HoneyGAN Pots: A Deep Learning Approach for Generating Honeypots

Ryan Gabrys, Daniel Silva and Mark Bilinski

Naval Information Warfare Center Pacific {ryan.c.gabrys, daniel.silva61, mark.bilinski}.civ@us.navy.mil

Abstract

This paper investigates the feasibility and effectiveness of employing Generative Adversarial Networks (GANs) for the generation of decoy configurations in the field of cyber defense. The utilization of honeypots has been extensively studied in the past; however, selecting appropriate decoy configurations for a given cyber scenario (and subsequently retrieving/generating them) remain open challenges. Existing approaches often rely on maintaining lists of configurations or storing collections of pre-configured images, lacking adaptability and efficiency. In this pioneering study, we present a novel approach that leverages GANs' learning capabilities to tackle these challenges. To the best of our knowledge, no prior attempts have been made to utilize GANs specifically for generating decoy configurations. Our research aims to address this gap and provide cyber defenders with a powerful tool to bolster their network defenses.

1 Introduction

The field of cybersecurity constantly faces the challenge of defending networks and systems against malicious attacks. One effective approach to deceive adversaries and gather intelligence about their tactics is through the use of decoy systems, which are commonly known as honeypots [1, 2]. By strategically deploying honeypots at different stages of the cyber kill chain, organizations can gain valuable insights into the attacker's methods, motives, and vulnerabilities [3]. Honeypots can provide early warning signs, capture attack tools or malware samples, and gather valuable threat intelligence that can enhance overall security.

Honeypots are typically categorized as being either high or low-interaction depending on their level of sophistication. Low-interaction honeypots typically target an attacker at the earlier phases of the cyber kill chain, such as reconnaissance, whereas the high-interaction honeypots aim to disrupt potential attackers at later phases, such as delivery and lateral movement [4]. Although the methods described in this work can be applied to generate either low-interaction or high-interaction honeypots, our primary focus in this work is on the design and generation of realistic-looking low-interaction

honeypots that can be used in the context of a cyber defense strategy to detect, deter and/or delay potential attackers.

1.1 Our Approach

Our proposed method utilizes an adaptable infrastructure that only requires two essential pieces of information: the services available on each open port and the operating system details. We note that such infrastructures have existed for some time [5],[6] but one of the challenges that still exists is determining which decoy configurations to choose from, as well as how the configurations should be generated. Some naive approaches involve maintaining a list of possible device configurations or in some cases even storing collections of pre-configured images.

Our approach harnesses the capabilities of GANs to learn the distribution of network device configurations using real data. This approach offers remarkable flexibility, as cyber defenders no longer need to maintain collections of potential configurations. Instead, our GAN-powered system dynamically generates realistic-looking decoy configurations based on specified requirements. Particularly in scenarios where a large number of diverse and authentic-looking decoys are desired, this methodology has the potential to offer enormous benefit.

The main objective of this paper is to explore the feasibility and effectiveness of using GANs for generating decoy configurations in cyber defense, which, to the best of the author's knowledge, has not been attempted before. This work represents a first effort where future works will incorporate additional information about the network environment that can be used to better inform the design of honeypots. By leveraging the learning capabilities of GANs, we aim to provide cyber defenders with a powerful tool to enhance their network defenses.

1.2 Contributions

Our contributions are the following:

- 1. Using a simple data model, we show that a GAN can generate high-quality replicas of actual network device configurations using the concepts of precision and recall from [7].
- 2. For the setup where a cyber defender wishes to generate certain types of decoys, we design two condi-

- tional GANs that generate network device configurations based upon operating system or service type.
- 3. We demonstrate that the resulting decoys created from our generative models are robust to current honeypot detection systems.

1.3 Outline

Section 2 reviews the concepts of Generative Adversarial Networks (GANs) as well as the use of honeypots for network defense. In Section 3, we present the models for our GAN architecture and Section 4 reviews our simple data model. In Section 5, we measure both the diversity and accuracy of the GANs that generated and trained using real-world data. Finally, Section 6 concludes the paper.

2 Background and Related Work

2.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) have gained significant attention due to their ability to generate synthetic data simulating realistic media such as images, text, audio and videos. GANs were introduced in 2014 by Ian Goodfellow [8] and have since sparked a revolution in the field of generative modeling. Unlike traditional generative models which are typically trained by maximizing a log likelihood, GANs possess a unique architectural setup consisting of two neural networks: the generator and the discriminator.

