning as a tool for evading such systems has gained significant attention. While many researchers attempt to improve the performance and accuracy of their classification algorithms [21], evaluation of machine learning based intrusion detection *robustness* has been studied slightly less. Kos et. al proved in [13] that small, but carefully crafted, perturbations to original input images can mislead a neural network classifier to produce incorrect output. They used adversarial examples to attack generative models, and were able to mislead neural networks trained on MNIST [14], SVHN [18] and CelebA [15] datasets with high confidence by feeding the target models with adversarial examples. The researchers showed how to break the integrity of an ML-based image classifier, but more importantly demonstrated how a generative adversarial model could be used to attack neural networks trained on a range of different datasets.

The advancement of adversarial techniques that use machine learning to perturb input features has seen development in a range of other domains. Rigaki et. al [22] demonstrate the use of such techniques against the IPS domain by showing that it is possible to use Generative Adversarial Networks (GANs) to mimic network traffic, adapt malware communication and ultimately avoid detection; in their case eliminating blockage of C2 network traffic. Papernot et. al [8] showed in 2016 how adversarial sample attacks against malware classifiers could be constructed, and furthermore evaluated how defensive mechanisms could be improved by training malware classifiers using data gathered from adversarial training. In 2017 Papernot et. al expanded on their research by staging an attack against a malware classifier that ultimately reached a misclassification rate of up to 63%. There has also been research on measures to make systems robust in order to counter adversarial machine learning attacks, both against rule-based IDSs [16] [20] and against deep learning models [8] [19] [25]. In [26], Yuan et. al investigate and summarize approaches for generating adversarial examples, applications for adversarial examples and corresponding countermeasures. These ML-based techniques for adversarial examples, robustness improvement and adaptive malware provide a basis for the methods used in this paper.

3 APPROACH

This section provides a high-level overview of the technical implementations that were used to address the problem. The experiments conducted can be generally described as a process feeding Slips with a packet capture which has been manipulated based on a set of parameters. A high-level description of this approach can be seen in Figure 1.

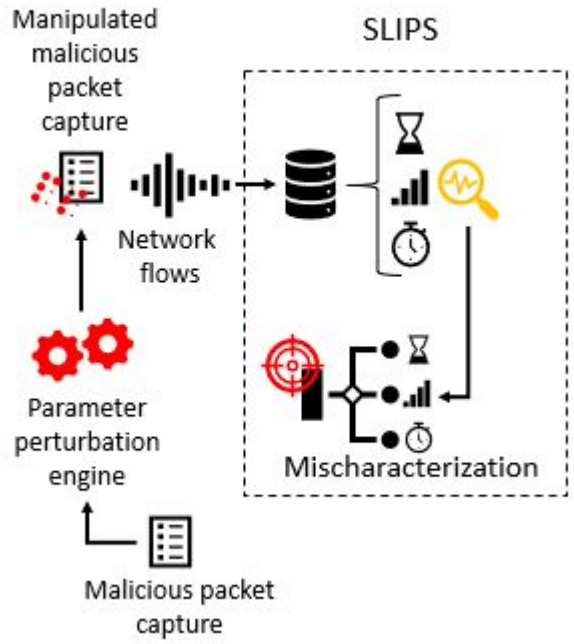

Figure 1: A malicious packet capture is fed into a manipulation scheme which uses perturbation algorithms to change the values of the network flows' behavioral properties.

Network Flow Size	Small			Medium			Large		
Duration	Short	Medium	Long	Short	Medium	Long	Short	Medium	Long
Strong Periodicity	a	b	c	d	e	f	g	h	i
Weak Periodicity	A	B	C	D	E	F	G	H	I
Weak Non-Periodicity	r	s	t	u	v	w	x	y	z
Strong Non-Periodicity	R	S	T	U	V	W	X	Y	Z
No data	1	2	3	4	5	6	7	8	9

Table 1: Character representation of network flow size in bytes and duration of network flow

3.1 Stratosphere Linux IPS

Stratosphere Linux IPS (Slips) is an open-source network flow analysis tool, developed by researchers from Czech Technical University, led by Sebastien Garcia et. al [7]. The tool takes as input network flows from a *ra client* and feed them to a set of network behavioral models and detection algorithms. More specifically, the Slips program models network flows as Markov chains by consuming parameters such as size, duration and periodicity of each flow, as represented in Figure 2. Based on the chain of states generated, each flow is assigned a symbol that characterizes the behavior of the connection in that specific state, chained together into a single characterization [11]. Character representations used to create chains of states can be viewed in Table 1.

