Towards Oblivious Network Analysis using Generative Adversarial Networks

Zinan Lin
Carnegie Mellon University
Pittsburgh, PA
zinanl@andrew.cmu.edu

Soo-Jin Moon
Carnegie Mellon University
Pittsburgh, PA
soojinm@andrew.cmu.edu

Carolina M. Zarate
Carnegie Mellon University
Pittsburgh, PA
carolinamz1@gmail.com

Ritika Mulagalapalli
Carnegie Mellon University
Pittsburgh, PA
rmulagal@andrew.cmu.edu

Sekar Kulandaivel
Carnegie Mellon University
Pittsburgh, PA
sekark@cmu.edu

Giulia Fanti
Carnegie Mellon University
Pittsburgh, PA
gfanti@andrew.cmu.edu

Vyas Sekar
Carnegie Mellon University
Pittsburgh, PA
vsekar@andrew.cmu.edu

ABSTRACT

Modern systems across diverse application domains (e.g., IoT, automotive) have many black-box devices whose internal structures and/or protocol formats are unknown. We currently lack the tools to systematically understand the behavior and learn the security weaknesses of these black-box devices. Such tools could enable many use cases, such as: 1) identifying input packets that lead to network attacks; and 2) inferring the format of unknown protocols. Our goal is to enable *oblivious network analysis* which can perform the aforementioned tasks for black-box devices. In this work, we explore the use of a recent machine learning tool called generative adversarial networks (GANs) [16] to enable this vision. Unlike other competing approaches, GANs can work in a truly black-box setting and can infer complex dependencies between protocol fields with little to no supervision. We leverage GANs to show the preliminary use cases of our approaches using two case studies: 1) generating synthetic protocol messages given only samples of messages; and 2) generating attack inputs for a black-box system. While there are still many open challenges, our results suggest the early promise of GANs to enable "oblivious" analysis of networked elements.

HotNets '19, November 13–15, 2019, Princeton, NJ, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7020-2/19/11…$15.00
https://doi.org/10.1145/3365609.3365854

ACM Reference Format:
Zinan Lin, Soo-Jin Moon, Carolina M. Zarate, Ritika Mulagalapalli, Sekar Kulandaivel, Giulia Fanti, and Vyas Sekar. 2019. Towards Oblivious Network Analysis using Generative Adversarial Networks. In *The 18th ACM Workshop on Hot Topics in Networks (HotNets '19), November 13–15, 2019, Princeton, NJ, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3365609.3365854

1 INTRODUCTION

Modern systems are seeing a rapid uptick in the number of interconnected "black-box" devices (e.g., IoT devices, electric control units in modern vehicles) where network analysts have no knowledge of internal protocol formats and software. While these devices have increased convenience and efficiency, they have also introduced significant security risks. These devices may have subtle implementation or security weaknesses, leaving gaps for network attacks (e.g., Denial-of-Service or DoS, data exfiltration).

To better secure modern systems running black-box devices, we need tools to systematically uncover the security implications of these devices (e.g., identify potential attack inputs, infer behavioral models). Such tools could enable many use cases: 1) reverse-engineering protocol formats of proprietary protocols; 2) understanding interoperability across different devices and protocols; 3) identifying device inputs (packets) that drive these devices into an exploitable state; and 4) generating accurate fingerprints for black-box devices. As a concrete example, control units in in-vehicle networks have become a target for attacks causing security hazards (e.g., shutting down control units in vehicles [12, 22],

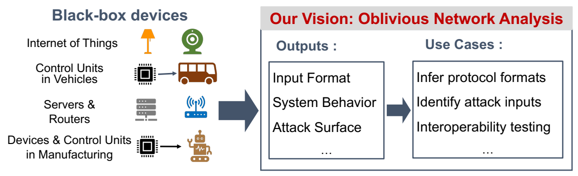


Figure 1: Vision: oblivious network analysis

gaining control of a vehicle's function [25]). In this case, we need tools to enable the network operator to proactively identify those adversarial input packets to vehicular control units. Such tools can benefit many diverse domains (e.g., manufacturing, IoT) with networked systems of black-box devices.

Prior work on uncovering the security implications of systems (e.g., [8, 15, 34]) assume a baseline level of knowledge about internal workings, protocol formats and/or an access to code or binaries. These assumptions do not hold for black-box devices. In this paper, we set an ambitious research agenda: *Is it possible to do these tasks with little knowledge about internals of protocol formats and software?* Our vision is to enable **oblivious network analysis** by equipping operators with tools that can analyze these black-box devices across different applications domains as shown in Figure 1.

