DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security Workshop

(DYNAMICS)

December 9th and 10th, 2019

Pre-Proceedings

Welcome to DYNAMICS 2019

On behalf of the program committee, we are pleased to welcome you to the 2019 Workshop on DYnamic and Novel Advances in Machine learning and Intelligent Cyber Security (DYNAMICS), held this year from December 9th to 10th in the Condado Plaza Hotel in San Juan, Puerto Rico. DYNAMICS is a two-day workshop whose goal is to focus on applying leading machine learning techniques to real world problems in Information Security.

Information security has relied on artificial intelligence to improve efficiency for decades. Examples include the development of the Intrusion Detection Expert System (IDES) in the 1980s, the use of Bayesian Belief networks for antispam throughout the 2000's, or the current day adoption of Deep Learning techniques to detect anomalous behavior. While security personnel need effective machine learning to address security problems at scale, they have a deeper experience with its failures than most other communities. This focus - machine learning as it relates to real world information security problems - is the motivation for DYNAMICS.

As in last year's inaugural workshop, DYNAMICS 2019 includes both paper presentations and forums. These pre-proceedings include each paper being presented, as well as a summary of each forum. The workshop is intended to be highly interactive, with full audience participation in the forums. We look forward to a lively debate on each topic, and to incorporating your perspectives on them, and to building a community that will continue to address these topics beyond the end of the workshop.

Again, thank you for attending, and enjoy the workshop!

Michael Clifford and Michael Collins

Co-Chairs, DYNAMICS 2019

Workshop Schedule Page 1

Time	Monday Morning	Tuesday Morning
7:30-8:30 AM	**Breakfast (**Brisas Del Mar)	**Breakfast (**Brisas Del Mar)
8:30-9:00	**Welcome and Introduction** DYNAMICS Co-Chairs: Michael Clifford, Noblis Michael Collins, USC ISI	**Socialization and Networking**
9:00-10:00	**Keynote**: Protecting Privacy by Attacking Learning Systems Speaker: Dr. Rachel Greenstadt Tandon School of Engineering, NYU	**Forum**: Attacks on Privacy Preserving Systems Moderator: Nidhi Rastogi, RPI
10:00-10:45	**Paper**: DeepCloak: Adversarial Crafting as a Defensive Measure to Cloak Processes Presenter: Dr. Berk Sunar, WPI	**Paper**: Malicious Behavior Detection Based on Extracted Features from APIs for Windows Platforms Presenter: Dima Rabadi, I2R Singapore
10:45-11:00	Coffee and Tea Break	Coffee and Tea Break
11:00-12:00	**Forum**: Privacy Preserving Federated Learning Moderator: Nidhi Rastogi, RPI	**Forum**: Applying ML to improve operational security Moderator: Michael Collins, USC ISI

Workshop Schedule Page 2

Time	Monday Afternoon	Tuesday Afternoon
12:00-1:00 PM	Lunch	Lunch
1:00-1:45	**Paper**: Traffic Generation using Containerization for Machine Learning Presenter: Henry Clausen**,** University of Edinburgh	**Paper**: Generating targeted e-mail at scale using neural machine translation Presenter: Sridhar Venkatesan, Perspecta Labs
1:45-2:45	**Forum**: Challenges for Detection and Forensics Analysis of Advance Persistent Threats for Enterprise Security Moderator: Anoop Singhal, NIST	**Forum**: Continuity and Long-term Maintenance for Machine Learning Systems Moderator: Michael Collins, USC ISI
2:45-3:15	Snack Break	Snack Break
3:15-4:00	**Forum**: Challenges to generating valid security research data Moderator: Michael Collins, USC ISI	**Summary and Closeout**

Dynamics 2019 Committee

Workshop Co-Chairs

Dr. Michael Clifford	Noblis
Dr. Michael Collins	USC ISI

Workshop Committee

Evita Bakopoulou	UC Irvine
Dr. Matt Bishop	UC Davis
Dr. Ebrima Ceesay	Noblis
Sanchari Das	IU Bloomington
Dr. Karl Levitt	UC Davis
Dr. Anthony Palladino	Boston Fusion
Dr. Nidhi Rastogi	Rensselaer Polytechnic Institute
Daniel Kats	Symantec

DYNAMICS is co-located with the 2019 Applied Security Applications Conference (ACSAC) in San Juan, Puerto Rico

Keynote: Attacking Machine Learning to Protect Privacy

Dr. Rachel Greenstadt, New York University

In recent years, there has been an explosion of work showing that machine learning algorithms are vulnerable to adversarial attacks. While the research community works on better defenses against these attacks, they provide an opportunity to push back against abuses in which machine learning technology is applied in ways that invade user privacy or perpetrate discrimination. Protective Optimization Technologies (POTs) are solutions that enable optimization subjects to defend from unwanted consequences. I will discuss how adversarial ML-based POTs can help protect people who want to protect their identity and sensitive attributes such as mental health status when contributing to public discourse online and how these ideas can be applied to protect Internet browsing over Tor from traffic analysis.

Biography

Dr. Rachel Greenstadt is an Associate Professor of Computer Science at New York University where she teaches graduate-level courses in computer security and privacy. She founded the Privacy, Security, and Automation Laboratory at Drexel University in 2008. She leads a research team of PhD students and undergraduates with interests and expertise in information extraction, machine learning, agents, privacy, trust, and security. Dr. Greenstadt was among the first to explore the effect of adversarial attacks on stylometric methods, and the first to demonstrate empirically how stylometric methods can fail in adversarial settings while succeeding in non-adversarial settings. She was among the first to propose the idea of active authentication in a 2008 position paper at the First Workshop on Artificial Intelligence and Security (AISTech).

Dr. Greenstadt's scholarship has been recognized by the privacy research community. She is an alum of the DARPA Computer Science Study Group and a recipient of the NSF CAREER Award. Her work has received the **PET Award for Outstanding Research in Privacy Enhancing Technologies** and the **Andreas Pfitzmann Best Student Paper Award**. She served as co-editor-in-chief of the journal Proceedings on Privacy Enhancing Technologies (PoPETs) for the 2017 and 2018 volumes and will serve as co-program chair for USENIX Security 2021. Her research has been featured in the New York Times, the New Republic, Der Spiegel, and other local and international media outlets.

DYNAMICS 2019
FORUMS

Privacy Preserving Federated Learning

Moderator: Nidhi Rastogi, RPI

Machine learning requires enormous volumes of data for analysis, a problem which is exacerbated with security research as it often looks for rare phenomena. Consider problems in epidemiology: effectively predicting and tracking outbreaks requires access to personal health information. One mechanism for managing the balance between a global view facilitating machine learning and managing privacy is to use a *federated* design, where modeling is done at the edges in direct contact with the data, and model updates, rather than the source data is returned to a central system for normalization and integration.

In this forum, we will discuss the goal of federated defenses and how to implement effective federated machine learning to protect privacy.

Topics

1. Federated learning is often combined with differential privacy, how would we apply this to information security specific scenarios?
2. How to convince information security practitioners to share models rather than data? What ideas do you have that haven't been tried in the last 20 years?
3. How to ensure trustworthy data in a federated learning environment explicitly focused on hostile activity?
4. How to manage adversarial data poisoning at in federated learning?

Challenges for Detection and Forensics Analysis of Advance Persistent Threats for Enterprise Security

Moderator: Anoop Singhal, Computer Security Division, NIST

Advanced Persistent Threats (APT) has become the concern of many enterprise networks. APT can lead to stealing of sensitive data, broken workflow, denial of service and so on which can do harm to both the enterprises and their customers. Based on the purpose of the attacker and the method he/she leverages, an APT can be modeled as a list of attack steps. These APTs present a challenge for current detection methods as they make use of advanced techniques and unknown vulnerabilities. Moreover, the economic damage due to successful APT attack is quite significant. Therefore, the accurate detection and prediction of APT is an ongoing challenge.

Topics

1. Most of the current research work is based on simulated data. It would be beneficial to validate their results based on real APT attacks. However, obtaining such data is not easy and currently there are no publicly available data sources.
2. How to use machine learning to detect attack steps which have no signature.
3. How to use correlation and clustering techniques to construct the attack scenario from the individual attack steps.
4. Attack Prediction: How to use machine learning to predict attacks based on a historical record of the monitored network.

Challenges to Generating Valid Security Research Data

Moderator: Michael Collins, USC ISI

Campbell advanced the idea of *validity* in *Experimental and Quasi-Experimental Design for Generalized Causal Inference*. Validity, as Campbell conceived of it, is a judgement of the strength of an argument, and he identified four classes of challenges to validity – Internal, External, Statistical and Construct. Researchers since Campbell have identified validity challenges specific to their disciplines – for example, education researchers have noted that children subjected to multiple tests will start sabotaging them, which impacts the value of the data.

In this forum, we will discuss how validity impacts security experimentation. Beginning with Campbell's four primary categories, we will discuss how they apply to security experiments, and how we might expand Campbell's original categories.

Topics

1. How do the original classes of validity (Internal, External, Statistical and Construct) relate to contemporary information security?
2. Different disciplines have expanded these validity classes to reflect the circumstances unique to their discipline. For example, psychologists have expanded External Validity into Population and Ecological validity. Do we have unique problems (such as actively adversarial subjects) that require new classes of validity?
3. How do we apply validity challenges when working with machine learning? Does it affect data selection? Training? Result evaluation? If it affects everything, do we apply it in different ways at different stages?
4. Can we develop ways to tag validity problems with training sets?

Attacks on Privacy-Preserving Systems

Moderator: Nidhi Rastogi, RPI

Researchers apply machine learning to break the guarantees of privacy-preserving systems such as onion routing, encrypted voice communications, and anonymized datasets. There is a long and successful history of techniques such as hidden Markov models and stylometry breaking privacy, impacting research and lives.

In this forum, we will discuss the conflicts between the design of privacy preserving systems and attacks on them. In particular, we should consider the social and technical elements of privacy preserving systems, and the impact of currently unknown attacks on existing systems. We will discuss whether or not there are distinct classes of privacy preserving systems and how different attacks may affect each.

Topics

1. What is the state of the art for privacy attacks and how do we adapt to them?
2. Are anonymity systems like Tor and I2P privacy preserving? Are there different classes of privacy preserving systems?
3. How does the volume of data required for machine learning impact the lifetime of data for privacy preserving systems?
4. How does the rate of change of the data impact the lifetime of data for privacy preserving systems?
5. Communications often relies on identifying and exploiting repetitive patterns for efficiency – for example, repetitive data facilitates compression. How much efficiency do we have to give up in order to ensure privacy?
6. We can often confound machine learning systems by adding noise to the data, how much noise do we need to effectively confound the systems?

Applying Machine Learning to Improve Operational Security

Moderator: Michael Collins, USC ISI

Operational security personnel increasingly discuss defense actions in terms of EPAH (Events per Analyst-Hour); effectively, the EPAH reflects how many events a single analyst can handle within a single hour. EPAH normally ranges around 10-12 events or so, whereas operational floors are normally receiving thousands (or more) events in an hour. Because operational floors are always running at capacity, a new classifier means that the operators must decide how to (and *if* to) incorporate it into their existing budget of alerts.

EPAH suggests two ways to manage analyst response. The first is to aggregate multiple events into a single event, reducing the number of events an analyst receives. The second is to reduce the time required to process a single event; a fair amount of analyst work involves highly heuristic matching between sources such as WHOIS databases and geolocation. In this forum, we will discuss machine learning venues for operational defenders that move beyond simple classification.

Topics

1. Can we improve EPAH by aggregating similar events?
2. How can we improve analyst processing time? For example, by pre-fetching data, pre-writing responses, or testing likely hypotheses?
3. Threat intelligence requires judgment calls by the intelligence provider. For example, a simple property like an IP's country of origin might involve factors such as the physical location of the IP, the organization that controls it, and whether the country is recognized as such. How can we encode this information to help analysts identify ambiguity and reduce missteps?
4. How can we improve analyst trust?

Continuity and Long-Term Maintenance for Machine Learning Systems

Moderator: Michael Collins, USC ISI

The Internet is a giant moving target, and smart attackers are adaptable. These two behaviors mean that network defenders must constantly update their defenses, this is arguably the strongest argument for threat intelligence: to keep track of how attacker behavior changes.

This degradation process means that defenses must be retrained and recalibrated regularly. In this forum, we will discuss this problem explicitly in the context of information security.