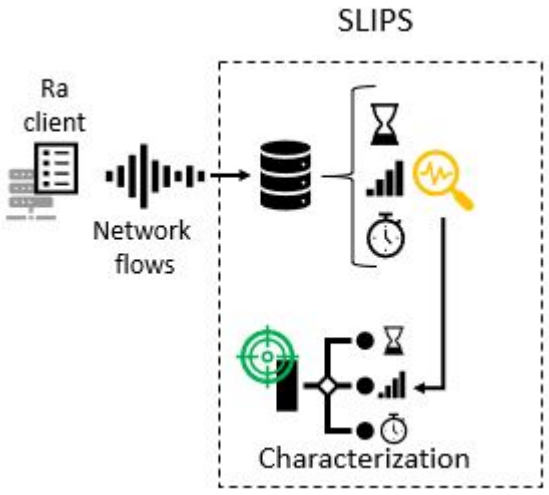

Figure 2: Duration, size and periodicity of network flows are analyzed by Stratosphere IPS to create characterizations of network traffic.

3.2 Flow perturbation

In order to test whether perturbation of network flow parameters could assist in evading behavioral analysis intrusion detection, a set of initial experiments were conducted on existing malware capture datasets. It was desirable to see if one could reliably alter Slips' representation of network connections' state by altering the properties of packet capture files that has already been modeled and labeled by Slips. Specifically, these experiments aimed to alter properties that affect periodicity in network flows; duration of connections, network flows and time between network flows are parameters that affect this property [6]. The size of network flows also impact the state of network flows and the probability that it matches a model labeled as malicious - hence, this parameter was also included as part of the perturbation scheme. The size of network flows could be used as a target for a loss function in an optimization scheme to e.g. minimize the degree of change in amount of bytes transmitted, as a way of closely mimicking the malicious network traffic's actual content. It is worth noting that the altered network packets should not be subject to data loss, as that would conceptually strip the malicious capabilities of the captured network packets. If network traffic which has previously been observed, labeled and detected by Slips as malicious can be altered to evade detection, without losing their malicious content, conducting other experiments that perturb network flow properties would be justified.

3.3 Procedures for parameter perturbation

After the functionality for perturbing network flow parameters was implemented, we wanted to evaluate to what extent the manipulation of these parameters affected the detection rate of malware. Since Slips computes behavioral patterns for network connections based on network flow size, network flow duration and *periodicity* [6], we wanted to cause some manual disturbance of parameters that affect these properties. If manual perturbations affected Slips' detection rate on a given malware capture, we wanted to repeat the experiment using more scientific approaches where the goal was to minimize the perturbations' negative impact on network connection delays. Finally, if we to some extent were able to minimize the impact on connection delays, we wanted to adopt a type of machine learning algorithm to optimize the parameters used with the given algorithms.

An important aspect when generating random perturbations, and in particular for problems that utilize Stochastic Optimization, is the *objective function*. Since we mainly focused on perturbing the periodicity of network flows, and therefore did not perturb the byte-size of network flows, perturbation to that variable does not really impact our resulting packet capture in any way. However, we could still use this variable in our objective function, whether the goal was to reduce the amount of time used by malicious connections, or to make it as similar to the original traffic as possible. Moreover, we could use the information gained by the objective function as feedback to other optimization problems, for instance as a means to generate parameters for a perturbation algorithm. To increase throughput, we could define our objective function as $\hat{y}$,

$$\hat{y_i} = \underset{D}{\operatorname{argmin}} \frac{flow_i^{size}}{flow_i^{D}}, \tag{1}$$

where D is the product of the duration of the i_{th} network flow and the time until its next network flow.

In scenarios where it is desirable to minimize the difference in throughput, the objective function may for instance minimize the euclidean distance between the perturbed sample and the original input. The objective function $\hat{y}$ could then be denoted as the 2-norm distance between the two samples x and y, where x is the observed data and y is its perturbed version:

$$\hat{y} = (\sum_{i=1}^{n} \|x_i - y_i\|^2)^{1/2}. \tag{2}$$

Depending on what problem is to be solved, however, it is often hard to obtain a direct gradient of the objective function, particularly if one has continuous or generative data. In such cases it could be a good idea to rely on a technique that does not require measurements of the direct gradient of the objective function.