Our contribution in this paper is to shed light on how we can leverage a recently-developed machine learning tool called Generative Adversarial Networks (GANs [16]) to achieve this vision. At a high-level, GANs are a new type of neural architectures that can implicitly learn an underlying distribution based on samples and generate new samples from that distribution. Specifically, GANs are appealing for their ability to infer constraints and correlations in the input space to automatically synthesize input packets that meet certain properties. Compared with competing approaches (e.g, regular expression learning) which make some assumptions on inputs, GANs enable "truly" oblivious network analysis.

In this paper, we demonstrate the feasibility of this vision with two preliminary use cases.

Identifying protocol-compliant packets (§4.1): The first use case is to generate protocol-compliant packets for a black-box protocol where the network operator only has access to a seed set of observed protocol messages. For example, modern vehicles deploy electronic control units (ECU) that often use proprietary protocols (e.g., tire pressuring monitor) [19]. To prevent attacks that exploit ECUs, we demonstrate the feasibility of a GAN-based tool for producing new, protocol-compliant messages from raw binary packets.

Identifying attack inputs for network devices (§4.2): A second use case is using GANs to generate attack inputs for network protocols. Attackers often send carefully-crafted packets to devices in an effort to trigger network attacks. For instance, in amplification attacks, attackers search for input packets that trigger large responses [31]. To this end, we build a preliminary tool that only requires a black-box system for identifying exploitable packets that induce high amplification (i.e., trigger large responses). One challenge is that attack packets are rare, which makes it difficult for the GAN to learn the patterns of attack packets in particular. To address this, we design a feedback training mechanism to handle highly-imbalanced data. Our tool can identify UDP packets that cause high amplification.

We believe that these preliminary case studies are really only scratching the surface of the potential for GANs in these kinds of "oblivious" analysis use cases. We hope that this tool can open up a range of new exciting research directions both in theory and in practice. For instance, instead of just identifying input packets, can we explicitly learn the input grammar to describe a set of identified packets? Such capability will enable operators to generate signatures and/or generate patches for these input packets. Further, can we improve GAN architectures to identify attack inputs for stateful black-box devices [27] (e.g., firewalls that keep connection state)? Can we generalize the lessons learned from our preliminary case studies to apply to other application domains?

2 MOTIVATION AND RELATED WORK

Many application domains (e.g., IoT, manufacturing, automotive) can benefit from oblivious network analysis. In this section, we discuss some concrete use cases.

2.1 Use Cases

Infer protocol formats: One concrete and useful application is to infer protocol formats given only network traces from deployments. Many application domains feature proprietary protocols. To enable security testing frameworks, it is important to understand the structure of protocol-compliant messages, which defines the search space of exploitable packets. These inferred protocol-compliant messages can be used for many applications. For instance, we can use to gain understanding about malware and botnets [7, 11]. We can also use the inferred messages to identify vulnerabilities [6, 14].

Identify attack inputs: Another application is directly identifying device inputs (i.e., packets) that trigger outputs of interest to an attacker. As a concrete example, consider an operator of IoT devices who wants to identify input packets that cause high CPU consumption, thus leading to DoS attacks. Similarly, anomalous industry control system (ICS) behaviors have resulted in loss of life on real-world factory floors [1]. Hence, the ICS operator needs to proactively identify device inputs that trigger physical hazards.

Test interoperability across devices: Interoperability is the ability for more than one system to successfully communicate over an agreed-upon protocol. Ensuring interoperability is crucial to the reliability of networked systems [32]. By doing oblivious network analysis, we can automatically understand the interoperability of different devices communicating via the same protocol. For example, two robots in ICS use the same protocol. But, due to implementation issues in one robot, a protocol-compliant input packet will cause the robot to crash.

2.2 Prior Works and Limitations

Prior work has long explored how to reverse-engineer protocols [4, 5, 14] and automatically identify attack inputs for network devices [2, 33], protocols [14], and software [35, 41, 42].

Source code or binary analysis: Many have looked into enabling these applications with access to code (e.g., [33]) and/or binary (e.g., [8, 34, 42]). For instance, CASTAN [33] uses symbolic execution to generate adversarial workloads for network functions. Unfortunately, it is practically challenging to obtain the source code for these proprietary network elements.

Analysis assuming known protocol format: Other approaches [2, 27, 36] perform protocol and device analysis assuming protocol format is known. For instance, Alembic [27] tackled the problem of inferring the stateful model in a form of a finite state machine (FSM) of network functions (e.g., firewalls, NAT). SFADiff [2] tackles the problem of automatically generating attack inputs for firewalls. This class of work assumes that the protocol format is given; e.g., firewalls take TCP packets. The focus of these works is on inferring the stateful model (i.e., how a firewall tracks TCP state machine). These works are beneficial in certain application settings but cannot be directly applied when protocol formats are unknown.