Topics

1. How do we identify that an approach is out of date?
2. How do we deal with sea change events such as QUIC or DNS-over-HTTP adoption?
3. How does the aging out of data affect the impact of information in repositories such as IMPACT or the LARIAT datasets?
4. Do we simply have to rely on time, or can we identify natural experiments for determining change?
5. Do we have a requirement, when developing a system, to identify how it should be maintained and retrained?
6. How can we continuously retrain the system, in particular when dealing with rare and unlabeled events?

DYNAMICS 2019 PAPERS

DeepCloak: Adversarial Crafting As a Defensive Measure to Cloak Processes

Mehmet Sinan İnci
Intel Corporation
mehmet.inci@intel.com

Thomas Eisenbarth
University of Lübeck
thomas.eisenbarth@uni-luebeck.de

Berk Sunar
Worcester Polytechnic Institute
sunar@wpi.edu

ABSTRACT

Over the past decade, side-channels have proven to be significant and practical threats to modern computing systems. Recent attacks have all exploited the underlying shared hardware. While practical, mounting such a complicated attack is still akin to listening on a private conversation in a crowded train station. The attacker has to either perform significant manual labor or use AI systems to automate the process. The recent academic literature points to the latter option. With the abundance of cheap computing power and the improvements made in AI, it is quite advantageous to automate such tasks. By using AI systems however, malicious parties also inherit their weaknesses, most notably the vulnerability to adversarial samples.

In this work, we propose the use of adversarial learning as a defensive tool to obfuscate and mask side-channel information. We demonstrate the viability of this approach by first training CNNs and other machine learning classifiers on leakage trace of different processes. After training a highly accurate model (99+% accuracy), we test it against adversarial learning. We show that through minimal perturbations to input traces, the defender can run as an attachment to the original process and cloak it against a malicious classifier.

Finally, we investigate if an attacker can use adversarial defense methods, adversarial re-training and defensive distillation to protect the model. Our results show that even in the presence of an intelligent adversary that employs such techniques, adversarial learning methods still manage to successfully craft perturbations hence the proposed cloaking methodology succeeds.

ACM Reference Format:
Mehmet Sinan İnci, Thomas Eisenbarth, and Berk Sunar. 2019. DeepCloak: Adversarial Crafting As a Defensive Measure to Cloak Processes. In *Proceedings of DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security (DYNAMICS) Workshop (DYNAMICS).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep learning (DL) has proven to be a very powerful tool for a variety of tasks like handwritten digit recognition [26], image classification and labeling [25, 38, 66], speech recognition [64], lip reading [3], verbal reasoning [40], playing competitive video games [20, 53] and even writing novels [45]. As with any booming technology, it is also adopted by malicious actors e.g.,posting to internet boards to shift public opinion by social engineering. One of the latest known examples of this is the millions of AI-generated comments on the FCC net-neutrality boards [32, 59]. These AI-crafted posts were semantically sound and not easy to detect as fake. Another example of malicious use of AI is for spam and phishing attacks. It is now possible to craft custom phishing e-mails using AI with higher 'yield' than human crafted ones [15, 57]. AI system can even participate in hacking competitions where human creativity and intuition was thought to be irreplaceable. In 2016, DARPA sponsored a hacking competition for AI systems where the task was to find and fix vulnerabilities in computer systems within a given time period [33]. According to a survey among cybersecurity experts, the use of AI for cyber attacks will become more common with time [62].

Classical side-channel attacks (SCA) deal with bulk amounts of noisy data that require human interpretation and intuition to process. Such task are perfectly suited for AI systems and it is reasonable that malicious parties are aware of this opportunity. The academic literature already shows the use of AI for processing side-channel leakage. In 2011 Hospodar et al. [28] demonstrated the first use of machine learning, LS-SVM specifically, on a power SCA on AES and showed that the ML approach yields better results than the traditional template attacks. Later, Heuser et al. [27] showed the superiority of multi-class SVM for noisy data in comparison to the template attacks. Martinasek et al. [42, 43] showed that artificial neural networks can recover AES keys from power measurements with a success rate of 96%. In 2015 Beltramelli [4] used LSTM to collect meaningful keystroke data via motions of the smart watch user. In 2016, Maghrebi et al. [41] compared four DL based techniques with template attacks to attack an unprotected AES implementation using power consumption and showed

that CNN outperforms template attacks. Finally in 2017, Gulmezoglu et al. [24] showed that machine learning can be used to extract meaningful information from cache side-channel leakage to recover web traffic of users.

A straightforward countermeasure against SCAs is to drown the sensitive computation leakage in noise to cloak it. However, this defense has proven to be ineffective in addition to being computationally expensive with significant performance overhead. In this study, we argue that we can do much better than random noise and craft much smaller noise by using adversarial learning (AL). By using AL, we achieve a stronger cloaking effect using smaller changes to the trace hence minimal overhead to the system. Also, the proposed defense does not require redesigning the software or the hardware stacks. The proposed framework can be deployed as an opt-in service that users can enable or disable at wish, depending on their privacy needs at the time.

In summary, attacking machine learning is easier than defending it [18] and if used strategically in a non-traditional way i.e., as a defensive countermeasure, AL against malicious parties with AI capabilities can be quite advantageous. In this work, we expand this idea and show that AL is indeed a useful defensive tool to cloak private processes from AI capable adversaries.

Our Contribution

In this work, we propose a framework and explore the necessary steps to cloak processes against SCAs as well as the defenses a malicious party can use. More specifically in this paper we;

- show how to profile crypto processes with high accuracy via their side-channel leakage using deep learning and various classical machine learning models. We classify 20 types of processes using readily available, high resolution Hardware Performance Counters (HPC). Further, we investigate the effect of parameter choices like the number of features, samples and data collection intervals on the accuracy of such classifiers.
- present the use of AL methods to craft perturbations and add them to the system hardware trace to cloak the side-channel leakage of private processes. We show that this is a strong defense against an attacker using DL classifiers.
- test and quantify the efficiency of different AL methods and present the accuracy and applicability of each attack.
- show that even when adversarial defense methods adversarial re-training or defensive distillation is employed by the attacker, adversarial perturbations still manage to cloak the process.

2 BACKGROUND

In this section, we provide the necessary background information to better understand the attack, adversarial sample crafting as a countermeasure and the improved attack. More specifically, we go over micro-architectural attacks, hardware performance counters, convolutional neural networks (CNNs), and AL attacks.

2.1 Micro-architectural Attacks

Over the last decade, there has been a surge of micro-architectural attacks. Low-level hardware bottlenecks and performance optimizations have shown to allow processes running on shared hardware to influence and retrieve information about one another. For instance, cache side-channel attacks like Prime&Probe and Flush+Reload exploit the cache and memory access time difference to recover fine-grain secret information and even recover secret crypto keys [5, 21–23, 30, 31, 36, 47, 54, 55, 65, 69]. In these works, the attacker exploits micro-architectural leakages stemming from memory access time variations, e.g.,when the data is retrieved from small but faster caches as opposed to slower DRAM memory.

2.2 Hardware Performance Counters

Hardware Performance Counters (HPCs) are special purpose registers that provide low-level execution metrics directly from the CPU. This low-level information is particularly useful during software development to detect and mitigate performance bottlenecks before deployment. For instance, low number of cache hits and high cache misses indicate an improperly ordered loop. By reordering some operations, a developer can significantly improve the performance.

HPCs are also useful to obtain system health check and/or anomaly detection in real-time. For instance, in [1] Alam et al. leverages *perf_event* API to detect micro-architectural side-channel attacks. In 2009, Lee et al. [67] showed that HPCs can be used on cloud systems to provide real-time side-channel attack detection. In [10, 11], researchers used HPCs to detect cache attacks. Moreover, Allaf et al. [2] used a neural network, decision tree, and kNN to specifically detect Flush+Reload and Prime&Probe attacks on AES. Moreover, researchers have shown that by using the fine-grain information provided by HPCs, it is possible to violate personal privacy as well. In [24], Gulmezoglu et al. showed that HPC traces can be used to reveal the visited websites in a system using variety of ML techniques such as auto-encoder, SVM, kNN and decision trees and works even on privacy conscious Tor browser. More applications of HPCs in cyber security countermeasures can be found in [17].

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) is a supervised feed-forward artificial neural network architecture. One important aspect of CNNs is that they don't saturate easily and can reach high accuracy with more training data. Also, unlike the classical ML methods, CNNs do not require data features to be identified and preprocessed before training. Instead, CNNs discover and learn relevant features in the data without human intervention, making them very suitable for automated

tasks. In the past decade, CNNs surpassed humans in many tasks that were considered nearly impossible to automate. This breakthrough is fueled by the rapid increase in GPU powered parallel processing power and the advancements in deep learning. CNNs have been successfully applied to image, malware and many other classification problems. Training a CNN model is done in 3 phases. First, the labeled dataset is split into three parts; training, validation and test. The training data is fed to the CNN with initial hyper-parameters and the classification accuracy is measured using the validation data. Guided by the validation accuracy results, the hyper-parameters are updated to increase the accuracy of the model while maintaining its generality. After the model achieves the desired validation accuracy, it is tested with the test data and the final accuracy of the model is obtained.

2.4 Adversarial Learning

AL is a sub-field of machine learning (ML) that studies the robustness of trained models under adversarial settings. The problem stems from the underlying assumption that the training and the test data comes from the same source are consistent in their features. Studies have shown however that by introducing some small external noise or in this context what is commonly referred to as `adversarial perturbations`, it is possible to craft adversarial samples and manipulate the output of ML models. In other words, by carefully crafting small perturbations, one can push a test sample from the boundaries of one class to another. Due to the mathematical properties of the high-dimensional space that the classifier operates in, this modification can be very small. AL refers to the group of techniques that are used to perturb test samples to classifiers and force misclassification. While there are many different methods of crafting such perturbations, ideally they are desired to be minimal and not easily detectable.

Adversarial attacks on classical ML classifiers (under both white-box and black-box scenarios) have been known for quite some time [6–9, 29, 35, 39, 70]. However, it was Szegedy et al. [60] that first introduced AL attacks on DNNs. The 2013 study showed that very small perturbations that are indistinguishable to human eye can indeed fool CNN image classifiers like ImageNet. The perturbations in the study are calculated using the technique called Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS). This algorithm searches in the variable space to find parameter vectors (perturbation) that can successfully fool the classifier. Later in 2014, Goodfellow et al. [19] improved the attack by using the Fast Gradient Sign Method (FGSM) to efficiently craft minimally different adversarial samples. Unlike the L-BFGS method, the FGSM is computationally conservative and allows much faster perturbation crafting. In 2016, Papernot et al. [50] further improved upon Goodfellow's FGSM by using Jacobian Saliency Map Attack (JSMA) to craft adversarial samples. Unlike the previous attacks, JSMA does not modify randomly selected data points or pixels in an image. Instead, it finds the points of high importance with regards to the classifier decision and then modifies these specific pixels. These points are found by taking the Jacobian matrix of the loss function given a specific input sample, allowing an attacker to craft adversarial samples with fewer modifications.

In 2016 [34, 37, 48], multiple new adversarial attacks were discovered. Moreover, the research showed that these adversarial samples are transferable i.e., perturbations that can fool a model can also work on other models trained on the same task. In [49], Papernot et al. showed that adversarial attacks can also succeed under the black-box attack scenario where an attacker has only access to the classification labels. In this scenario, the attacker has no access to the model parameters such as weights, biases, classification confidence or the loss, therefore, cannot directly compute or use the gradients to craft a perturbation. Instead, the attacker uses the target model as an oracle that labels the inputs and then uses these labeled images to train her own classifier. Authors demonstrated the feasibility of the attack on MetaMind and Deep Neural Network (DNN) classifiers hosted by Amazon and Google. With 84.24%, 96.19% and 88.94% misclassification rates respectively, they were able to fool the targeted classifiers.

In [14], researchers have shown that by iteratively morphing a structured input, it is possible to craft adversarial samples under black-box attack scenario. Authors have implemented the attack against a PDF malware classifier and have reported 100% evasion rate. Moreover, the study acknowledges the fact that black-box attack model has a cost of obtaining labeled data from observations and defines and uses a cost function that takes into account the number of observations. The attack works by adding and/or removing compilable objects to the PDF. Black-box scenario does not assume to obtain confidence scores from the model under attack, only the class output. In summary, the AL is an active research area with plethora of new attacks, defenses and application cases emerging daily [12, 16, 44, 58, 63].

In addition to attack classifiers, adversarial learning have also been used to provide privacy for streaming traffic and facial recognition databases [46, 68]. In contrast to these works, DeepCloak uses adversarial learning to craft additional traffic on micro-architectural level to mask the overall side-channel leakage.