The generator network takes random noise as input and generates synthetic samples, such as images, text, or even audio. The discriminator network, on the other hand, receives both real and generated samples and tries to distinguish between them. The two networks are trained together in a competitive setting, constantly improving and challenging each other's performance. More specifically, the discriminator, denoted as D, and generator, denoted as G, play the following two-player min-max game with value function V(G,D):

$$\min_{G} \max_{D} V(D, G) = E_{x \sim p_r(x)} \left[\log D(x) \right] + E_{z \sim p_g(z)} \left[\log \left(1 - D(G(z)) \right) \right],$$
(1)

where p_g represents the generator's distribution and p_z represents a prior on input noise variables. The generator aims to generate samples that are indistinguishable from real data, while the discriminator strives to correctly classify between real and fake samples. As training progresses, the generator learns to produce increasingly realistic samples by receiving feedback from the discriminator. The discriminator, in turn, becomes more adept at distinguishing between real and generated data.

The applications of GANs span a wide range of domains. In computer vision, GANs have been used for image synthesis, style transfer, image-to-image translation, and superresolution [8], [9]. They have also been employed in generating realistic deepfake videos and enhancing image generation in areas like fashion, art, and design [10]. In natural language processing, GANs have been utilized for text generation, language translation, and dialogue systems. GANs have even found applications in healthcare, where they have been used

for generating synthetic medical images, augmenting data for training medical models, and drug discovery [11].

Nowadays, GANs are broadly studied and applied through academic and industrial research in different domains beyond media (e.g., natural language processing, medicine, electronics, networking, and cybersecurity) [12], [13]. After a GAN has been trained, its generator can produce as many synthetic examples as necessary, providing an efficient mechanism for solving the problem of lack of labelled data sets and potential privacy restrictions.

2.2 Cyber Defensive Deception

Honeypots are decoy systems or resources intentionally designed to attract and deceive malicious actors, allowing cybersecurity professionals to study their behavior, gather intelligence, and enhance their defensive strategies. Dating back to at least as far as the early 1980s, researchers have considered strategies involving staging imaginary computer environments (honeypots) to lure attackers and reveal their objectives [1].

The Honeynet Project, initiated by Lance Spitzner and a group of security professionals, was one of the pioneering efforts in developing and deploying honeypots. The project aimed to capture and analyze the tactics, techniques, and tools used by hackers. These can be used to provide valuable insights into honeypot deployment strategies and the analysis of captured attack data [14].

Later works considered various methods used by attackers to detect and evade honeypots. In order to address many of the challenges, in [15] the authors emphasized the importance of continuous monitoring and adaptation to stay ahead of sophisticated adversaries. The work in [16] focuses on using honeypots as a means to gather intelligence on botnets, which are networks of compromised computers controlled by malicious actors. It demonstrates how honeypots can be used to detect, monitor, and analyze botnet activities, providing valuable information for improving network defenses and disrupting cybercriminal operations.

In [17], the use of honeypots to analyze and understand Advanced Persistent Threat (APT) campaigns is considered. APTs are stealthy and prolonged cyber attacks typically orchestrated by nation-state actors or advanced criminal organizations. One of the many advantages of employing honeypots in these environments is that they can provide insights into attacker tactics, helping organizations improve their defense mechanisms against such threats [18].

3 Experimental Setup and GAN Architecture

We instantiated our GANs with the following properties:

- **Batch Size**: This refers to the number of machine configurations that are used for training at each iteration of the optimizer. We used a batch size of 64.
- Number of steps: Each step involves a single generator iteration along with 3 discriminator training iterations. Our unconditional GAN was trained for 11844 steps whereas our conditional O/S GAN and conditional DT GAN were trained for 5922 and 23688 steps, respectively.

- **Gradient Penalty Coefficient**: This represents the loss term that keeps the L2 norm of the discriminator gradients close to 1. We set this parameter to 10.
- · Adam optimizer hyper parameters:
 - Learning Rate: Controls how quickly parameters in the model are updated.
 - Coefficient β₁, β₂: Manages the decay rates of the moving average of the gradient and the squared gradient.

For our setup, we set the learning rate to be 0.0002 and the values of β_1 , β_2 to be 0.5, 0.9 respectively.