Protocol reverse engineering tools: Similar to our proposed use case, prior works (e.g., [4, 14, 15]) have looked at inferring format of unknown protocols given network traces. For instance, PIP [4] uses bioinformatics algorithms [28, 29, 37] to perform alignment of message bits. ProDecoder [39] uses statistical methods to identify tokens to identify sequences of strings that appear together frequently. NEMESYS [15] determines field boundaries by examining the distribution of changes in the bits throughout the protocol message. First, these works assume that the given input network traces are complete and do not have ability to generate more protocol-compliant messages. The appeal of GAN is its ability to automatically generate new, randomized, protocol-compliant packets given an input trace. More importantly, these works (e.g., [4, 15]) *cannot model functional dependencies* between fields (i.e., the value of field X depends on field Y). GANs overcome the limitations of these prior works by automatically learning complex correlations and dependencies in an unsupervised manner.

Fuzzing tools: Existing software fuzzing tools [42] identify inputs to cause unintended software behavior such as crashes. At a high-level, these software fuzzing tools assume access to the binary [34, 35, 42] and/or typically intend to find inputs that lead to unintended actions [34, 35, 41, 42] such as crashes. As an example, one state-of-the-art fuzzer, American Fuzzy Lop (AFL) [42], mutates inputs and focuses on testing new code paths. Unfortunately, in our problem settings, we do not have visibility into the coverage of code path.

Apart from software, fuzzing has been broadly applied to other domains such as IoT, CPS, and network devices (e.g., [9, 40]). Unfortunately, these tools also require access to the application code [9] and/or domain-specific tools (e.g., emulator for routers [40]) which make them not applicable for broad types of network devices and protocols. In summary, we find that these fuzzing tools across multiple domains are not black-box in nature and requires access to varying degrees of inputs and prior knowledge.

While our proposal is related to fuzzing, our vision goes beyond the capabilities of these fuzzing tools by requiring less inputs and the use cases are different. We posit that this vision can also complement existing fuzzing efforts. For instance, the normal network traffic generated by one of our use cases in §4.1 can be used to improve existing black-box fuzzing tools which require having access to network traces (e.g., [14]). Further, we conjecture that GANs are also more efficient at generating more viable (well-formed) packets than pure black-box fuzzing techniques, which typically produce random bit variations that can result in non-protocol-compliant fuzzing inputs (more in §5).

Case for oblivious network analysis: While it is indeed difficult for aforementioned use cases to have access to internal structure or code, it is however easy to obtain "in situ" data and signals of how these black-box devices behave. Hence, in enabling **oblivious network analysis**, we assume we have: 1) output access to the black-box device; and/or 2) access to "samples" of network traces or example inputs (as network operators can easily monitor the network and collect traffic in many cases). Unlike many prior works, we assume less knowledge of internal working of devices and do not need access to the code and/or binary.

Roadmap: Having motivated the need for oblivious network analysis, we give background on GANs and why they are suitable baseline to enable the vision in §3.1. Then, we discuss our preliminary applications: 1) inferring protocol-compliant packets for CAN protocols used in in-vehicle networks (§4.1), and 2) identifying input packets to network servers that cause high amplification for DNS implementations (§4.2).

3 GENERATIVE ADVERSARIAL NETWORKS

GANs are a type of data-driven generative model that learns to generate random samples from an underlying distribution, given only a collection of samples from the distribution. As seen from our use cases in §2, oblivious network analysis entails identifying input packets that drive these black-box devices to certain states (e.g., protocol compliance, cause high CPU consumption, cause robots to move wildly). Hence, we observe a natural parallel between our goal and the capability of GANs. In this section, we first give a brief introduction to GANs and a variant called conditional GANs (§3.1). We next discuss why GANs are a suitable tool for our goals (§3.2).

3.1 Background on GANs

GANs [16] generate samples from a target distribution using a neural network called a *generator*. The generator takes as input a random vector z and outputs a sample $\hat{x}$ (e.g., an image) that is a deterministic function of z. By feeding the generator random z's, a user can produce random samples from the underlying distribution. In order to train this generator to generate realistic samples, GANs use another component called discriminator, which is also implemented by a neural network. This discriminator reads either a real sample or a sample generated by the generator. The goal of the discriminator is to classify whether the input is real or generated. The generator, on the contrary, tries to fool the discriminator into thinking its generated samples are real. The generator and discriminator are trained alternately, in a procedure called *adversarial training* [16]. The architecture of GANs is shown in Fig. 2.