3 METHODOLOGY

Our goal is to show that side-channel classifiers can be successfully stopped using the concept of AL. To validate this assumption, we first train DL-based classifiers using real side-channel data, and show their degradation as the result of AL techniques, even if the DL-based classifier is aware of the AL based cloaking defense. In our experiments, we take the following steps:

(1) Training the process classifier **C** using side-channel leakage **Ω**.
(2) Crafting adversarial samples δ to cloak the user processes and force C to misclassify.
(3) Training a new classifier C' with adversarial defense methods; Defensive Distillation and Adversarial Re-training.
(4) Testing previously crafted adversarial samples δ against the new classifier C'. Also crafting and testing new adversarial samples δ' against the protected classifier C'.

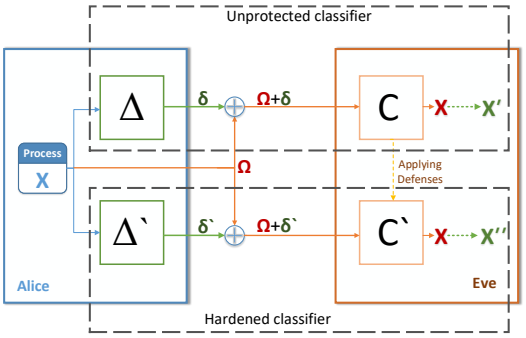

Figure 1: Alice, the defender runs the process X and leaks the Ω. Eve, the attacker obtains the leakage and identifies the process X using the classifier, C. Then, Alice crafts the adversarial perturbation δ and forces C to misclassify the trace as X′. Eve then trains C′ with adversarial retraining and defensive distillation. Now, Eve can classify $\delta' + \Omega$ partially correct. However, when Alice crafts δ' against C′, X is again misclassified.

We outline this methodology in Figure 1. In the first stage, Alice the defender runs a privacy sensitive process X. The eavesdropper Eve collects the side-channel leakage Ω and feeds it into her classifier C and discovers what type of process X is. Then in stage 2, Alice cloaks her process by crafting the adversarial sample δ. When faced with this adversarial sample, Eve's classifier C fails and misclassifies the leakage trace as X'. In the third stage, Eve trains a new classifier C' using *defensive distillation* and *adversarial re-training* to protect it from misclassification cause by the adversarial perturbation δ. In the final stage, Alice first tests previously crafted adversarial samples against Eve's protected classifier C'. Then, Alice updates her adversarial sample crafting target to fool C' rather than the original classifier C.

We apply this methodology to a scenario where a malicious party trains a CNN to classify running processes using the HPC trace as the input. This information is extremely useful to the attacker since it helps to choose a specific attack or pick a vulnerable target among others. Once a a target is found, an attacker can perform micro-architectural or application specific attacks. To circumvent this information leakage and protect processes, the defender attempts to mask the process signature. Ideally, the masking is minimal and does not interfere with the running process.

Specifically, in our methodology, the attacker periodically collects 5 HPC values over 10 msec total with 10 usec intervals, resulting in total of 5000 data points per trace. Later, trace is fed into classical ML and DL classifiers. In this section, we explain our choice of the specific HPCs, the application classifier design and implementation details, the AL attacks applied to these classifiers and finally test the efficiency of adversarial defenses against our cloaking method.

3.1 HPC Profiling

HPCs are special purpose registers that provide detailed information on low-level hardware events in computer systems. These counters periodically count specified event like cache accesses, branches, TLB misses and many others. This information is intended to be used by developers and system administrators to monitor and fine-tune performance of applications. The availability of a specific counter depends on the architecture and model of the CPU. Among many available HPCs, we have selected the following 5 for the classification task;

(1) **Total Instructions:** the total number of retired i.e., executed and completed CPU instructions.
(2) **Branch Instructions:** the number of branch instructions (both taken and not taken).
(3) **Total Cache References:** the total number of L1, L2, and L3 cache hits and misses.
(4) **L1 Instruction Cache Miss:** the occurrence of L1 cache instruction cache misses.
(5) **L1 Data Cache Miss:** the occurrence of L1 cache data cache misses.

We have selected these HPCs to cover a wide variety of hardware events with both coarse and fine-grain information. For instance, the `Total Instructions` does not directly provide any information about the type of the instructions being executed. However, different instructions execute in varying number of cycles even if the data is loaded from the same cache level. This execution time difference translates indirectly into the total instructions executed in the given time period and hints about the instruction being executed.

`Branch Instructions` HPC provides valuable information about the execution flow as well. Whether the branches are taken or not taken, the total number of branches in the executed program remains constant for a given execution path. This constant in the leakage trace helps eliminate noise elements and increases classification accuracy. The `Total Cache References` HPC provides similar information to the Branch Instructions HPC in the sense that it does not leak information about the finer details like the specific cache set or even the cache level. However it carries information regarding the total memory access trace of the program. Regardless of the data being loaded from the CPU cache or the memory, the total number of cache references will remain the

same for a given process. The `L1 Instruction Cache Miss` and the `L1 Data Cache Miss` HPCs provide fine-grain information about the *Cold Start* misses on the L1 cache. Since the L1 cache is small, the data in this cache level is constantly replaced with new data, incrementing these counters. Moreover, separate counters for the instruction and the data misses allows the profiler to distinguish between arithmetic and memory intensive operations and increase accuracy. Finally, all five of the HPCs are interval counters meaning that they count specific hardware events within selected time periods.

3.2 Classifier Design and Implementation

In the first part of the study, we design and implement classifiers that can identify processes using the HPC leakage. To show the viability of such classifier, we chose 20 different ciphers from the OpenSSL 1.1.0 library as the classification target. Note that these classes include ciphers with both very similar and extremely different performance traces e.g.,AES-128, ECDSAB571, ECDSAP521, RC2 and RC2-CBC. Moreover, we also trained models to detect the version of the OpenSSL library for a given cipher. For this task, we used OpenSSL versions 0.9.8, 1.0.0, 1.0.1, 1.0.2 and 1.1.0.

3.2.1 ***Classical ML Classifiers:*** In this study, we refer to non-neural network classification methods as classical ML classifiers. In order to compare and contrast classical ML methods with CNNs, we trained a number of different classifiers using the Matlab Classification Learning Toolbox. The trained classifiers include SVMs, decision trees, kNNs and variety of ensemble methods.

3.2.2 ***Deep Learning Classifier:*** We designed and implemented the CNN classifier using Keras with Tensorflow-GPU back-end. The model has the total of 12 layers including the normalization and the dropout layers. In the input layer, the first convolution layer, there are a total of 5000 neurons to accommodate the 10 msec of leakage data with 5000 HPC data points. Since the network is moderately deep but extremely wide, we used 2 convolution and 2 MaxPool layers to reduce the number dimensions and extract meaningful feature representations from the raw trace.

In addition to convolution and MaxPool layers, we used batch normalization layers to normalize the data from different HPC traces. This is a crucial step since the hardware leakage trace is heavily dependent on the system load and scales with overall performance. Due to this dependency, the average execution time of a process or parts of a process can vary from one execution to another. Moreover, in the system-wide leakage collection scenario, the model would train over this system load when it should be treated as noise. If not handled properly, the noise and shifts in the time domain results in over-fitting the training data with the dominant average execution time, decreasing the classification rate. By using the batch normalization layer, the model learns the features within short time intervals and the relation between different HPC traces. Finally, the output layer has 20 neurons with softmax activation, each representing a classes of process. To train the model, we use *Categorical Cross-entropy* loss function with the *Adam Optimizer*.

3.3 Adversarial Learning Attacks

AL remains an important open research problem in AI. Traditionally, AL is used to fool AI classifiers and test model robustness against malicious inputs. In this study however, we propose to use AL as a defensive tool to mask the side-channel trace of applications and protect against micro-architectural attacks and privacy violations. In the following, we explain the specific adversarial attacks that we have used. We consider the following 10 attacks:

- **Additive Gaussian Noise Attack (AGNA):** Adds Gaussian Noise to the input trace to cause misclassification. The standard deviation of the noise is increased until the misclassification criteria is met. This AL attack method is ideal to be used in the cloaking defense due to the ease of implementation of the all-additive perturbations. A sister-process can actuate such additional changes in the side-channel trace by simply performing operations that increment specific counters like cache accesses or branch instructions.
- **Additive Uniform Noise Attack (AUNA):** Adds uniform noise to the input trace. The standard deviation of the noise is increased until the misclassification criteria is met. Like AGNA, AUNA is easy to implement as a sister-process due to its additive property.
- **Blended Uniform Noise Attack (BUNA):** Blends the input trace with Uniform Noise until the misclassification criteria is met.
- **Contrast Reduction Attack (CRA):** Calculates perturbations by reducing the 'contrast' of the input trace until a misclassification occurs. In case of the side-channel leakage trace, the attack smooths parts of the original trace and reduces the distance between the minimum and the maximum data points.
- **Gradient Attack (GA):** Creates a perturbation with the loss gradient with regards to the input trace. The magnitude of the added gradient is increased until the misclassification criteria is met. The attack only works when the model has a gradient.
- **Gaussian Blur Attack (GBA):** Adds Gaussian Blur to the input trace until a misclassification occurs. Gaussian blur smooths the input trace and reduces the amplitude of outliers. Moreover, this method reduces the resolution of the trace and cloaks fine-grain leakage.
- **GSA (Gradient Sign Attack) [19]:** Also called the Fast Gradient Sign Method, the attack has been proposed by Goodfellow et al. in 2014. GSA works by adding the sign of the elements of the gradient of the cost function with regards to the input trace. The gradient sign is then multiplied with a small constant that is increased until a misclassification occurs.

- **L-BFGS-B Attack (LBFGSA) [61]:** The attack utilizes the modified Broyden-Fletcher-Goldfarb-Shanno algorithm, an iterative method for solving unconstrained nonlinear optimization problems, to craft perturbations that have minimal distance to the original trace. The attack morphs the input to a specific class. However, in our experiments, we did not target a specific class and chose random classes as the target.
- **Saliency Map Attack (SMA) [52]:** Works by calculating the forward derivative of the model to build an adversarial saliency map to detect which input features e.g.,pixels in an image, have a stronger effect on the targeted misclassification. Using this information, an adversary can modify only the features with high impact on the output and produce smaller perturbations.
- **Salt and Pepper Noise Attack (SPNA):** Works by adding Salt and Pepper noise (also called impulse noise) to the input trace until a misclassification occurs. For images, salt and pepper values correspond to white and black pixels respectively. For the side-channel leakage trace however, these values correspond to the upper and the lower bounds in the trace.

3.4 Adversarial Learning Defenses

In order to see the viability of any defense method that can be used by an attacker against our adversarial perturbations, we have explored two methods: *adversarial re-training* and *defensive distillation* (DD). These defenses are an integral part of this study since an attacker capable of overcoming adversarial perturbation would deem any cloaking mechanism moot.

3.4.1 ***Gradient Masking:*** The term gradient masking defense has been introduced in [49] to represent group of defense methods against adversarial samples. The defense works by hiding the gradient information from the attacker to prevent it from crafting adversarial samples. Papernot et al. [49] however showed that the method fail under the oracle access scenario. An attacker can query the classifier with enough samples to create a cloned classifier. Since the clone and the original classifiers have correlated gradients, the attacker can use the gradient from the clone and craft adversarial samples, bypassing the defense. Due to the known weaknesses and limitations of this defense method, we do not further investigate it in this study.

3.4.2 ***Adversarial Re-training:*** This defense idea was first proposed by Szegedy et al. in 2013 [60]. Later in 2014, Goodfellow et al. [19] improved the practicality of the method by showing how to craft adversarial samples efficiently using the Fast Gradient Sign Method. In this defense, the model is re-trained using adversarial samples. By doing so, the model is 'vaccinated' against adversarial perturbations and can correctly classify them. In other words, the method aims to teach adversarial perturbations to the model so that it can generalize better and not be fooled by small perturbations. While this method works successfully against a specific type of attack, it has been shown to fail against attack methods that the model was not trained for. Nevertheless, we apply this defense method to our classifiers and investigate its applicability to side-channel leakage classifiers.