Adaptive Step-Size Random Search Algorithm

The ASRS Algorithm [23] was implemented to take as input a set of boundaries that defined the maximum amount of distance each step could take. Additional inputs were perturbation factors for each parameter subject to perturbation, as well as one factor for small step sizes and one factor for large step sizes. The ASRS algorithm was implemented to trial a large step size for each iteration and adopt the larger step size if it yielded a better result. This algorithm was implemented according to the description in [23] and modified to include the perturbation of three parameters.

Simultaneous Perturbation Stochastic Approximation

Simultaneous Perturbation Stochastic Approximation (SPSA) is an algorithm used to solve challenging optimization problems where it is difficult or impossible to obtain a gradient of the objective function [24]. In our case, it was not possible to know the gradient of a certain configuration because we did not know whether the flow would be classified as malicious or not at the time of perturbation. SPSA, in contrast to optimization algorithms that work with discrete noise-free data, relies on two separate measurements of the objective function to obtain an approximation of the gradient, regardless of the number of parameters being optimized. This "two-step", *simultaneous* approach is desirable for solving our optimization problem because the data we attempt to provide perturbations for, while trying to reach a particular objective for each iteration, is both continuous and noisy. SPSA is applicable to a variety of problems within engineering and social sciences, but must be adapted to fit each use case. We chose to implement SPSA because of its versatility with respect to different optimization problems and objective functions, such that we may extend functionality at a later point if desirable.

The goal of SPSA in our case was to minimize the loss function $\hat{y}(\theta)$, where the loss function is a scalar-valued measurement of performance, and *theta* is the vector of parameters to be perturbed. SPSA starts by iterating from an initial guess of *theta*, a vector which in our case is based on measurements of a network flow. We can then obtain measurements $\bar{y}(\theta)$ of the loss function $\hat{y}(\theta)$ by adding noise to the initial loss function:

$$\bar{y}(\theta) = \hat{y}(\theta) + \zeta, \tag{3}$$

where ζ is the added noise. When we have exact measurements, however, adding noise is not necessarily desirable. We implemented support for two different objective functions, namely *throughput maximization* and *Euclidean Distance minimization* and slightly increased the noisiness of the value by adding a random variable chosen from a gaussian distribution of 0 ± 0.1 to θ. The SPSA algorithm was implemented by completing the following steps:

We selected initial values for α, γ, c, a, n and the vector β as 0.602, 0.101, 1, 1, 1000 and [0.1, 0.9], respectively. Then, we defined two iterators, or *gain sequences*, as

$$a_i = \frac{a}{(\beta[1] \times (i+1))^{\alpha}}, \text{ for } i \in \{0, ..., n\} \tag{4}$$

and

$$c_i = \frac{c}{(\beta[0] \times (i+1) * 0.5)^{\gamma}}, \text{ for } i \in \{0, ..., n\}. \tag{5}$$

The gain sequences were created so that they would not produce values of $\hat{\theta}$ with excessively large magnitude of perturbation.

Furthermore, we generated an initial *Simultaneous Perturbation Vector* δ_0 by selecting one Bernoulli-value for each parameter subject to perturbation, such that $\delta_0 = [\pm 1, \pm 1, \pm 1]$. Bernoulli-values were selected due to the requirements stated in [24] Section III. A. We then started iterating over SPSA with n iterations. For each iteration k, we computed the following [24]:

- **Two objective function evaluations**: Two measurements of the loss function $\hat{y}(\theta)$ based on the simultaneous perturbation around $\hat{\theta_k}$. The two perturbations were calculated as $\hat{\theta}_{k1} = y(\hat{\theta}_k + c_k \times \delta_k)$ and $\hat{\theta}_{k2} = y(\hat{\theta}_k - c_k \times \delta_k)$, where c_k is the gain sequence as defined in Formula 9.
- **Simultaneous Perturbation Gradient Approximation**: An approximation to the unknown gradient $g(\hat{\theta_k})$ as

$$g_k(\hat{\theta_k}) = \frac{\hat{\theta}_{k1} - \hat{\theta}_{k2}}{2 \times c_k} \tag{6}$$

- **Estimate update**: The next estimate of $\hat{\theta}$, denoted as $\hat{\theta}_{k+1}$ was computed as the difference between $\hat{\theta_k}$ and $g_k(\hat{\theta_k}) \times a_k$, where a_k is the k_{th} element of our gain sequence a:

$$\hat{\theta}_{k+1} = \hat{\theta_k} - g_k(\hat{\theta_k}) \times a_k \tag{7}$$

- **Termination**: SPSA terminates if there is little change over several successive iterates or the maximum number of iterations has been reached. If these conditions are not met, k is incremented and a new iteration is initiated.