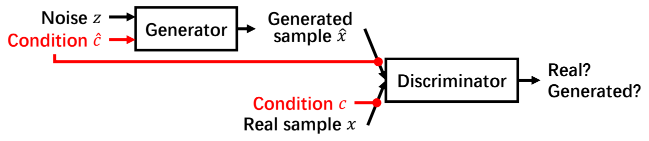


Figure 2: Architecture of vanilla GAN [16] (black) and conditional GAN [26] (black + red).

It can be shown that with infinite training data and generator/discriminator capacity, the generator and discriminator reach an equilibrium where the generator learns the real distribution and the discriminator does no better than random guessing [16]. In practice, this point is never reached because of limited training data and neural network capacity.

Conditional GANs (CGANs): Beyond basic GAN, there are extensions that can drive system to generate conditionally interesting inputs, that can be helpful for our use cases. Specifically, CGANs [26] (Fig. 2) are a variant of GANs for learning *conditional* distributions. For example, consider a dataset of human facial images. Training a vanilla GAN can generate both male and female faces but vanilla GAN does not give us the control over the gender. However, we can train a CGAN to control this factor assuming training data has gender labels.

3.2 Why GANs are a Suitable Tool

GANs have capabilities that enable oblivious network analysis. We describe our reasons below.

GANs can deal with enormous search spaces: One challenge in oblivious network analysis is that without any prior knowledge of protocols, the space of possible protocol-compliant or exploitable packets is very large. GANs have been able to generate realistic images as large as 1024×1024 by observing a relatively small subset of possible images [21]. This suggests that GANs can learn patterns even in very high-dimensional spaces.

GANs can learn complex data structures: The other challenge in achieving oblivious network analysis is that protocols may have complicated structures (e.g., dependencies between different fields), and a system's or device's response to packets can also be difficult to model. We observe that GANs have been successfully applied to infer complicated, high-dimensional distributions as shown in their applications to generate photo-realistic high-resolution images [21]. Hence, GANs are able to infer complex structures in contrast to prior works (e.g., [5, 15]) that we discussed in §2. Further, the generality of GANs to work on a wide range of images (e.g., human faces, bedrooms, plants) [21] suggests that they can capture diverse data patterns.

4 PRELIMINARY CASE STUDIES

We consider two case studies illustrating our vision. We show that a native GAN without special design can achieve many non-trivial tasks. In addition, by altering network architectures, utilizing more advanced GAN variants, and introducing new GAN training mechanism, the capability of GANs for oblivious networking can be further boosted.

4.1 Generating Protocol-Compliant Packets

The first problem we address for network operators working with black-box protocols is the ability to synthesize protocol-compliant packets. Such packets could be used as a stepping stone towards vulnerability testing. As discussed in §2, we assume the operator does not have access to the source code or previous knowledge of the protocol. However, we do assume that the operator can collect a seed set of raw binary, protocol-compliant packets, including the ability to distinguish the start and end of a packet. Our goal is to use these seed packets to generate new protocol-compliant packets with the hope that these packets can potentially induce behaviors or actions in a given system. In tackling this goal, we face the following main challenge.

Protocols are complex: We scope our work to a stateless definition of protocols: a sequence of fields that can include dependencies within or between fields. We acknowledge that protocols can also be defined as a series of packets in a particular order (e.g. SYN, SYN-ACK, ACK for TCP), but as this is stateful, it is outside of our scope. Using our definition of protocols, we categorize field dependencies into two major groups:

- *intra-field dependencies*, where one field can only take certain candidate values. Examples include rdatatype in DNS requests, which can only take certain values.
- *inter-field dependencies*, where a field is a function of one or more other fields. Examples include cyclic redundancy checks (CRC) in Ethernet frames, Content-Length fields in HTTP responses, and data-length codes in Controller Area Network (CAN) frames.

While these types of dependencies are not exhaustive, they introduce substantial complexity. Indeed, modeling such dependencies in an unsupervised fashion is one of the main challenges for prior protocol reverse-engineering approaches [4, 5, 15].

4.1.1 *Design.*

We consider a GAN-based protocol reverse engineering tool with the following baseline design. It uses a *gradient-penalized Wasserstein loss*, which is known to encourage good performance on discrete data types [17]. To deal with variable-length signals, we represent each protocol packet as a sequence of bits (i.e. 0 and 1). We pad packets with 2's until the maximum possible length. GANs may not perfectly learn this embedding; generated packets may insert a 2 between strings of 0's and 1's. When such cases happen, we simply treat the generated packet as ending at the first occurrence of a 2.

For our preliminary design, we use a multi-layer perceptron (MLP) generator and MLP discriminator, both with 6 layers and 600 units per layer. For each bit of the protocol, the generator's output layer has a 3-dimensional softmax function that converts the numeric GAN outputs to a probability vector over "0", "1", and "2". The described architecture is fairly general to explore what GANs can do without special tuning. We will see that this architecture, yet simple, can learn many non-trivial dependencies. With simple adjustment to the network architectures, GAN can learn even more complicated dependencies.