3.4.3 ***Defensive Distillation:*** The DD has been proposed by Papernot et al. [48] in 2016 to protect DL models against AL attacks. The goal of this technique is to increase the entropy of the prediction vector to protect the model from being easily fooled. The method works by pre-training a model with a custom output layer. Normally, the softmax temperature is set to be as small as possible to train a tightly fitted, highly accurate model. In the custom layer however, the temperature value is set to a higher value to distill the probability outputs. The first model is trained with the training data using hard labels i.e., the correct class label is set to '1' and all other class labels are set to '0'. After the model is trained, the training samples are fed into it and the probability outputs are recorded as *soft labels*. Then these soft labels are used to train the second, distilled model with the same training data. This process smooths the model surface on directions that an adversary would use to craft perturbations. This smoothing process increases the perturbation size required to craft an adversarial samples and invalidates some of the previously crafted adversarial samples. This smoothing can be set to different levels by adjusting the temperature value. Note that however, the DD can reduce the classification accuracy significantly if the temperature value is set too high.

4 EXPERIMENT SETUP AND RESULTS

In this section, we give details of our experiment setup, and the results of different adversarial attacks on the crypto-process classifier, and finally present the results of hardened adversarially re-trained and distilled models.

Experiment Setup: DL models perform large amounts of matrix multiplications that can run efficiently in modern GPU systems. For that reason, we used a workstation with two Nvidia 1080Ti (Pascal architecture) GPUs, 20-core Intel i7-7900X CPU and 64 GB of RAM. On the software side, the classifier model is coded using *Keras v2.1.3* with *Tensorflow-GPU v1.4.1* back-end and other Python3 packages such as *Numpy v1.14.0, Pandas, Sci-kit, H5py etc.*

The HPC data is collected from a server with Intel Xeon E5-2670 v2 CPU running Ubuntu 16 LTS. The CPU has 85 HPCs of which 50 are accessible from user-space. To access the HPCs, we had the choice of using *Perf* and *PAPI* libraries. Due to the lower sampling rate of Perf, we chose to use the PAPI with *QuickHPC* [13] front-end. QuickHPC is a tool developed by Marco Chiappetta to collect high-resolution HPC data using the PAPI back-end. It is over 30000 times faster than *perf-stat* and provides an easy to use interface.

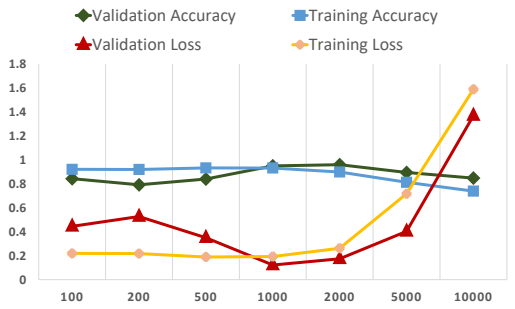

Figure 2: Results for the CNN classifier trained using varying number of features. Models reach highest validation accuracy with 1000 and 2000 features.

4.1 Classification Results

The classifiers are trained to identify 20 classes representing a diverse set of different ciphers of five different versions of OpenSSL, as detailed in Section 3.2. For training, we split our dataset into three parts as training (%60), validation (%20) and test (%20).

4.1.1 ***CNN Classifier:*** For the CNN classifier, we firstly investigated the effect of the number of HPCs collected and trained our models for 100 epochs with data from a varying number of HPCs. Not surprisingly, even using a single HPC, our CNN classifier achieved 81% validation accuracy by training more epochs. Moreover, we noticed that after the 30th epoch, the model overfitted the training data i.e., the validation accuracy started to drop while the training accuracy kept increasing. When we increased the number of HPCs collected, our models became much more accurate and achieved over 99% validation accuracy as seen in Figure 3. Moreover, when we use the data from all 5 HPCs, our model achieved 99.8% validation accuracy in less than 20 epochs. While our validation accuracy saturates even with only 2 HPCs, *Total Instructions* and *Branch Instructions* we have decided to use all 5 of them. We made this decision because in a real-world attack scenario, an attacker might be using any one or more of the HPCs. Since it would not be known which specific hardware event(s) an attacker would monitor, we decided to use all 5, monitoring different low level hardware events to provide a comprehensive cloaking coverage. To find the optimum number of features per HPC, we have trained multiple models with using various number of features. As shown in Figure 2, the validation accuracy saturates at 1000 and 2000 features, validation loss drops after 1000 features. For this reason, we chose to use 1000 features for our experiments.

Further, we investigated how the number of training samples affect the validation accuracy. For that, we have trained 6 models with a varying number of training samples. For the first model, we have used only 100 samples per class (2000 samples in total) and later on trained models with 300, 1000, 3000, 10000 and 30000 samples per class. In the first model, we achieved 99.8% validation accuracy after 40 epochs of training. When we trained models with more data, we have reached similar accuracy levels in much fewer epochs. To make a good trade-off between the dataset size and training time, we have opted to use 1000 samples per class. This model reached 100% accuracy with 20 epochs of training as shown in Figure 4. Finally, our last model achieved 100% accuracy just after 4 epochs when trained with 30000 samples per class.

Table 1: Application classification results for the classical ML classifiers with and without PCA feature reduction.

Classification Method	Without PCA	With PCA (99.5% variance)
Fine Tree	98.7	99.9
Medium Tree	85.4	94.8
Coarse Tree	24.9	25
Linear Discriminant	99.6	99.7
Quadratic Discriminant	N/A	99.4
Linear SVM	99.9	98.2
Quadratic SVM	99.9	96.9
Cubic SVM	99.9	94.3
Fine Gaussian SVM	40	88.2
Medium Gaussian SVM	98.3	92.1
Coarse Gaussian SVM	99.7	13.4
Fine kNN	96.8	11.1
Medium kNN	94.9	7.8
Coarse kNN	85.5	5.2
Cosine kNN	92.5	19.6
Cubic kNN	85.2	7.7
Weighted kNN	95.9	8.3
Boosted Trees	99.2	99.8
Bagged Trees	99.9	94.8
Subspace Discriminant	99.8	99.7
Subspace kNN	84.8	88.1
RUSBoosted Trees	76	92.8
Best	**99.9**	**99.9**

We also show that in addition to detecting the process type, we can also distinguish between different versions of OpenSSL. For each of the 20 analyzed ciphers, we built classifiers to identify the library version. Figure 5 presents the classification results of two models trained using 1 and 5 HPC traces respectively. As cipher updates between versions can be very small, the added information from sampling several HPCs is essential for high classification rates, as shown in the results.

4.1.2 ***Classical ML Methods:*** In our training of ML classifiers, the first challenge was the fact that the side-channel leakage data is extremely wide. We have chosen to train our models using 1000 data points per HPC with 5 HPCs monitored simultaneously. This parameter selection is done empirically to provide wide cloaking coverage and train highly accurate models as explained in Section 4.1.1. Using 1000 data points with 10 usec intervals per HPC allowed us to obtain high quality data in a short observation window. Nevertheless, 5000 dimensions is unusually high for classifiers, especially considering that we are training multi-class classifiers with 20 possible classes.

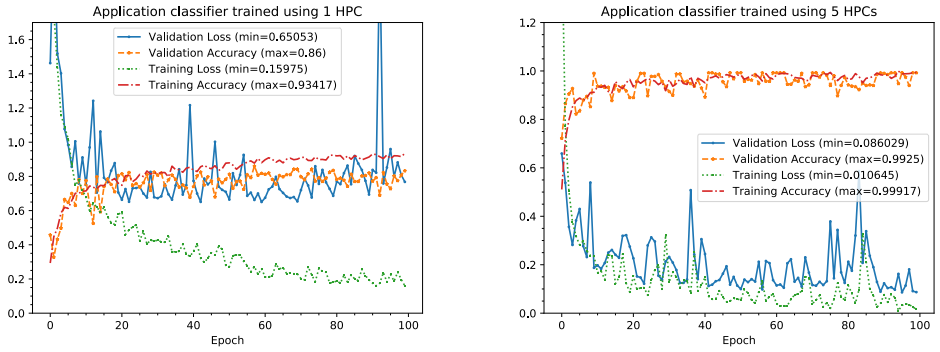

Figure 3: Classification accuracy of models trained using 1 and 5 HPCs. Even using data from a single HPC trace is enough to obtain high accuracy top-1 classification rates, albeit taking longer to train.

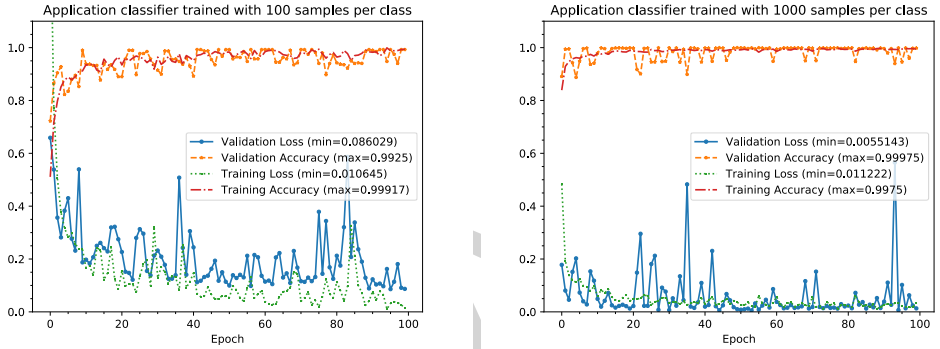

Figure 4: Classification accuracy of models trained using 100 and 1000 samples per class. Both models reach 99% accuracy in 20 and 40 epochs of training respectively.

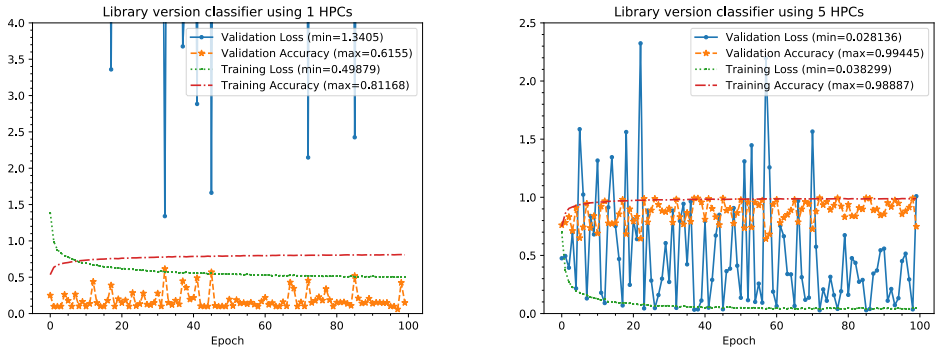

Figure 5: Library version classifier accuracy for models trained using 1 and 5 HPCs. Accuracy of the former model saturate at 61% while the latter reach 99%.

In order to find optimal settings for the hardware leakage trace, we tried different parameters with each classifier. For instance in the case of decision trees, we have trained the 'Fine Tree', 'Medium Tree' and 'Coarse Tree' classifiers. The difference between these classifiers is that respectively they allow 5, 20, and 100 splits (leaves in the decision tree) to better distinguish between classes. For the case of Gaussian SVM, fine, medium and coarse refers to the kernel scale set to sqrt(P)/4, sqrt(P) and sqrt(P)*4 respectively. As for the kNN, the parameters refer to the number of neighbors and the different distance metrics.

Results for the classical ML classifiers are given in Table 1. Classic ML algorithms achieve very high success rates for the given classification task. The trained models can in fact classify running processes by using their HPC traces. Note that the Quadratic Discriminant did not converge to a solution without the PCA application hence no score is given in the table.

4.2 AL on the Unprotected Model

Next, we crafted adversarial perturbations for the unprotected classifiers by using and adapting the publicly available Foolbox [56] library to our scenario. The library provides numerous adversarial attacks and provides an easy to use API. For a selected attack, Foolbox crafts necessary perturbations on a given sample and classifier model pair to 'fool' the given model. Detailed information about these attacks can be found in Section 3.3.

Table 3 presents the classification accuracy of perturbed samples. As the results show, almost all test samples are misclassified by the classifier model with very high accuracy at over 86%. Another important metric for the adversarial learning is the Mean Absolute Distance (MAD) and the Mean Squared Distance (MSD) of the perturbed traces from the originals. These metrics quantify the size of the changes i.e., perturbations made to the original traces by various adversarial attack methods. The difference between the MAD and the MSD is that, the latter is more sensitive to the larger changes due to the square operation. For instance, if an adversarial perturbation requires a significant change in 1 sample point among the 5000 features, it will have a stronger impact in the final MSD value than average change distributed over few points. MAD however is more dependent on the overall change in the trace, i.e., all 5000 sample points have the same impact on the final distance. Our results show that with most adversarial attacks, perturbation MAD is around or well below 1% and within the ideal range.