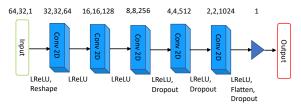


Figure 1: Discriminator Model

Our architecture consists of two neural networks where, as is illustrated in Figure 1, the discriminator network is using fractionally strided convolutions and the generator network is performing a deconvolution procedure. The discriminator model has a normal convolution layer followed by four convolution layers using a stride of 2 and a kernel of size 5 to downsample the input. The final layer of the discriminator is a dense layer that provides a single output.

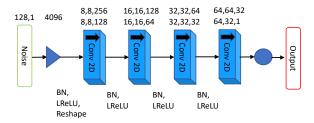


Figure 2: Generator Model

The generator model consists of four blocks where each block performs upsampling followed by a convolution layer that has a kernel of size 3 with a stride of size 1. After these four blocks, the data is passed through a final activation layer. Both models were trained according using the Wassterstein GAN with Gradient Penalty method [19].

4 Dataset and Model

For the purpose of evaluating our proposed approach, we trained the GAN described in the previous section using a set of device configurations that were obtained using the Shodan search engine. Using Shodan, we were able to extract the configurations of 378973 internet-connected devices. Each configuration consisted of the set of open ports, the service running on each of those ports, along with any Common Platform

Enumeration (CPE) data that is associated with the particular service. For the purposes of illustrating this data, Figure 3 shows an example JSON document representing the device configuration for a Windows Server running three services.

Figure 3: JSON representation of Windows Server device

As can be seen in Figures 1 and 4, each device configuration is represented as a 64x32 object. The first two columns (which contains 64 rows) encodes the information associated with the operating system and build associated with the particular device. The remaining 30 columns contain information pertaining to the services. Under this model, and as is being illustrated in Figure 4, each column will contain exactly two ones.¹

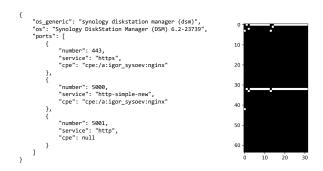
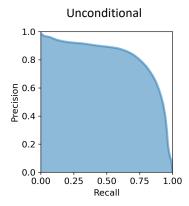
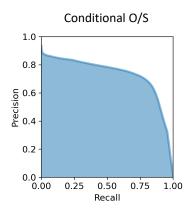


Figure 4: Sample Data Model for Machine Configuration

Each of the next 30 columns (after the first two) represent a particular port. The assignment of columns to ports as well as the methodology in selecting these 30 ports, is described in more detail in Appendix A. Each of these 30 columns contains exactly two ones where the one in the first 32 rows indicates the service that is running and the second one present in rows 33-64 indicates the CPE associated with the service.

¹The ones in each column of Figure 4 are colored white.





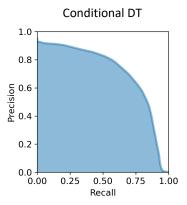


Figure 5: PRD Plots for Achievable (α, β) Pairs Across Different GANs

5 Evaluation

In order to evaluate the performance of our GAN, we leveraged the notion of precision and recall originally introduced in [7]. Conceptually, *precision* measures how accurately the generated samples are to the true reference distribution whereas *recall* attempts to capture how well the true sample distribution is "covered" by the generated distribution. More precisely, suppose that P_R represents a reference distribution and P_G represents the generated (or learned) distribution. Then for $\alpha, \beta \in [0,1]$, we say that the probability distribution P_G has precision α and recall β with respect to P_R if there exists distributions $\mu, \nu_{P_R}, \nu_{P_G}$ if

$$P_R = \beta \mu + (1 - \beta)\nu_{P_R}$$

$$P_G = \alpha \mu + (1 - \alpha)\nu_{P_G}.$$

As can be seen from the expressions above, the parameter $1-\beta$ is the recall loss whereas $1-\alpha$ is the loss due to precision. The set of attainable pairs of precision and recall of a distribution P_G with respect to P_R consists of all (α,β) pairs that are achievable.

Figure 5 is illustrating the PRD plots for the three types of GANs that were developed, which we refer to as our unconditional GAN, conditional O/S GAN, and conditional Device Type (DT) GAN. The left plot of Figure 5 displays the (α,β) pairs for our unconditional (unlabeled) GAN. The unconditional GAN was trained for 11844 steps after which time the performance of the GAN did not improve. In general, and as can be seen above, this GAN typically exhibited the highest achievable (α,β) pairs. As a concrete example, if we fix the level of recall to be 0.75, we see that a precision of $\alpha>0.8$ is possible.