Guidelines for selection of gain sequences in [24] were used in an attempt to select viable parameters for our implementation of the SPSA algorithm.

4 EXPERIMENTS SETUP

The experiments described in this section aimed to test the feasibility of behavioral analysis systems in a semi-realistic environment that had already been compromised by an attacker. The attack scenario could be described as the Command and Control(C&C) stage of a red team exercise [3], where it is assumed that a malicious entity already has compromised at least one host within a victim network, and that privileges are escalated to the extent that attackers are able to install arbitrary software on the host. During the C&C stage of a red team exercise, it is detrimental that penetration testers do not give away that they have compromised infrastructure on target systems. For that reason, when a communication channel with the control server is to be established, it is desirable that the malware's communication channel remains hidden from any IPS agents analyzing traffic on internal or external network endpoints.

Technical implementation. To reach the ultimate objective of confirming whether perturbation to network flow parameters affected the probability that a malicious capture packet file was misclassified as benign, some infrastructure and software had to be implemented:

- (1) A container configuration that installs *Stratosphere Linux IPS*, *argus* and *ra* alongside all relevant dependencies to enable execution of experiments on a platform that is trivial to deploy on any operating system.
- (2) Software taking as input a packet capture file and a set of *perturbation parameters*. All relevant network flows had to be identified, and properties such as *duration*, *size of network flow* and *time between current flow and next flow* had to be computed. When all network flows and other connections had been identified and grouped, the argument perturbation parameters should then be applied to the relevant network flows. The program must return the same connections, in the

same order, but with timestamps that are modified according to the input perturbation parameters.
- (3) A script taking as input the packet capture file generated in (2) to create a network flow file compatible with analysis in Slips.
- (4) A module providing perturbation parameters to the program that changes network flow properties of the selected pcap-files. A set of techniques for computing perturbations were features of this module.
- (5) An implementation of a Learning Automata that enabled a large set of iterations to be computed over different configuration options of the algorithm in (4). The goal of the Learning Automata was to find an optimal configuration of parameters provided to the selected algorithm. The configuration could, for instance, minimize the impact that perturbation has to network flow durations, while ensuring that the entire network flow remained undetected by Slips. For each iteration, the probability that the Learning Automata selected a feasible solution, should be increased.

5 RESULTS

This section provides a short description of the experiments that were conducted as part of this paper and the results produced by each experiment. The experiments were designed to provide data that reflected the performance of Slips, or to serve as building blocks that justify other experiments conducted in the paper that this article is based upon. To simulate malicious network flows, the researchers used publically available datasets from Czech Technical University (CTU)[4]. While multiple datasets were used in the initial paper, this article presents results from manipulating the properties of network flows in the dataset CTU-108-1. Recall that the term *magnitude* in this context is used to describe the degree of which a parameter has been perturbed; lower magnitude equals lower degree of change.

5.1 Adaptive Step-size Random Search

Since ASRS is threshold-based, we wanted to run a larger amount of iterations using the same set of parameters as in Iteration 2 of 5.2.1 in [5], with the goal of observing whether a slightly more intelligent way of selecting perturbation parameters could result in a smaller magnitude of perturbation. Furthermore, we decided to run the SPSA algorithm with the Cridex dataset, such that we could look at flows having some varying flow characteristics.

With respect to magnitude per flow, ASRS performed a lot better than simple random variables. When using the same parameters as in Iteration 2 of 5.2.1 [5], the average magnitude was *1.96*, compared to the much higher 2.78 using random thresholds.