4.1.2 *Preliminary Evaluation.*

To evaluate this approach, we measure its performance on a set of increasingly-complex dependencies (Table 1).

Dependency Type		Accuracy (%)
Intra-Field	Multiple of 4	100
	ASCII	93
Inter-Field	XOR (8 bit)	82
	Field length	85
	TCP Checksum	90
	CRC-15	0
	CRC-15 (new)	39

Table 1: For each dependency, we generate 10,000 messages and compute the fraction of correctly capturing the desired dependency.

GANs learn intra-field dependencies: We develop two synthetic protocols testing intra-field dependencies. Each has three fields, where the first two fields contain arbitrary values and the third field contains (a) a multiple of four, or (b) a printable ASCII character. The aim here is to infer that the last field must be within the appropriate value range, despite placing random values in the first two fields. We find that GANs correctly learn both dependencies.

GANs learn simple inter-field dependencies: We first consider synthetic protocols that demonstrate common dependencies found in network protocols, such as XOR, a TCP Checksum, a field that captures the length in bits of another field, and a 15-bit cyclic redundancy check (CRC). The GAN predictably learns the easier dependencies (XOR, TCP Checksum). For the field length experiments, we used messages from the CAN protocol, widely adopted in automotive systems. CAN has a field that captures the length of another field in bits. To our surprise, it also learns to capture the length of a field without special tuning. The approach begins failing on CRCs; we observe performance no better than random guessing. We hypothesize that this is because computing a CRC requires repeated long division by a fixed *generator polynomial*, unknown to the GAN.

GANs can learn more complicated inter-field dependencies: To capture CRC-like repeated dependencies on unknown constants, we modify our architecture by feeding the discriminator input (i.e., the real or generated protocol message) directly to every layer in the discriminator. This allows the discriminator to make more direct use of the input as it learns to long-divide numbers. This caused a substantial jump in CRC-15 accuracy to 39 percent. Note that this architecture actually makes the GAN more general and powerful instead of being restricted to CRC functions, because we do not delete any component from the original architecture. This experiment shows that architecture alterations can significantly improves GANs' protocol learning capabilities, and open up a large research space on architecture design for protocol learning.

GAN is not simply memorizing the seed samples: We also measure *the fraction of new samples* observed in the test set to verify that GAN is not just memorizing the training samples. For example, in TCP Checksum, the fraction of new samples is 86%; in CRC-15 (new), the fraction of new samples is 66%.

4.2 Identifying Attack Inputs to Black-Box Systems

A second problem is identifying what packets can trigger security-sensitive events like bugs or high resource utilization. The assumption here is different from §4.1 in two ways: (1) The operator has access to the system, to which they can make queries and measure whether the target event happened; (2) We do not require the operator to have prior access to traffic traces. Even if we can observe traffic, the packets that trigger target events are usually very rare, which helps little. As in §4.1, we limit our scope to stateless protocols. Since the system is black-box, our goal is to generate packets that can trigger events of interest. Besides the challenges listed in §4.1, we have the following new challenge:

Target packets are rare: A straw-man solution is to generate random packets, pass them through the system, keep the ones that trigger the target event, and then apply the approach in §4.1 on those packets to generate more packets of interest. However, since the target events are rare, we would need to produce an unreasonable number of system calls to get enough useful packets to train the GAN.

Therefore, this problem is more challenging than the one in §4.1. In the next subsection, we will see how we can utilize more advanced GAN variant and introduce new training mechanism to tackle the problem.

4.2.1 Design. Besides the design choices in §4.1.1, we explore the following approaches to deal with the sparsity problem. For simplicity, we call packets that can trigger the target event as positive packets, and the rest as negative packets.

Utilizing Conditional GAN (CGAN): Because of the sparsity problem, most of the packets we get through random requesting are negative, which cannot be utilized by the approach in §4.1. However, since the patterns of positive and negative samples are complementary, those negative packets can actually help the learning of positive packets. Therefore, we want a different GAN architecture that can incorporate negative packets in training. CGAN (§3.1) is exactly what we need. By setting the condition as a binary indicator on whether the samples are of interest or not, CGAN can jointly learn to generate positive and negative packets.

New training mechanism: However, since the number of positive samples in training data is too low, CGAN still has a hard time learning the patterns of positive packets. In order to increase the number of positive samples in training data without making too many queries, we propose a novel feedback training mechanism: we use the trained CGAN to generate packets with condition=positive, pass them through the system to get the actual label, and add them back to the training data. Using this way we can increase the number of positive samples in training data much more efficient than random sampling. The workflow is shown in Fig. 3. We do not claim that this approach is perfect, but we just want to convey the idea that modifying training mechanism can also help GANs identifying attack inputs.