Despite its high levels of precision and recall, our unconditional GAN has the drawback that a human cyber defender has no control over the types of samples that it outputs. In order to overcome this potential issue, we investigated the performance of two additional conditional GANs:

- Conditional O/S GAN: This model was trained using labels that reflected the operating system.
- Conditional Device Type (DT) GAN: This model was trained using labels reflecting the function or type of the

underlying device. Details of how these device type categories were assigned to devices can be found in Appendix B.

For the conditional O/S GAN, each device was assigned exactly one of the following labels reflecting its O/S: Mikrotik Routeros, Windows Server, Windows, Synology Diskstation Manager, Sonicwall Sonico, Linux, Ubuntu, Debian, Synology Router Manager, or QTS. The exact distribution of these labels to our data set is included in Appendix A.

For the conditional DT GAN, each device was assigned one or more of the following labels: file sharing, remote access, webserver, mailserver, database, dns, vpn, router, management. The procedure that was followed in assigning devices to labels under the DT GAN architecture is described in more detail in Appendix B. The performance of our conditional GANs were evaluated similarly by computing the respective PRD plots. Recall that the conditional DT GAN was trained for 23688 steps whereas the conditional O/S GAN was trained for 5922 steps.²

As can be seen from Figure 5, although the conditional GANs afford a cyber defender greater control over the output configuration (or samples), the penalty for such control are lower levels of achievable (α,β) pairs. For example, the conditional O/S GAN never achieves a precision above 0.80 when the recall is above 0.55 and similarly the conditional DT GAN does not achieve a precision above 0.75 when the recall is held at 0.75. Interestingly, the conditional DT had higher values of precision when the recall was allowed to be smaller, but the conditional O/S exhibited larger levels of precision for larger recall values.

In order to better understand the performance of our GAN architecture when provided a smaller number of generated samples, we considered the quality of the unconditional GAN as a function of the number of outputs. In particular, Table 1 is displaying the number of distinct port/service/os combinations that are generated from our GAN as a function of the number of samples requested. For these outputs, we sampled with replacement from the set of real configurations that

²Similar to the unconditional GAN, training the GANs for more steps did not improve the performance.

were retrieved from Shodan as well as a set of configurations that were generated by our unconditional GAN. For example, the first row of the table is showing that among 500 randomly selected real configurations, there were 406 whose port/service/os information was unique. Then, after our generator produced an equivalent 500 samples, there were 295 unique samples and 247 matched a real configuration. As the generator could generate a given sample multiple times, some of the matches may be repeats of the same configuration, which is why in many cases there are a larger number of matches than unique configurations and is likely a result of over-fitting. As expected, as the number samples increases, the diversity of outputs from our GAN also decreases. Similar trends were observed with respect to the other two GANs and this data is included in Appendix C.

Samples	Real, Unique	Gen, Unique	Gen, Match
500	406	295	247
1000	720	465	536
1500	1006	623	750
2000	1319	702	999
2500	1602	837	1280
3000	1881	916	1470
3500	2109	969	1812
4000	2351	1075	1951
4500	2603	1105	2337
5000	2844	1209	2514

Table 1: Generator Output as Function of Sample Size for Unconditional GAN

In addition to considering how the samples generated from each of our three GANs compared to the real set, we also leveraged HoneyD to generate actual low-interaction decoys using port/service/OS data from our unconditional GAN. Using these decoys, we then evaluated the quality of the resulting decoys using the Checkpot [20] honeypot checker utility, which was developed for the purposes of helping security researchers check that their honeypots are properly set up in such a manner as to attract "high-quality traffic." Towards this end, the Checkpot utility works by taking the network address of a honeypot as input and then outputting a number, known as a Karma value, that indicates the quality of the honeypot specified as input. We created 5000 low-interaction honeypots with HoneyD using 5000 randomly generated configurations from our unconditional GAN. The average Karma value for the resulting collection of honeypots was 491.584. As a baseline, we also evaluated the Karma value of another publicly available low-interaction honeypot, which according to Checkpot, had a Karma value of only 60 [21]. In comparison, we generated HoneyD decoys using 5000 randomly selected real device configurations and found the average Karma value to be 504.91, indicating that the quality of honeypots is nearly the same as if we had stored hundreds of thousands of actual machine configurations.