By observing the magnitudes of perturbation per flow in Figure 3, we spot one outlier network flow, with roughly 5 times the average value:

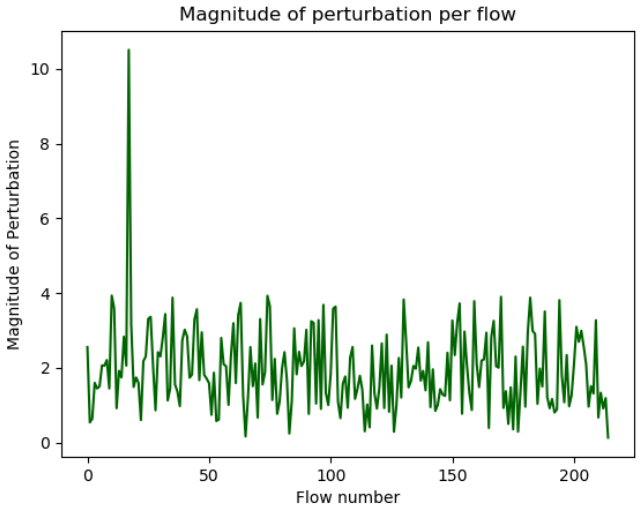

Figure 3: Magnitude of perturbation per flow for CTU108-1 using ASRS

Because ASRS bases perturbation on initial values and thresholds, the outlier could be due a large observational value.

The mean standard deviation of magnitude for CTU108-1 using ASRS was measured to *1.60*, while the mean variance was measured to *2.75*.

If our goal is to minimize magnitude while remaining undetected, ASRS provides us with a consistent improvement in performance because it yields a smaller magnitude of perturbation while remaining undetected in all cases.

Iter	Mean Magnitude	Detected
0	0	True
1	20.68	False
2	1.96	False
3	0.41	False

Table 2: Magnitudes of perturbations with varying thresholds on CTU108-1 using ASRS.

5.2 Simultaneous Perturbation Stochastic Approximation

This experiment did not require any bounds or thresholds in order to run the perturbation algorithm. Using the static default parameters for the vector β as $[0.1, 0.9]$ to generate gain sequences, we ran a total of 50 iterations. We used the objective function as stated in Formula 6 for maximizing throughput. All solutions generated by the SPSA were *feasible*, i.e. not detected as Slips as malicious. The mean of the mean magnitudes for all trials was *4.18*, indicating an increase compared to the ASRS algorithm. The reason for this could be that we used an objective function that aimed to maximize throughput by minimizing duration of flows and the time between flows.

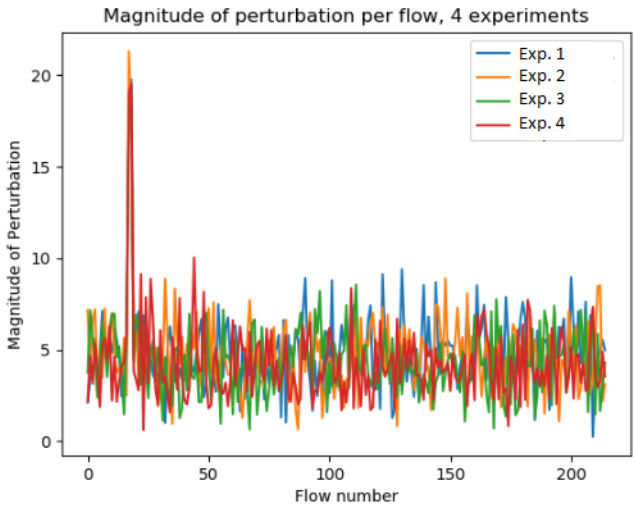

Figure 4: Magnitude of perturbation per flow for 4 iterations on CTU108-1 using SPSA with static gain sequence parameters.

We can observe from the graph in Figure 4 not only that the magnitudes are more severe than when using the ASRS algorithm, but that the variance in magnitude is also significantly increased. We measured the mean standard deviation and variance for all 50 iterations of this experiment to *4.78* and *2.18*, respectively. This could imply that the algorithm's gain sequence parameters are not sufficiently tuned to work with our solution.

6 CONCLUSION

The results of this article may contribute to increasing awareness around the importance of taking adaptive behavior into consideration when using NBA as a component in IDSs. As threats often emerge faster than their security measure counterparts, one should in critical environments assume that malicious actors and their software are trivially able to adapt to detection systems. Information Security specialists must ensure that their security software, and the models they depend on, are at all times up to date and on-par with the current highest standard. While NBA can be a positive supplement to existing IDS solutions, security professionals should be careful not to blindly trust existing models.