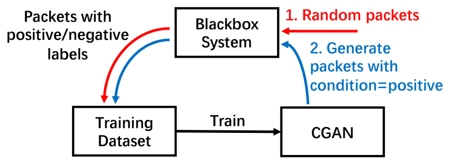

Packets with positive/negative labels
Blackbox System
1. Random packets
2. Generate packets with condition=positive
Training Dataset
Train
CGAN

Figure 3: The training mechanism of our approach.

As for the architecture of generator and discriminator, we use the same choices as in §4.1, except that both discriminator and generator has an extra input of the binary condition represented in one-hot encoding ("10" or "01").

4.2.2 Preliminary Evaluation. **Scenario:** In the experiment we consider DNS amplification attacks as discussed in §2: we want to find DNS request packets that will trigger a large response on a DNS resolver. We say a request packet is of interest if the *size of response* / *size of request* > 10. As a preliminary evaluation, we consider a gray-box setting, where we assume that we know the request should be a UDP packet, and only let GANs explore the UDP payload; in the position of the following 5 DNS request fields: *qr*, *opcode*, *rdatatype*, *rdataclass*, and *url*, we provide candidate values for GANs to choose (the corresponding output layer in generator is a softmax whose dimension equals the length of the candidate list). But for the other 12 fields, we assume no prior knowledge and GANs will explore all possible bits. Though this is not a complete black-box setting, the search space is still very large (3.6×10^{17}).

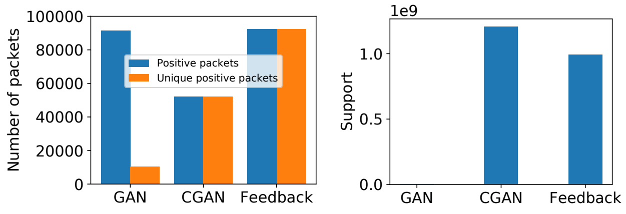
Metrics: We consider the following metrics.

- *Number of positive packets:* We use the GAN to generate 100k packets with condition=positive, and count the number of unique packets among them that are actually positive (by checking with the DNS resolver).
- *Support size of positive packets:* This metric is defined as the asymptotic value of the above metric when the size of generated packets goes to *infinity*. Obviously, we cannot get the exact value of this metric under limited resource. We use the method proposed in [3] to estimate the lower bound of this metric.

Results: The results are shown in Fig. 4. For all methods, the initial training data is 100k random DNS requests (each field is randomly picked from all possible bit combinations or a list of candidates), among which 778 (0.78%) packets are positive.

(1) CGAN helps: Because of insufficient training data, naively training the approach in §4.1 on those 778 packets gives significant overfitting, as evidenced by the phenomenon that most of generated positive packets are duplicate and the support size is almost invisible compared with the other two methods. By contrast, CGAN trained with negative samples performs much better, improving both metrics.

(2) Feedback training mechanism helps: Feedback training gives further improvements over the first metric, with similar performance on the second metric. In particular, Fig. 4(a) shows that among packets generated by the feedback training approach, over 90% are actually positive packets. Fig. 4(b) shows that the support of positive packet of feedback training approach is on the order of 10^9.



(a) Number of positive packets (b) Support of positive packets

Figure 4: Results on DNS amplification attacks.

Again, this experiment is not completely black-box as we provide candidate lists for some fields, but this task is still very challenging considering the large search space. The results show that although vanilla GAN does not work well in this problem, more advanced GAN variants and specially designed training mechanism are promising for this problem.

5 DISCUSSION

We conclude by highlighting a subset of new and exciting opportunities for further research that our vision opens up.

Supporting explicit learning: Operators not only want to identify input packets, but also infer the "grammar" describing a set of packets that meet certain properties (e.g., protocol compliance). Disentangled Generative Adversarial Networks are a new approach that may provide a preliminary solution in this space. Disentangled GANs aim to capture the features responsible for the learned generative model [10, 20, 24] . Inferring the protocol format is useful to generate signature, patches, and understand the underlying properties of protocol structure, enabling richer set of applications.

Handling stateful behavior: Our preliminary use cases handle stateless protocols and behaviors. However, many devices and protocols employ stateful behavior where supporting stateful behavior can also enable a set of new applications. For instance, sending *a sequence of network packets* to IoT devices and/or network gateways can incur high CPU consumption, affecting the availability of these devices. To this end, we are exploring the ideas of incorporating Recurrent Neural Networks (RNNs) [18] into the generator to learn time-dependent information and the transitions between states (e.g., [13, 23]), or combining with other prior efforts for inferring stateful behavior [2, 27].