4.3 AL on the Hardened Models

Here we present the results of the AL attacks on the hardened classifier models. As explained in Section 3, we wanted to test the robustness of our cloaking mechanism against classifiers hardened with Adversarial Re-training and DD. To recap the scenario, Alice the defender wants to cloak her process by adding perturbations to her execution trace so that eavesdropper Eve cannot correctly classify what Alice is running. Then Eve notices or predicts the use of adversarial perturbations on the data and hardens her classifier model against AL attacks using adversarial re-training and DD.

In order to test the attack scenario on hardened models, we first craft 100,000 adversarial samples per adversarial attack against the unprotected classifier. Then we harden the classifier with the aforementioned defense methods and feed the adversarial samples. Here, we aim to measure the level of protection provided by the adversarial re-training and the DD methods. As presented in Table 2, the application of both the adversarial re-training and the DD invalidates some portion of the previously crafted adversarial samples. For the adversarial re-training, the success rate varies between 99% (GSA) and 4% (SPNA). In other words, 99% of the adversarial samples crafted using GSA against the unprotected model are invalid on the hardened model. As for the DD, we see similar rates of protection ranging from 61% up to 100% for old perturbations. Impressively, 100% of the adversarial samples crafted using the BUNA are now ineffective against the model trained with DD at temperature T=100. In short, by using the adversarial re-training or the DD, Eve can indeed harden her classifier against AL. However, keep in mind that Alice can observe or predict this behavior and introduce new adversarial samples targeting the hardened models.

4.3.1 ***Adversarial Re-training:*** After training the DL classifier model and crafting adversarial samples, we use these perturbations as training data and re-train the classifier. The motivation here is to teach the classifier model to detect these perturbed samples and correctly classify them. With this re-training stage, we expect to see whether we can 'immunize' the classifier model against given adversarial attacks. However, as the results in Table 3 show, all of the adversarial attacks still succeed albeit requiring marginally larger perturbations. Moreover, while we observe a drop in the misclassification confidence, it is still quite high at over 63% i.e., Eve's classifier can be fooled by adversarial samples.

4.3.2 ***Defensive Distillation:*** We have used the technique proposed in [51] and trained hardened models with DD at various temperatures ranging from 1 to 100. Our results show that, even if the eavesdropper Eve hardens her model with DD, the trained model is still prone to adversarial attacks albeit requiring larger perturbations in some cases. In Table 4, we present the MAD i.e., the perturbation size, of various attack methods on both unprotected and hardened models. Our results show that the application of DD indeed offers a certain level of hardening to the model and increases the perturbation sizes. However this behavior is erratic compared to the adversarial re-training defense i.e., the MAD is significantly higher at some temperatures while much smaller for others. For instance, the MAD for the AUNA perturbations against the unprotected model is 0.00292 in average for 100,000 adversarial samples. The perturbation size for the same attack drops to 0.00033 when the distillation defense is applied with temperature T=1. This in turn practically makes Eve's classifier model easier to fool. Same behavior is observed with the adversarial samples crafted using AGNA and GBA as well. For the cases that the DD actually hardens Eve's model against adversarial samples, the MAD is still minimal. Hence, Alice can still successfully craft adversarial samples with minimal perturbations and fool Eve's model.

Table 2: Effectiveness of adversarial re-training and DD on 100,000 previously crafted adversarial samples. The results show what percentage of previously successful adversarial samples are ineffective on the hardened models.

Adversarial Attack	Adversarial Re-training	DD with T=1	DD with T=2	DD with T=5	DD with T=10	DD with T=20	DD with T=30	DD with T=40	DD with T=50	DD with T=100
AGNA	42	77	60	70	83	83	63	64	62	75
AUNA	43	77	60	70	83	82	63	65	61	75
BUNA	94	92	92	91	94	94	94	96	94	100
CRA	94	95	99	94	94	94	94	99	88	95
GA	97	99	72	83	99	99	96	80	90	90
GBA	84	83	84	88	82	94	93	91	93	88
GSA	99	91	90	91	99	99	99	95	99	98
LBFGSA	51	76	63	63	78	87	65	65	63	71
SMA	26	71	50	52	62	82	49	47	32	48
SPNA	4	84	76	78	94	93	76	79	73	80

Table 3: Perturbation results against both the unprotected and the hardened (Adversarial Re-training) CNN classifier. The new adversarial samples have up to 29% lower misclassification confidence compared to the unprotected model. However, adversarial samples are still misclassified with quite high confidence values in the range of 63-98%.

	Unprotected Classifier			Hardened Classifier (Adv. Re-training)		
Adversarial Attack	Original Sample Classification Confidence	Adversarial MisClassification Confidence	Perturbation Size (MAD)	Original Sample Classification Confidence	Adversarial MisClassification Confidence	Perturbation Size (MAD)
AGNA	99	96	0.00294	92	82	0.00294
AUNA	99	97	0.00292	91	82	0.00332
BUNA	99	99	0.05000	96	93	0.05000
CRA	99	99	0.05254	97	98	0.04999
GA	99	99	0.00250	88	97	0.00398
GBA	99	97	0.00080	93	74	0.00071
GSA	99	99	0.00499	89	97	0.00596
LBFGSA	99	86	0.00025	89	72	0.00031
SMA	99	92	0.00001	88	63	0.00008
SPNA	99	96	0.01528	92	74	0.08268

Table 4: MAD sizes of adversarial perturbations crafted against the DD applied classifier. Application of the adversarial re-training and DD only marginally increases the perturbation size needed to fool the classifier in most cases.

Adversarial Attack	Unprotected Model	Adversarial Re-trained	DD with T=1	DD with T=2	DD with T=5	DD with T=10	DD with T=20	DD with T=30	DD with T=40	DD with T=50	DD with T=100
AGNA	0.00294	0.00294	0.00035	0.00265	0.00062	0.01059	0.01389	0.00452	0.00766	0.01797	0.00431
AUNA	0.00292	0.00332	0.00033	0.00238	0.00071	0.01114	0.01336	0.00514	0.00948	0.01787	0.00451
BUNA	0.05000	0.05000	0.04989	0.05301	0.08198	0.10293	7.52282	0.05006	0.05002	0.16376	0.07748
CRA	0.05254	0.04999	0.04999	0.10247	0.08548	0.10780	NA	0.04998	1.44158	0.27992	0.07448
GA	0.00250	0.00398	0.00355	0.00283	0.10224	0.00783	0.00364	0.00860	2.75805	0.00849	0.00303
GBA	0.00080	0.00071	0.00058	0.00092	0.00060	0.04992	NA	0.00169	0.00046	0.00062	0.00083
GSA	0.00499	0.00596	0.00504	0.00534	0.13665	0.02482	0.00540	0.01064	0.04121	0.01670	0.00550
LBFGSA	0.00025	0.00031	0.00533	0.00178	0.00024	0.00210	0.07670	0.00088	0.02872	0.00521	0.00920
SMA	0.00001	0.00008	0.00003	0.00012	0.00007	NA	NA	0.00001	0.00001	NA	NA
SPNA	0.01528	0.08268	0.01060	0.00940	0.00360	0.01804	0.02000	0.00260	0.02000	0.02000	0.01980

Finally, NA values in Table 4 represent cases where the model had very low classification accuracy and could not correctly classify original samples.

5 CONCLUSION

Side-channel leakage on shared hardware systems pose a real and present danger to the security and the privacy of users. Even when the software is perfectly isolated, co-resident tenants still share the underlying hardware and are prone to side-channel attacks. Especially considering the wide adoption of AI across many disciplines, it is not surprising that such attacks will become automated and even easier to perform in the future. There is a clear need for users to cloak their execution fingerprints from the underlying shared system.

With this work we took a first step in this direction. Specifically, by making clever defensive use of adversarial crafting we introduced a new cloaking defense against the side-channel leakage classifiers. We first demonstrated the threat side-channel leakage poses by processing leakage profiles to yield highly accurate AI models which may be used by an adversary to violate privacy and security policies of applications. We trained various types of classifiers including the classical ML methods and showed how the parameter selection affects the learning rate and the validation accuracy. While this is a strong

threat to shared hardware systems, we showed that it can be mitigated using carefully crafted adversarial samples. Moreover, we investigated defenses that can potentially help an attack to bypass the adversarial samples. Our results show that even in the presence of defensive distillation and adversarial re-training, the defender can craft working adversarial samples and fool the attacker. These perturbations can be implemented as a sister-process that will run side-by-side with the original and easily cause misclassification to the attacker's model without any significant overhead. Using the adversarial crafting-based cloaking mechanism that we have outlined in this work, users can enable such services on-demand for sensitive operations. Efficient design and implementation of such defenses for shared hardware systems like cloud remains an open research problem. Finally, to the best of our knowledge, this work is the first use of adversarial crafting for defensive purposes. We envision the same approach to be useful in other application scenarios.

REFERENCES

[1] Alam, M., Bhattacharya, S., Mukhopadhyay, D., and Bhattacharya, S. Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks. Cryptology ePrint Archive, Report 2017/564, 2017. https://eprint.iacr.org/2017/564.

[2] Allaf, Z., Adda, M., and Gegov, A. A comparison study on flush+reload and prime+probe attacks on aes using machine learning approaches. In *Advances in Computational Intelligence Systems* (Cham, 2018), F. Chao, S. Schockaert, and Q. Zhang, Eds., Springer International Publishing, pp. 203–213.

[3] Assael, Y. M., Shillingford, B., Whiteson, S., and de Freitas, N. Lipnet: Sentence-level lipreading. *arXiv preprint arXiv:1611.01599* (2016).

[4] Beltramelli, T., and Risi, S. Deep-spying: Spying using smartwatch and deep learning. *arXiv preprint arXiv:1512.05616* (2015).

[5] Bernstein, D. J. Cache-timing attacks on AES, 2004. URL: `http://cr.yp.to/papers.html#cachetiming`.

[6] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases* (2013), Springer, pp. 387–402.

[7] Biggio, B., Fumera, G., and Roli, F. Security evaluation of pattern classifiers under attack. *IEEE transactions on knowledge and data engineering 26*, 4 (2014), 984–996.

[8] Biggio, B., Nelson, B., and Laskov, P. Support vector machines under adversarial label noise. In *Asian Conference on Machine Learning* (2011), pp. 97–112.

[9] Biggio, B., Nelson, B., and Laskov, P. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389* (2012).

[10] Briongos, S., Irazoqui, G., Malagón, P., and Eisenbarth, T. Cacheshield: Protecting legacy processes against cache attacks. *arXiv preprint arXiv:1709.01795* (2017).

[11] Briongos, S., Irazoqui, G., Malagón, P., and Eisenbarth, T. Cacheshield: Detecting cache attacks through self-observation. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy* (New York, NY, USA, 2018), CODASPY '18, ACM, pp. 224–235.

[12] Carlini, N., and Wagner, D. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on* (2017), IEEE, pp. 39–57.

[13] Chiappetta, M. Quickhpc. https://github.com/chpmrc/quickhpc, 2015.

[14] Dang, H., Huang, Y., and Chang, E.-C. Evading classifiers by morphing in the dark. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 119–133.

[15] Emmanuel, Z. Security experts air concerns over hackers using AI and machine learning for phishing attacks. https://www.computerweekly.com/news/450427653/Security-experts-air-concerns-over-hackers-using-AI-and-machine-learning-for-phishing-atttacks/.

[16] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Song, D., Kohno, T., Rahmati, A., Prakash, A., and Tramer, F. Note on attacking object detectors with adversarial stickers. *arXiv preprint arXiv:1712.08062* (2017).

[17] Foreman, J. C. A survey of cyber security countermeasures using hardware performance counters. *arXiv preprint arXiv:1807.10868* (2018).

[18] Goodfellow, I., and Papernot, N. Is attacking machine learning easier than defending it? http://www.cleverhans.io/security/privacy/ml/2017/02/15/why-attacking-machine-learning-is-easier-than-defending-it.html.

[19] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[20] Gordon, R. AI beats pros at Super Smash Bros. http://www.csail.mit.edu/ai_beats_pros_at_super_smash_bros. Accessed: 2017-10-27.

[21] Gruss, D., Maurice, C., Wagner, K., and Mangard, S. Flush+ flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2016), Springer, pp. 279–299.

[22] Gruss, D., Spreitzer, R., and Mangard, S. Cache template attacks: automating attacks on inclusive last-level caches. In *Proceedings of the 24th USENIX Conference on Security Symposium* (2015), USENIX Association, pp. 897–912.