We set out to answer how the behavior of malicious network traffic could be altered to evade detection by a behavioral analysis tool. While we did not prove that evasion was possible for a live solution, we did conceptually show that by creating a packet manipulation scheme supporting perturbations to network flow parameters, we may perturb network flow patterns to effectively confuse NBA intrusion detection. Furthermore, we explored and implemented different search techniques that provided perturbed versions of an initial set of network flow parameters. Finally, we demonstrated how a simple reinforcement-based ML method could be used as a tool to provide optimal parameters for the perturbation algorithms that were implemented. While the experiments were not comprehensive enough to provide directly applicable adversarial techniques, the work incentivizes further research in the areas of ML-based intrusion detection and evasion.

REFERENCES

[1] Aldweesh, A., Derhab, A., and Emam, A. Z. Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. *Knowledge-Based Systems 189* (2020), 105124.

[2] Amoli, P. V., Hamalainen, T., David, G., Zolotukhin, M., and Mirzamohammad, M. Unsupervised network intrusion detection systems for zero-day fast-spreading attacks and botnets. *JDCTA (International Journal of Digital Content Technology and its Applications 10*, 2 (2016), 1–13.

[3] Consulting, R. T. S. Red teaming methodology, 2018.

[4] CTU. CTU Public Datasets. https://mcfp.felk.cvut.cz/publicDatasets/, 2020. [Online; accessed 17-September-2018].

[5] Fladby, T. F. Adaptive network flow parameters for stealthy botnet behavior. Master's thesis, 2018.

[6] Garcia, S. Modelling the network behaviour of malware to block malicious patterns. the stratosphere project: a behavioural ips. *Virus Bulletin, number September* (2015), 1–8.

[7] Garcia, S., and Pechoucek, M. Detecting the behavioral relationships of malware connections. In *Proceedings of the 1st International Workshop on AI for Privacy and Security* (2016), ACM, p. 8.

[8] Grosse, K., Papernot, N., Manoharan, P., Backes, M., and McDaniel, P. D. Adversarial perturbations against deep neural networks for malware classification. *CoRR abs/1606.04435* (2016).

[9] Hajiheidari, S., Wakil, K., Badri, M., and Navimipour, N. J. Intrusion detection systems in the internet of things: A comprehensive investigation. *Computer Networks 160* (2019), 165–191.

[10] Inc., C. S. Cisco 2018 Annual Cyber Security Report. https://www.cisco.com/c/en/us/products/security/security-reports.html, 2018. [PDF; accessed 17-September-2018].

[11] IPS, S. Stf. https://www.stratosphereips.org/stratosphere-testing-framework, 2018.

[12] Khraisat, A., Gondal, I., Vamplew, P., and Kamruzzaman, J. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity 2*, 1 (2019), 20.

[13] Kos, J., Fischer, I., and Song, D. Adversarial examples for generative models. *arXiv preprint arXiv:1702.06832* (2017).

[14] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[15] Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 3730–3738.

[16] Lowd, D., and Meek, C. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* (2005), ACM, pp. 641–647.

[17] McAfee. Navigating a Cloud Sky. https://www.mcafee.com/enterprise/en-us/assets/reports/restricted/rp-navigating-cloudy-sky.pdf, 2018. [PDF; accessed 11-October-2018].

[18] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning* (2011), vol. 2011, p. 5.

[19] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on* (2016), IEEE, pp. 372–387.

[20] Perdisci, R., Gu, G., and Lee, W. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *Data Mining, 2006. ICDM'06. Sixth International Conference on* (2006), IEEE, pp. 488–498.

[21] Rieck, K., Holz, T., Willems, C., Düssel, P., and Laskov, P. Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2008), Springer, pp. 108–125.

[22] Rigaki, M., and Garcia, S. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In *2018 IEEE Security and Privacy Workshops (SPW)* (2018), IEEE, pp. 70–75.

[23] Schumer, M., and Steiglitz, K. Adaptive step size random search. *IEEE Transactions on Automatic Control 13*, 3 (1968), 270–276.

[24] Spall, J. C. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronic Systems 34*, 3 (July 1998), 817–823.

[25] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[26] Yuan, X., He, P., Zhu, Q., Bhat, R. R., and Li, X. Adversarial examples: Attacks and defenses for deep learning. *arXiv preprint arXiv:1712.07107* (2017).