Benefits compared to black-box fuzzing techniques: As mentioned in §2.2, we conjecture that GAN-based approach is more efficient at generating well-formed, protocol-compliant inputs, as it can infer the characteristics of protocol-compliant inputs. However, traditional black-box fuzzing approaches typically generate random bit variations, resulting in many unviable messages. While we do not have full evaluation results, we did observe some evidence in our evaluation (Table 1, Fig. 4(a)), that the GAN learns to generate a high fraction of viable (protocol-compliant) messages.

Enabling more use cases and opportunities: Our agenda can also benefit multiple application domains. For instance, while we have shown the promise of using GAN to identify protocol-compliant messages for CAN protocols, we can extend the mechanism to identify packets that cause physical hazards. Similarly, we can apply the idea in the context of manufacturing industry (e.g., to identify inputs causing robots to behave wildly).

ACKNOWLEDGMENTS

The authors would like to thank Yucheng Yin, Yun Chen Wu, Ping Luo, and Wenyu Wang for helpful discussions about experiments.

This research was sponsored in part by the U.S. Army Combat Capabilities Development Command Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. This work was also supported in part by by Siemens AG and by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. This work used the Extreme Science and Engineering Discovery Environment (XSEDE) [38], which is supported by National Science Foundation grant number ACI-1548562. Specifically, it used the Bridges system [30], which is supported by NSF award number ACI-1445606, at the Pittsburgh Supercomputing Center (PSC).

REFERENCES

[1] [n. d.]. Robot kills worker at Volkswagen plant in Germany. https://www.theguardian.com/world/2015/jul/02/robot-kills-worker-at-volkswagen-plant-in-germany. ([n. d.]). Accessed: 2019-06-24.

[2] George Argyros, Ioannis Stais, Suman Jana, Angelos D. Keromytis, and Aggelos Kiayias. 2016. SFADiff: Automated Evasion Attacks and Fingerprinting Using Black-box Differential Automata Learning. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 1690–1701. https://doi.org/10.1145/2976749.2978383

[3] Sanjeev Arora and Yi Zhang. 2017. Do gans actually learn the distribution? an empirical study. *arXiv preprint arXiv:1706.08224* (2017).

[4] Marshall A Beddoe. 2004. Network protocol analysis using bioinformatics algorithms. *Toorcon* (2004).

[5] Georges Bossert, Frédéric Guihéry, and Guillaume Hiet. 2014. Towards automated protocol reverse engineering using semantic information. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM, 51–62.

[6] Sergey Bratus, Axel Hansen, and Anna Shubina. 2008. LZfuzz: a fast compression-based fuzzer for poorly documented protocols. *Darmouth College, Hanover, NH, Tech. Rep. TR-2008* 634 (2008).

[7] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. 2009. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 621–634.

[8] Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn Song. 2007. Polyglot: Automatic Extraction of Protocol Message Format Using Dynamic Binary Analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*. ACM, New York, NY, USA, 317–329. https://doi.org/10.1145/1315245.1315286

[9] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. 2018. IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing.. In *NDSS*.

[10] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*. 2172–2180.

[11] Chia Yuan Cho, Eui Chul Richard Shin, Dawn Song, et al. 2010. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 426–439.

[12] Kyong-Tak Cho and Kang G Shin. 2016. Error handling of in-vehicle networks makes them vulnerable. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1044–1055.

[13] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. 2017. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633* (2017).

[14] Hugo Gascon, Christian Wressnegger, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. 2015. Pulsar: Stateful Black-Box Fuzzing of Proprietary Network Protocols. In *SecureComm*.

[15] Erol Gelenbe, Gökçe Görbil, Dimitrios Tzovaras, Steffen Liebergeld, David Garcia, Madalina Baltatu, and George Lyberopoulos. 2013. NEMESYS: Enhanced network security for seamless service provisioning in the smart mobile ecosystem. In *Information Sciences and Systems 2013*. Springer, 369–378.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*. 5767–5777.

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[19] Rob Millerb Ishtiaq Roufa, Hossen Mustafaa, Sangho Ohb Travis Taylora, Wenyuan Xua, Marco Gruteserb, Wade Trappeb, and Ivan Seskarb. 2010. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *19th USENIX Security Symposium, Washington DC*. 11–13.

[20] Insu Jeon, Wonkwang Lee, and Gunhee Kim. 2018. IB-GAN: Disentangled Representation Learning with Information Bottleneck GAN. (2018).

[21] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2017. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196* (2017).