[23] Gullasch, D., Bangerter, E., and Krenn, S. Cache games–bringing access-based cache attacks on AES to practice. In *Security and Privacy (SP), 2011 IEEE Symposium on* (2011), IEEE.

[24] Gülmezoglu, B., Zankl, A., Eisenbarth, T., and Sunar, B. PerfWeb: How to Violate Web Privacy with Hardware Performance Events. In *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II* (2017), pp. 80–97.

[25] Gulshan, V., Peng, L., Coram, M., Stumpe, M. C., Wu, D., Narayanaswamy, A., Venugopalan, S., Widner, K., Madams, T., Cuadros, J., et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama 316*, 22 (2016).

[26] Hadad, Y. 30 amazing applications of deep learning. http://www.yaronhadad.com/deep-learning-most-amazing-applications/, Aug 2017. Accessed: 2017-10-27.

[27] Heuser, A., and Zohner, M. Intelligent machine homicide. *COSADE 7275* (2012), 249–264.

[28] Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., and Vandewalle, J. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering 1*, 4 (2011), 293.

[29] Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., and Tygar, J. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence* (2011), ACM, pp. 43–58.

[30] Inci, M. S., Gulmezoglu, B., Irazoqui, G., Eisenbarth, T., and Sunar, B. Cache attacks enable bulk key recovery on the cloud. In *International Conference on Cryptographic Hardware and Embedded Systems—CHES* (2016), Springer Berlin Heidelberg, pp. 368–388.

[31] Irazoqui, G., Inci, M. S., Eisenbarth, T., and Sunar, B. Wait a minute! a fast, cross-vm attack on aes. In *International Workshop on Recent Advances in Intrusion Detection* (2014), Springer, pp. 299–319.

[32] Jon Brodkin. 2 million people-and some dead ones-were impersonated in net neutrality comments. https://arstechnica.com/tech-policy/2017/12/dead-people-among-millions-impersonated-in-fake-net-neutrality-comments/.

[33] Keren Elazari. Hackers are on the brink of launching a wave of AI attacks. http://www.wired.co.uk/article/hackers-ai-cyberattack-offensive.

[34] Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236* (2016).

[35] Laskov, P., and Lippmann, R. Machine learning in adversarial environments, 2010.

[36] Lipp, M., Gruss, D., Spreitzer, R., Maurice, C., and Mangard, S. Armageddon: Cache attacks on mobile devices. In *25th USENIX Security Symposium (USENIX Security 16)* (2016), USENIX Association, pp. 235–252.

[37] Liu, Y., Chen, X., Liu, C., and Song, D. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770* (2016).

[38] Liu, Y., Gadepalli, K., Norouzi, M., Dahl, G. E., Kohlberger, T., Boyko, A., Venugopalan, S., Timofeev, A., Nelson, P. Q., Corrado, G. S., et al. Detecting cancer metastases on gigapixel pathology images. *arXiv preprint arXiv:1703.02442* (2017).

[39] 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.

[40] MacDonald, F. A Deep Learning Machine Just Beat Humans in an IQ Test. https://www.sciencealert.com/a-deep-learning-machine-just-beat-humans-in-an-iq-test.

[41] Maghrebi, H., Portigliatti, T., and Prouff, E. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering* (2016), Springer, pp. 3–26.

[42] Martinasek, Z., Hajny, J., and Malina, L. Optimization of power analysis using neural network. In *International Conference on Smart Card Research and Advanced Applications* (2013), Springer, pp. 94–107.

[43] Martinasek, Z., and Zeman, V. Innovative method of the power analysis. *Radioengineering 22*, 2 (2013), 586–594.

[44] Meng, D., and Chen, H. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 135–147.

[45] O'Brien, E. Romance novels, generated by artificial intelligence. https://medium.com/towards-data-science/romance-novels-generated-by-artificial-intelligence-1b31d9c872b2, Aug 2017.

[46] Oh, S. J., Fritz, M., and Schiele, B. Adversarial image perturbation for privacy protection a game theory perspective. In *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), IEEE, pp. 1491–1500.

[47] Osvik, D. A., Shamir, A., and Tromer, E. Cache attacks and countermeasures: the case of AES. In *Cryptographers Track at the RSA Conference* (2006), Springer, pp. 1–20.

[48] Papernot, N., McDaniel, P., and Goodfellow, I. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277* (2016).

[49] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (2017), ACM, pp. 506–519.

[50] 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.

[51] Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on* (2016), IEEE, pp. 582–597.

[52] Papernot, N., McDaniel, P. D., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The limitations of deep learning in adversarial settings. *CoRR abs/1511.07528* (2015).

[53] Peng, P., Yuan, Q., Wen, Y., Yang, Y., Tang, Z., Long, H., and Wang, J. Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games. *arXiv preprint arXiv:1703.10069* (2017).

[54] Percival, C. Cache missing for fun and profit, 2005.

[55] Pessl, P., Gruss, D., Maurice, C., Schwarz, M., and Mangard, S. Drama: Exploiting dram addressing for cross-cpu attacks. In *USENIX Security Symposium* (2016), pp. 565–581.

[56] Rauber, J., Brendel, W., and Bethge, M. Foolbox v0.8.0: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131* (2017).

[57] Simonite, T. This AI Will Craft Tweets That Youll Never Know Are Spam. https://www.technologyreview.com/s/602109/this-ai-will-craft-tweets-that-youll-never-know-are-spam/.

[58] Su, J., Vargas, D. V., and Kouichi, S. One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864* (2017).

[59] Susan Decker. FCC Rules Out Delaying Net Neutrality Repeal Over Fake Comments. https://www.bloomberg.com/news/articles/2018-01-05/fcc-rules-out-delaying-net-neutrality-repeal-over-fake-comments.

[60] 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).

[61] Tabacof, P., and Valle, E. Exploring the space of adversarial images. *CoRR abs/1510.05328* (2015).

[62] The Cylance Team. Black Hat Attendees See AI as Double-Edged Sword. https://threatmatrix.cylance.com/en_us/home/black-hat-attendees-see-ai-as-double-edged-sword.html.

[63] Tramèr, F., Kurakin, A., Papernot, N., Boneh, D., and McDaniel, P. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204* (2017).

[64] Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., and Zweig, G. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256* (2016).

[65] Yarom, Y., and Falkner, K. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *(USENIX Security 2014)* (2014).

[66] Yu, K.-H., Zhang, C., Berry, G. J., Altman, R. B., Ré, C., Rubin, D. L., and Snyder, M. Predicting non-small cell lung cancer prognosis by fully automated microscopic pathology image features. *Nature communications 7* (2016).

[67] Zhang, T., Zhang, Y., and Lee, R. B. Cloudradar: A real-time side-channel attack detection system in clouds. In *International Symposium on Research in Attacks, Intrusions, and Defenses* (2016), Springer.

[68] Zhang, X., Hamm, J., Reiter, M. K., and Zhang, Y. Statistical privacy for streaming traffic. In *NDSS* (2019).

[69] Zhang, Y., Juels, A., Reiter, M. K., and Ristenpart, T. Cross-tenant side-channel attacks in PaaS clouds. In *CCS* (2014), pp. 990–1003.

[70] Zhou, Y., Kantarcioglu, M., Thuraisingham, B., and Xi, B. Adversarial support vector machine learning. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining* (2012), ACM, pp. 1059–1067.

Traffic Generation using Containerization for Machine Learning

Henry Clausen
henry.clausen@ed.ac.uk
University of Edinburgh
Edinburgh, UK

Robert Flood
s1784464@ed.ac.uk
University of Edinburgh
Edinburgh, UK

David Aspinall
University of Edinburgh
Edinburgh, UK
The Alan Turing Institute
London, UK

ABSTRACT

The design and evaluation of data-driven network intrusion detection methods are currently held back by a lack of adequate data, both in terms of benign and attack traffic. Existing datasets are mostly gathered in isolated lab environments containing virtual machines, to both offer more control over the computer interactions and prevent any malicious code from escaping. This procedure however leads to datasets that lack four core properties: heterogeneity, ground truth traffic labels, large data size, and contemporary content. Here, we present a novel data generation framework based on Docker containers that addresses these problems systematically. For this, we arrange suitable containers into relevant traffic communication scenarios and subscenarios, which are subject to appropriate input randomization as well as WAN emulation. By relying on process isolation through containerization, we can match traffic events with individual processes, and achieve scalability and modularity of individual traffic scenarios. We perform two experiments to assess the reproducability and traffic properties of our framework, and demonstrate the usefulness of our framework on a traffic classification example.

KEYWORDS

Network security, datasets, machine learning, intrusion detection

ACM Reference Format:
Henry Clausen, Robert Flood, and David Aspinall. 2019. Traffic Generation using Containerization for Machine Learning. In *DYNAMICS '19: DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security Workshop, December 09–10, 2019, San Juan, PR.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The exponential growth of data availability enabled the machine learning revolution of this decade that transformed many areas of our lives. Ironically, security oriented data describing computer networks is notoriously hard to obtain, and researchers struggle to evaluate new network intrusion detection systems (NIDS) or similar tools on suitable network traffic data. Well-designed datasets are such a rarity that researchers often rely on datasets that are well over a decade old [18, 35], calling into question their effectiveness on modern traffic and attacks. The lack of quantity, variability, meaningful labels, and ground truth has so far slowed scientific progress and objective and appropriate measurements on ML-based network security methods.

Privacy and security concerns discourage network administrators to release rich and realistic datasets for the public. Network traffic produced by individuals contains a mass of sensitive, personal information, such as passwords, email addresses, or usage habits, requiring researchers to expend effort anonymising the dataset [24]. To examine malicious behaviour, researchers are often forced to build artificial datasets using isolated machines in a laboratory setting to avoid damaging operational devices. Background traffic is usually generated in real-time from scripts executed on the virtual machine, which constrains both the amount and heterogeneity of the data. Current experiments show that it is possible to use large network of virtual machines for traffic generation [7, 8], however we did not find any publicly available dataset of that category.

Existing network intrusion detection datasets are predominantly designed to support a broad range of applications, and are collected in a static manner, unable to be modified or expanded. This proves to be a serious defect as the ecosystem of intrusions is continually evolving. Furthermore, it prohibits a more detailed analysis of specific areas of network traffic due to the available data only being a fraction of the original dataset. To combat this, new datasets must be periodically built from scratch.

Allowing researchers to create datasets dynamically to circumvent these issues would be extremely beneficial. We propose a such framework, based on application containers using Docker [13]. Docker is a service for developing and monitoring containers, also known as OS-level virtual machines. By moving from virtual machines to containers, we enable the scalable, modular, and dynamic creation of network traffic datasets. Since Docker containers can be arranged in complex settings with a few commands, it is a lot easier with containers to script a variety of network activities thus increase the heterogeneity and realism of the generated data.

Furthermore, each Docker container is highly specialized in its purpose, generating traffic related to only a single application process. Therefore, by scripting a variety of Docker-based *scenarios* that simulate benign or malicious behaviours and collecting the resultant traffic, we can build a dataset with perfect ground truth, something that has so far not been possible for network traffic.

Finally, the most import reason to rely on Docker for containerization is that many containerized applications are shared on the Docker Hub platform.

This work provides the following contributions:

(1) We present a novel network traffic generation framework that is designed to improve several shortcomings of current datasets for NIDS evaluation. This framework is openly

accessible for researchers and allows for straightforward customization.
(2) We define four new requirements a network intrusion dataset should fulfil in order to be suitable to train machine-learning based intrusion detection methods.
(3) We perform a number of experiments to demonstrate the suitability and utility of our framework.

1.1 Outline

The remainder of the paper is organized as follows. Section 2 discusses existing NIDS datasets and the problems that arise during their usage as well as background information about network traffic data formats and virtualization methods. The section concludes with a set of requirements we propose to improve the training and evaluation of machine-learning-based methods. Section 3 describes the general design of our framework, and how it improves on the discussed problems in existing datasets. We also discuss a specific example in detail. Section 4 discusses several experiments to validate the improvements and utility our framework provides. Section 5 concludes the results and discusses limitations of our work and directions for future work.

2 BACKGROUND

2.1 Data formats

Computers in a network communicate by sending *network packets* to each other, which are split into the control information, also called packet header, and the user information, called payload. The payload of a packet in general carries the information on behalf of an application and can be encrypted, while the header contains the necessary information for the correct transmission of the packet, including the transmission protocol layer, IP addresses, etc. Methods using packet-level input can be divided into payload inspection, header-based, or hybrid. Packets are usually stored in the widespread *pcap* format.