6 Conclusion

In this paper, we have considered the feasibility of applying deep learning techniques to the problem of generating realistic-looking decoy configurations. We have shown that a GAN can generate high-quality replicas of actual network device configurations using precision and recall metrics, and that this result can be achieved through the use of a relatively simple data model. Additionally, two conditional GANs were designed to generate network device configurations based on either operating system or service type. The resulting decoys produced by these generative models were then demonstrated to be resilient against current honeypot detection systems.

Despite the progress made in our research, it is important to acknowledge the limitations or potential drawbacks of the approach considered. One of the potential shortcomings of our approach is the relatively sparse data representation that was chosen. Future works will consider the use of embeddings in the hopes of yielding a more efficient representation. Another aspect of our design was that duplicate samples were used to train both the generator and discriminator models, which likely led to over-fitting, and potentially negatively impacted the diversity of our resulting model.

Moving forward, there are several promising avenues for further research in this field. Building upon the findings and insights gained from our study, future investigations could focus on the following areas: (a) Development of an efficient embedding for representing machine configurations along with the configurations of other machines in the same network environment, (b) Design of a decoy generation system whose output depends on other factors in the environment such as the configurations of other machines in the subnet or data about the presence or behaviors of suspected attackers, (c) Development of generative models for producing alternative types of honeytokens to aid in deception and network defense.

A Representation of Port/Service Information in Data Model

As mentioned in Section 4, our two-dimensional data representation uniquely associates columns $2, 3, \ldots, 31$ to represent the following ports:

 $21, 22, 23, 25, 53, 80, 110, 123, 135, 137, 139, 161, 443, 445, \\1433, 1701, 1723, 2000, 3306, 3389, 4433, 5000, 5001, 5985, \\8080, 8081, 8291, 8443, 8728, 9100.$

These ports represent the most frequently occurring open ports that were found across the set of 378973 Shodan device configurations that were used to train the models described in Section 3. On average, this data representation captured over 85% of the services running on each device. Table 2 is displaying the number of configurations in our dataset for each of the 9 possible types of operating systems.

B Device Types in Conditional DT GAN

In order to allow a user to specify the type(s) of device generated by our conditional DT GAN, we assigned one or more labels to each of the device configurations used during the

O/S Label	Count
MikroTik Routeros	119478
Windows Server	104584
Windows	50391
DiskStation Manager	49609
SonicOS	25489
Linux	9781
Ubuntu	8960
Synology Router Manager	3616
Debian	5391
QTS	1674

Table 2: Labeling Operating Systems

training process described in Section 3. The logic behind this assignment is being illustrated in Table 2 below. For each device, we labeled the device to be of the type listed in the first column if the module name that was listed in the data field provided by Shodan contained at least one of the substrings contained in the second column. For instance, if the module string returned by Shodan for a particular device contains the substring 'http,' then according to our procedure the device would be a webserver. We note that under this procedure, a device may be assigned multiple device type labels so that it is possible that a device is both a webserver and a file sharing server if, for instance, the device is running both an http service and an ftp service. The number of each device type is also represented in the third column.

Device Type	Substrings	Count
webserver	{http}	296999
file sharing	{smb, ftp}	110127
mailserver	{imap, pop3, smtp}	21550
database	{sql}	27975
management	{ldap, snmp, ntp, kerberos}	35161
dns	{ 'dns' }	40936
remote access	{telnet, ssh, rdp}	136116
vpn	{pptp, 12tp, openvpn}	42824
router	{router}	70949

Table 3: Labeling Device Types

C Conditional Generator Outputs for Finite Sample Sizes

Figure 6 is showing the fraction of unique samples that were generated by each of our GANs as a function of the number of total samples requested. Note that the data from Table 1 is represented as the top two lines of our graph. For instance, Table 1 is indicating that given 1000 samples (x-axis) generated from our unconditional GAN the fraction of unique samples is 465/1000 which is displayed as the second line from the top in Figure 6. Information pertaining to the accuracy of each of the three GANs is being displayed in an analogous manner in Figure 7.