[22] Sekar Kulandaivel, Tushar Goyal, Arnav Kumar Agrawal, and Vyas Sekar. 2019. CANvas: Fast and Inexpensive Automotive Network Mapping. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/usenixsecurity19/presentation/kulandaivel

[23] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. 2019. Generating High-fidelity, Synthetic Time Series Datasets with DoppelGANger. *arXiv preprint arXiv:1909.13403* (2019).

[24] Zinan Lin, Kiran Koshy Thekumparampil, Giulia Fanti, and Sewoong Oh. 2019. InfoGAN-CR: Disentangling Generative Adversarial Networks with Contrastive Regularizers. *arXiv preprint arXiv:1906.06034* (2019).

[25] Charlie Miller and Chris Valasek. 2015. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA* 2015 (2015), 91.

[26] Mehdi Mirza and Simon Osindero. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* (2014).

[27] Soo-Jin Moon, Jeffrey Helt, Yifei Yuan, Yves Bieri, Sujata Banerjee, Vyas Sekar, Wenfei Wu, Mihalis Yannakakis, and Ying Zhang. 2019. Alembic: Automated Model Inference for Stateful Network Functions. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 699–718. https://www.usenix.org/conference/nsdi19/presentation/moon

[28] Saul B Needleman and Christian D Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology* 48, 3 (1970), 443–453.

[29] Masatoshi Nei, Fumio Tajima, and Yoshio Tateno. 1983. Accuracy of estimated phylogenetic trees from molecular data. *Journal of molecular evolution* 19, 2 (1983), 153–170.

[30] Nicholas A Nystrom, Michael J Levine, Ralph Z Roskies, and J Scott. 2015. Bridges: a uniquely flexible HPC resource for new communities and data analytics. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*. ACM, 30.

[31] Vern Paxson. 2001. An Analysis of Using Reflectors for Distributed Denial-of-service Attacks. *SIGCOMM Comput. Commun. Rev.* 31, 3 (July 2001), 38–47. https://doi.org/10.1145/505659.505664

[32] Luis Pedrosa, Ari Fogel, Nupur Kothari, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. Analyzing Protocol Implementations for Interoperability. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, Berkeley, CA, USA, 485–498. http://dl.acm.org/citation.cfm?id=2789770.2789804

[33] Luis Pedrosa, Rishabh Iyer, Arseniy Zaostrovnykh, Jonas Fietz, and Katerina Argyraki. 2018. Automated Synthesis of Adversarial Workloads

for Network Functions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. ACM, New York, NY, USA, 372–385. https://doi.org/10.1145/3230543.3230573

[34] Theofilos Petsios, Jason Zhao, Angelos D. Keromytis, and Suman Jana. 2017. SlowFuzz: Automated Domain-Independent Detection of Algorithmic Complexity Vulnerabilities. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 2155–2168. https://doi.org/10.1145/3133956.3134073

[35] Dongdong She, Kexin Pei, Dave Epstein, Junfeng Yang, Baishakhi Ray, and Suman Jana. 2019. Neuzz: Efficient fuzzing with neural program learning. In *2019 IEEE Symposium on Security and Privacy (SP)*.

[36] Suphannee Sivakorn, George Argyros, Kexin Pei, Angelos D. Keromytis, and Suman Jana. 2017. HVLearn: Automated Black-box Analysis of Hostname Verification in SSL/TLS Implementations. In *Proceedings of the 38th IEEE Symposium on Security & Privacy*. San Jose, CA.

[37] Temple F Smith, Michael S Waterman, et al. 1981. Identification of common molecular subsequences. *Journal of molecular biology* 147, 1 (1981), 195–197.

[38] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D Peterson, et al. 2014. XSEDE: accelerating scientific discovery. *Computing in Science & Engineering* 16, 5 (2014), 62–74.

[39] Yipeng Wang, Xiaochun Yun, M Zubair Shafiq, Liyan Wang, Alex X Liu, Zhibin Zhang, Danfeng Yao, Yongzheng Zhang, and Li Guo. 2012. A semantics aware approach to automated reverse engineering unknown protocols. In *2012 20th IEEE International Conference on Network Protocols (ICNP)*. IEEE, 1–10.

[40] Zhiqiang Wang, Yuqing Zhang, and Qixu Liu. 2013. RPFuzzer: A Framework for Discovering Router Protocols Vulnerabilities Based on Fuzzing. *TIIS* 7 (2013), 1989–2009.

[41] Maverick Woo, Sang Kil Cha, Samantha Gottlieb, and David Brumley. 2013. Scheduling Black-box Mutational Fuzzing. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security (CCS '13)*. ACM, New York, NY, USA, 511–522. https://doi.org/10.1145/2508859.2516736

[42] Michal Zalewski. 2014. American fuzzy lop. (2014). http://lcamtuf.coredump.cx/afl/