The majority of packets are exchanged between two hosts within bidirectional *connections*. Another common format of network traffic information is based on connection summaries, also called *network flows*. RFC 3697 [5] defines a network flow as a sequence of packets that share the same source and destination IP address, IP protocol, and for TCP and UDP connections the same source and destination port. A network flow is usually represented by this information along with additional information such as the start and duration of the connection as well as the total number of packets and bytes transferred.

2.2 Related work and existing datasets

To evaluate their ability to model the behaviour of a network and to identify malicious activity and network intrusions, new methods have to be tested using existing datasets of network traffic. This network should ideally contain realistic and representative benign network traffic as well as a variety of different network intrusions. However, as network traffic contains a vast amount of information about a network and its users, it is notoriously difficult to release a comprehensive dataset without infringing the privacy rights of the network users [34]. Furthermore, the identification of malicious traffic in network traces is not straightforward and often requires a significant amount of manual labelling work. Introducing malicious software into an enterprise network would be impossible for ethical and security reasons. For that reason, only about four structured datasets for network intrusion containing real-world traffic and attacks are available openly. The most recent and notable real-world datasets have been released from the Los Alamos National Laboratory (LANL) in 2015 and 2017 [19, 36], and the University of Granada (UGR) in 2016 [21]. Both datasets contain network flow traffic data from a large number of hosts collected over multiple months, giving an accurate representation of medium- to large-scale structures in benign traffic. However, the amount of attack data is small and insufficient for accurate detection rate estimation. Furthermore, packet-level data is not available for both datasets. Other real-world datasets, such as CAIDA 2016 [37] or MAWI 2000 [33], provide packet headers, but are unstructured and contain no labeled attack data at all.

To improve the lack of attack traffic in NIDS datasets, several artificially created datasets have been proposed. For this, a testbed of virtual machines is usually hosted in an enclosed environment to prevent any malicious code from spreading to other machines on other networks. To generate attack traffic, these machines are then subject to a selection of attack carried out by other machines in the environment. Benign traffic is generated using commercial traffic generators such as the *IXIA PerfectStorm tool*, or by scripting a selection of tasks for each machine. Synthetic datasets cover a smaller timeframe and contain traffic from a small number of hosts. Notable examples are the CIC-IDS 2017 dataset from the Canadian Institute for Cybersecurity [29], and the UNSW-NB 2015 dataset from the University of New South Wales [25]. Both datasets contain traffic from a variety of attacks, and are available as packet headers or as network flows with additional features crafted for machine-learning. While the benign traffic for the CIC-IDS 2017 data was generated using scripted tasks from a number of host profiles, the benign data for the UNSW-NB 2015 data is a mixture of captured real traffic from another subnet and traffic generated using a commercial traffic replicator.

We omitted the synthetic KDD-Cup 1999 and the DARPA 1998 datasets along with their derivates from the discussion as they are well-known to be outdated and contain unrealistic benign traffic, artificially high benign/attack data ratios, and artifacts stemming from communication simulations [23, 35], problems which have been addressed by most modern datasets.

Container networks have recently been adopted to conduct traffic generation experiments, such by Fujdiak et al. [9] who use containerized web servers to collect DoS-traffic. Furthermore, significant effort has been put into the creation of large-scale virtualization frameworks to provide automatized network testbeds [3, 6].

2.3 Problems in modern datasets

The difficulty of obtaining malicious traffic in real-world captures means that the performance of new network intrusion detection algorithms are almost exclusively evaluated on synthetic datasets. Potential disadvantages of synthetic compared to real-world datasets have been discussed by several authors [32, 34]. However, none address problems in the particular design of such synthetic testbeds

that are holding machine-learning based methods back in performance and from getting more widespread application. Here, we focused on this aspect and four design problems common among modern synthetic datasets.

Lack of variation. To generate benign traffic, a selection of activities is scripted and executed on virtual machines. Activities are selected to cover the most prominent protocols, but seldom to cover the range of subactivites that each protocol offers. Instead, the manner in which each protocol is used is highly-restricted, and there are doubts about whether this traffic is representative of its real-world equivalent usage [32]. An illustrative example of the restricted protocol activity in synthetic datasets can be seen in the CIC-IDS 2017 dataset. Here, the vast majority of successful FTP transfers consist of a client downloading a single text file containing the Wikipedia page for 'Encryption' several hundred times in a day. In reality, FTP is used for a large number of tasks, which can occur in random order with varying input sizes and parameters.

In addition to that, implemented test bed environments are usually separated from external influence or even virtualized, which isolates them from fluctuations and faults introduced by the complexity of modern networking. These include packet delays through network congestions, unexpected connection drops or resets, and out-of-order arrivals, all of which lead to variations in the response behavior of particular services.

This general lack of variation in individual protocols leads to observed homogeneity both on a packet exchange level and on a network flow level, and thus to clearer structures in the data. Identifying separations of malicious and benign activity or between different services consequently becomes easier, which leads to overoptimistic results in the evaluation of machine-learning based methods. It is therefore clear that traffic variation is a crucial aspect of a comprehensive intrusion detection dataset. parameter

Lack of ground truth. To evaluate machine-learning-based methods that distinguish between different types of network traffic data, we need to verify that separating structures in the model correspond to distinct computational actions, using traffic labels. Most obvious is the labelling of benign and malicious traffic. More granular labels are desirable to distinguish between several different types of network traffic. An example for this is the design of 'stepping stone' detection methods, where researchers try to detect connections relayed over a jump-host. Similarity or correlation metrics that measure the closeness of two connections are a popular tool. To understand such a measure, ground truth about how computationally similar two connections are and what type of behavior they represent is necessary. Other areas that look at small-scale traffic structures and would benefit from detailed traffic labels include protocol verification, traffic classification, traffic disaggregation, or exploit discovery.

Ground truth labels for network traffic are hard to obtain. The network traffic produced by a typical PC will invariably contain traffic originating from background processes, such as software updates, authentication traffic, network discovery services, advertising features, as well as many other sources. To separate traffic from different origins retrospectively is often hard, if not impossible. Source attribution through port numbers is unreliable because port numbers can be dynamically allocated and are not restricted to particular processes, and processes can open connections on multiple ports at the same time. All of these reasons mean that the identification of different computational operations from captured traffic is often infeasible. Therefore, no public NID dataset currently considers the inclusion of ground truth traffic labels.

Static design. A released dataset can only contain data that is representative a system at the time of creation. In contrast to other many other data sources for machine-learning, network traffic, both benign and malicious, is constantly changing as computational protocols and systems evolve. All available NIDS datasets today have been created in a static manner , so that a fixed test bed of host machines is created designed to contain specific vulnerabilities to the selected attacks. This makes it very hard to change the test bed and thus adjust the dataset to updated traffic structures. Allix et al. [1] claim that it is impossible to release a NID dataset that is truly representative of the real-world attacks due to the inherent secrecy of the intrusion ecosystem and the rate at which it develops.

Limited size. Today's machine-learning revolution was supercharged by the exponential growth of available data. Larger amounts of data mean that a given model can identify more complex structures that remain invariant in noisy environments and thus generalize better. Although the amount of globally transmitted network traffic is growing every year, the size of available NIDS datasets is limited by small host numbers, typically 5-10, and short capture periods, at maximum a 5-6 weeks, inherent to test bed captures. This means that traffic models can experience difficulties to generalize over specific traffic types which represent a smaller fraction of the total dataset. In an ideal setting, researchers would have the ability to generate arbitrary amounts of specific traffic types.

2.4 Containerization with Docker

Virtual machines (VMs) share the same hardware infrastructure as the host machine. VMs necessitate the use of hypervisors, software responsible for sharing the host OS's hardware resources, such as memory, storage and networking capabilities. OS-level virtualization, also known as *containerization*, is a virtualization paradigm that has become popular in recent years due to its lightweight nature and speed of deployment. In contrast with standard VMs, containers forego a hypervisor and the shared resources are instead kernel artifacts, which can be shared simultaneously across several containers. Although this prevents the host environment from running different operating systems, containerization incurs minimal CPU, memory, and networking overhead whilst maintaining a great deal of isolation [20].

The main advantage of using containers for traffic generation is the isolation of individual applications. This enables us to gather ground truth about the traffic origin, and enables us to easily extend, modify, and scale our traffic generation framework, which would not be possible when relying on VMs.

Docker container. Docker is a software platform that allows for the creation, maintenance and deployment of containers. In Docker's terminology, a container is a single, running instance of a Docker *image*. Docker images are defined via a text file known as the `Dockerfile`, which consists a series of commands that modify an underlying *base image*, usually a containerized OS. Example

```
FROM ubuntu
MAINTAINER XYZ (email@domain.com)
RUN apt-get update
RUN apt-get install -y nginx
ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]
EXPOSE 80
```

Figure 1: Example of `Dockerfile` creating a nginx-container.

commands include installing libraries and copying files. Figure 1 displays a simple example of `Dockerfile`.

After each command is executed, the intermediate, read-only image is saved as a *layer*. These layers can be shared between containers. When a Docker image is run as a container, a final read-write layer is added and when the container is later stopped, this layer is discarded, preserving the integrity of the underlying layers. This allows Docker containers to be run repeatedly whilst always starting from an identical state.

Individual Docker containers are intended to be highly specialized in their purpose with each container running only a specific piece of software or application. Commonly used base images — such as Alpine Linux — have minimal background processes running during a container's lifetime. This means that the network traces of a Docker container can be associated with a specific application. The one-to-one correlation between containers and network traces allows us to produce labeled datasets with fully granular ground truths.

The Docker software platform includes a cloud-based repository called the Docker Hub [15] which allows users to download and build open source images on their local computers. At the time of writing, nearly 2.5 million images are available from the Docker Hub. Some common software — such as popular webservers and databases — have officially maintained images. We use these as far as possible to simplify the production of our scenarios, and keep them close to software configuration used in practice.

Docker Networking. Docker allows the creation of virtualized networks with one or more subnetworks, to which containers can connect via a virtualized network bridge. Containers attached to the bridge network are assigned an IP address and are able to communicate with other containers on their subnetwork. Containers can furthermore be connected to a host network, which allows communication with external networks using NAT via the host interface.

To host containers in an isolated network, we can create our own user-defined bridge networks, which provides greater isolation between containers [14]. Furthermore, this allows us to fix the subnet and gateway for our networks as well as the IP addresses of our containers, which simplifies scripting scenarios. Docker allows containers to share the same network interface. This enables us to assign the same IP address to multiple containers.

Netem. The Docker engine also provides network access to Linux traffic control facilities such as *NetEm*, on which we will rely in this project. NetEm is a Linux toolkit for testing protocols by emulating properties of wide area networks [11]. It allows the user emulates variable delay, loss, duplication and re-ordering of packets on particular network interfaces.

```
version: '3'
services:
 webserver:
 image: nginx:alpine
 ports:
 - "80:80"
 networks:
 - app-network
 db:
 image: mysql:5.7.22
 ports:
 - "3306:3306"
 environment:
 MYSQL_DATABASE: laravel
 MYSQL_ROOT_PASSWORD: your_mysql_root_password
 networks:
 - app-network
networks:
 app-network:
 driver: bridge
```

Figure 2: Example of `Docker-compose` file launching a nginx- and a mysql-container in an isolated network.

Docker Compose. Applications built using the Docker framework often need more than one container to operate, for example an Apache server and a MySQL server running in separate containers. We must build and deploy several interconnected containers simultaneously. Docker provides this functionality via `Docker compose`, a tool that allows users to define the services of multiple containers as well as the properties of virtual networks in a YAML file. By default, this file is named *docker-compose.yml*. This allows for numerous containers to be started, stopped and rebuilt with a single command in a consistent manner. This is particularly significant for our purposes; with `Docker compose`, we can launch several containers in a specific order, with a specific network configuration, whilst running specific commands within each container on start up. This ensures that our interactions are deterministic, barring any added randomization. Figure 2 displays an example of a simple `Docker compose` file.

2.5 Dataset Requirements

The primary task of this project is to provide a suite of Docker container compositions that is capable of generating traffic datasets suitable for machine-learning-based intrusion detection systems. This container suite is designed to address the criticism of current NIDS datasets discussed in Section 2.3. For this, we created a set of requirements that a modern intrusion detection dataset has to fulfil to address the problems discussed in Section 2.3:

Variation. To ensure that we produce representative data for modelling, we want the traffic generated by our container suite to cover a sufficient number of protocols that are commonly found in real-world traffic and existing datasets. For malicious traffic, we want to ensure that the attacks are modern and varied, both in purpose and in network footprint. For each protocol, we want to establish several capture scenarios to encompass the breadth of that protocol's possible network traces. Communication between containers should be subject to the same disturbances and delays as in a real-world setting.