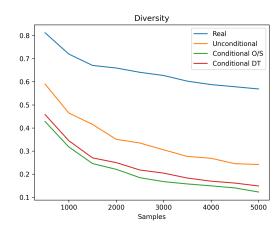


Figure 6: Fraction of Generated Samples that are Unique

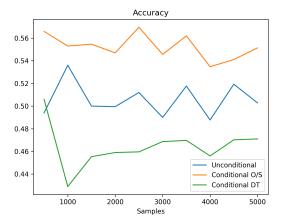


Figure 7: Fraction of Generated Samples that Match Real Samples

References

- [1] C. Stoll. *The Cuckoo's egg: Tracking a spy through the maze of computer espionage*. Doubleday, 1989.
- [2] N.C. Rowe. "A model of deception during cyberattacks on information systems". In: *IEEE First Symposium on Multi-Agent Security and Survivability*. Drexel, PA: IEEE, Aug. 2004, pp. 21–30.
- [3] K. Ferguson-Walter. An Empirical Assessment of the Effectiveness of Deception for Cyber Defense. University of Massachusetts Amherst, 2020.
- [4] Li Zhang and Vrizlynn LL Thing. "Three decades of deception techniques in active cyber defense-retrospect and outlook". In: *Computers & Security* 106 (2021), p. 102288.
- [5] N. Provos. "A virtual honeypot framework". In: *Proceedings of the 13th Conference on USENIX Security Symposium*. San Diego, CA: IEEE, Aug. 2004.
- [6] B. Nagpal et al. "Catch: Comparison and analysis of tools covering honeypots". In: International Conference on Advances in Computer Engineering and Appli-

- cations. Ghaziabad, India: IEEE, July 2015, pp. 783-786.
- [7] M.S.M. Sajjadi et al. "Assessing generative models via precision and recall". In: *Advances in Neural Information Processing Systems*. Montreal, Canada, Dec. 2015, pp. 5228–5237.
- [8] I.J. Goodfellow et al. "Generative Adversarial Nets". In: Advances in Neural Information Processing Systems. Vol. 27. Curran Associates, Inc., 2014.
- [9] P. Isola et al. "Image-to-image translation with conditional adversarial networks". In: *Conference on computer vision and pattern recognition*. Honolulu, HI: IEEE, July 2017, pp. 1125–1134.
- [10] D. Güera and E.J. Delp. "Deepfake Video Detection Using Recurrent Neural Networks". In: *International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. Auckland, New Zealand: IEEE, Nov. 2018.
- [11] M. Frid-Adar et al. "GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification". In: *Neurocomputing* 321.10 (Dec. 2018), pp. 321–331.
- [12] O. Press et al. "Language Generation with Recurrent Generative Adversarial Networks without Pretraining". In: *arXiv* 1706.01399 (2017).
- [13] C. Park et al. "An Enhanced AI-Based Network Intrusion Detection System Using Generative Adversarial Networks". In: *Internet of Things Journal* 10.3 (Feb. 2023), pp. 2330–2345.
- [14] L. Spitzner. "The Honeynet Project: trapping the hackers". In: *Security & Privacy* 1.2 (Apr. 2003), pp. 15–23.
- [15] E. Colbert J. Pawlick and Q. Zhu. "A Game-theoretic Taxonomy and Survey of Defensive Deception for Cybersecurity and Privacy". In: *ACM Computing Surveys* 52.4 (Aug. 2019), pp. 1–28.
- [16] G. Gu et al. "BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation". In: 16th USENIX Security Symposium (USENIX Security 07). Boston, MA: USENIX Association, Aug. 2007.
- [17] Z. Saud and M.H. Islam. "Towards proactive detection of advanced persistent threat (APT) attacks using honeypots". In: 8th International Conference on Security of Information and Networks. New York, NY: ACM, Sept. 2015.
- [18] V. Lorenc V. Bukac and V. Matyáš. "Red Queen's Race: APT Win-Win Game". In: *Cambridge International Workshop on Security Protocols*. Cambridge, United Kingdom: Springer, Mar. 2014, pp. 55–61.
- [19] A.K. Nain. WGAN-GP. https://keras.io/examples/generative/wgan_gp/. 2020.
- [20] The Honeynet Project. *checkpot*. https://checkpot.readthedocs.io/en/master/index.html. 2018.
- [21] Ivre. *Masscanned*. https://github.com/ivre/masscanned.2018.