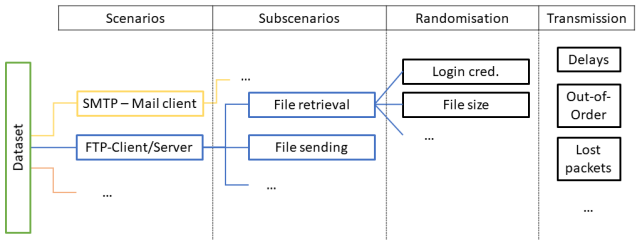

Figure 3: Visualization of the different levels at which traffic variation is introduced in DetGen.

Ground truth. Since ground truth is a main focus of this work, we want a capture scenarios to be consistent and reproducible in the traffic they generate. This way, we can be certain that a particular traffic trace corresponds to the capture scenario it was generated by, and can thus relate individual traffic events to computational operations. We discuss what it means for a scenario to be reproducible in detail in Section 4.1.

Modularity. Traffic capture scenarios should be implemented in a modular way allow for a straightforward addition or modification of traffic capture modules without disrupting the rest of the container suite. This reduces the effort to adjust a dataset to changing traffic patterns and allows the addition of modern attacks traffic.

Scalability. Each capture scenario should be running in a scalable manner to allow generation of large data quantities.

3 DESIGN

To cover a range of activities, the containers in our framework are arranged in different configurations corresponding to particular *capture scenarios*. Running a given capture scenario triggers the launch of several Docker containers, each with a scripted task specific to that capture scenario. A simple exemplary capture scenario may consist of a containerized client pinging a containerized server. We ensure that each Docker container involved in producing or receiving traffic will be partnered with a `tcpdump` container, allowing us to collect the resulting network traffic from each container's perspective automatically.

We outline different stages within the creation of a dataset at which traffic variation is introduced. Figure 3 visualizes this process.

3.1 Scenarios

We define a *scenario* as a series of Docker containers interacting with one another whereby all resulting network traffic is captured from each container's perspective. This constructs network datasets with total interaction capture, as described by Shiravi et al. [31]. Each scenario produces traffic from either a protocol, application or a series thereof. Both benign and malicious activities are implemented as scenarios. Examples may include an FTP interaction, a music streaming application and client, an online login form paired with an SQL database, or a C&C server communicating with an open backdoor. A full list of currently implemented scenarios can be found in Section 3.9.

Each scenario is designed to be easily started via a single script that allows the user to set the length of the capture time, and the specification of particular subscenarios, discussed below. Scenarios can be repeated indefinitely without further instructions and be run in parallel, therefore allowing the generation of large amounts of data.

Our framework is modular, so that individual scenarios are configured, stored, and launched independently. Adding or reconfiguring a scenario has no effect on the remaining framework.

3.2 Subscenarios

In contrast to scenarios, *subscenarios* provide a finer grain of control over the traffic to be generated, allowing the user to specify the manner in which a scenario should develop. The aim of having multiple subscenarios for each scenario is to explore the full breadth of a protocol or application's possible traffic behavior. For instance, the SSH protocol can be used to access the servers console, to retrieve or send files, or for port forwarding, all of which may or may not be successful. It is therefore appropriate to script multiple subscenarios that cover this range of tasks.

The same applies to malicious activity. For instance, it would be naive for an SSH password bruteforcing scenario to always successfully guess a user's password. Instead, we include a second subscenario in which the password bruteforcer fails.

Subscenarios are specific to particular scenarios and can be specified when launching that scenario.

3.3 Randomization within Subscenarios

Scripting activities that are otherwise conducted by human operators often leads to a loss of random variation that is normally inherent to the activity. As mentioned in Section 2.3, the majority of successful FTP transfers in the CIC-IDS 2017 data consist of a client downloading a single text file. In reality, file sizes, log-in credentials, and many other variables included in an activity are more or less drawn randomly, which naturally influences traffic quantities such as packet sizes or numbers.

To account for these fluctuations, we identify variable input parameters within scenarios and their subscenarios and systematically draw them randomly from a suitable distribution. Passwords and usernames, for instance, are generated as a random sequence of letters with a length drawn from a Cauchy distribution, before they are passed to the corresponding container. Files to be transmitted are selected at random from a larger set of files, covering different sizes and file names.

3.4 Network transmission

Docker communication takes place over virtual bridge networks, so the throughput is far higher and more reliable than in real-world networks, with the Docker virtual network achieving a bandwidth of over 90 Gbits/s when measured using iPerf [10]. This level of speed and consistency is worrying for our purposes as packet timings will be largely identical on repeated runs of a scenario and any collected data could be overly homogeneous.

To retard the quality of the Docker network to realistic levels, we rely on emulation tools. As discussed in section 2.4, Netem is a Linux command line tool that allows users to artificially simulate

network conditions such as high latency, low bandwidth or packet corruption in a flexible manner.

Although it is relatively straightforward to apply Netem commands to a Docker Bridge network, we decided not to invoke Netem in this manner as this would cause all network settings of all containers to be identical, such as all containers in a scenario having a latency of 50ms. Instead, we developed a wrapping script that applies Netem commands to the network interface of a given container, providing us with the flexibility to set each container's network settings uniquely. This script randomizes the values of each parameter, such as packet drop rate, bandwidth limit, latency, ensuring that every run of a scenario has some degree of network randomization if desired.

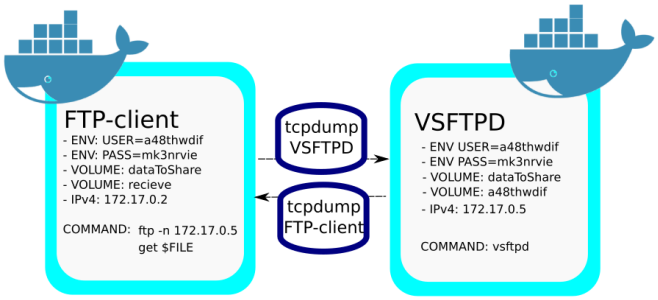

Figure 4: Diagram of FTP scenario

3.5 Capture

To capture traffic, we use containers running `tcpdump`, a widespread and free packet analyzer software that can capture packets arriving at or leaving from a network interface [17]. We attach `tcpdump` containers on every interface in the virtualized docker network and write packets into separate capture files. This allows us to capture traffic from the perspective of every container in a scenario, giving a complete view. In Section 3.8, we discuss how the collected capture files are coalesced into one dataset.

3.6 Implementation Process

The implementation process for each scenario follows broadly the same outline:

(1) Select containers which provide the required services and identify the *primary* container/s for a given scenario which is/are dictating the container interaction. Then create and build a Dockerfile containing all necessary dependencies.
(2) Identify different ways to use the service of the given scenario and define them into a set of subscenarios.
(3) Design and implement the behavior for secondary containers to provide the required service to the primary container(s).
(4) For each subscenario, identify variable input values and their appropriate range; then systematically implement their generation from appropriate distributions covering this range.
(5) Add `tcpdump` containers to every network interface.
(6) Create a `Docker compose` file that launches all containers simultaneously.
(7) Finally, write a script that, upon running, calls this `Docker compose` file, applies a network emulation script to each container network interface, and allows the user to specify how long and how many times a scenario should be run.

Following the Docker guidelines [12], each container in our framework consists of a single service with a specialized purpose, with as few additional dependencies as possible. Moreover, we ensure that there are minimal inter-dependencies between the containers of a scenario. This allows us to easily modify and update containers as new versions of the underlying software are released.

3.7 Simple Example Scenario - FTP server

We review the design of a prototypical capture scenario, namely, an FTP server and client interaction. The interaction is initiated by a single script, which allows the user to specify the length of the interaction, the number of times the interaction takes place as well as the specific subscenario. The script generates a random ftp username and password, creating the necessary *User* directory on the host machine before calling the Docker-compose file which creates a bridge network. Subsequently, the necessary containers are then started which, in this case, consist of a VSFTPD server, a client with ftp instaled and two containers running tcpdump to capture all of the traffic emitted and received by the client and server respectively into separate `.pcap`-files. These `.pcap`-files are shared with the host machine via a shared volume. The host machine also shares:

- A *dataToShare* volume containing files that can be downloaded by the client.
- The *User* directory with the server, which contains the same files as the *dataToShare* folder.
- An empty *receive* folder with the client into which the files will be downloaded.
- The random username and password is shared with the client container so it can authenticate itself to the server.

Up to this point, no network traffic has been generated and the containers are now ready to begin communicating with one another. For this particular interaction between an FTP server and client, we want to ensure that it is possible to capture the many ways in which an FTP session may develop. For instance, the client may seek to download files via the `get` command or the `put` command, alongside many other possibilities. We define 13 possible capture subscenarios intended to encapsulate a wide range of potential FTP sessions. These include downloading a single file using `get` or `put`, downloading every file using `mget` or `mput`, deleting every file from the server and requesting files from the server without the necessary authentication.

After the scenario ends, both the *User* directory and any downloaded files are removed from the host machine. The containers are then stopped and the bridge network is torn down. All necessary containers, volumes and scripts are in the same position prior to initiating the scenario — barring any generated `.pcap`-files — allowing for the scenarios to be started repeatedly with minimal human interaction. The `.pcap`-files are tagged with information about the time of creation, executed scenario and subscenario, and the container generating the traffic.

3.8 Dataset creation

Our framework generates network datasets consisting of a single interaction, but it is possible to coalesce these datasets to create larger datasets with a wide variety of traffic, albeit with some caveats. Due to the networking constraints of the Docker virtual network, such as limitations regarding clashing ports, running many of our Docker scenarios simultaneously over a large period of time is unfeasible. Thus, to ensure that the generated traffic is suitably heterogeneous, numerous datasets must be generated before being coalesced into a main dataset. If done naively, this presents a problem. As discussed by Shiravi et al. [31], merging distinct network data in an overlapping manner can introduce inconsistencies. For instance, if one wanted to create a dataset containing both normal webserver traffic and traffic originating from a Denial of Service attack, it would not work to generate these two datasets separately before merging them together. If these two events really did occur simultaneously, the high network throughput of the latter would likely effect the packet timings of the former.

To avoid such inconsistencies, we create larger datasets by collecting data in consecutive chunks of fixed time. Within each chunk, several scenarios are run simultaneously. All `.pcap`-files collected during a given chunk can be merged together. It is then simple to stitch together all of these chunks into a single `.pcap`-file using a combination of `Mergecap` [28] and `Editcap` [30]. This allows us to shift the timings of each `.pcap`-file by a fixed amount such that all of our chunks occur in succession whilst maintaining the internal consistency of each chunk.

3.9 Scenarios

Our framework contains 29 scenarios, each simulating a different benign or malicious interaction. The protocols underlying benign scenarios were chosen based on their prevalence in existing network traffic datasets.These datasets consist of common internet protocols such as HTTP, SSL, DNS, and SSH. According to our evaluation, our scenarios can generate datasets containing the protocols that make up at least 87.8% (MAWI), 98.3% (CIC-IDS 2017), 65.6% (UNSW NB15), and 94.5% (ISCX Botnet) of network flows in the respective dataset. Our evaluation shows that some protocols that make up a substantial amount of real-world traffic are glaringly omitted by current synthetic datasets, such as BitTorrent or video streaming protocols, which we decided to include.

In total, we produced 17 benign scenarios, each related to a specific protocol or application. Further scenarios can be added in the future, and we do not claim that the current list exhaustive. Most of these benign scenarios also contain many subscenarios where applicable.

The remaining 12 scenarios generate traffic caused by malicious behavior. These scenarios cover a wide variety of major attack classes including DoS, Botnet, Bruteforcing, Data Exfiltration, Web Attacks, Remote Code Execution, Stepping Stones, and Cryptojacking. Scenarios such as stepping stone behavior or Cryptojacking previously had no available datasets for study despite need from academic and industrial researchers.

We provide a complete list of implemented scenarios in Table 1.

Name	Description	#Ssc.
Ping	Client pinging DNS server	1
Nginx	Client accessing Nginx server	2
Apache	Client accessing Apache server	2
SSH	Client communicating with SSHD server	5
VSFTPD	Client communicating with VSFTPD server	12
Scrapy	Client scraping website	1
Wordpress	Client accessing Wordpress site	1
Syncthing	Clients synchronize files via Syncthing	1
mailx	Mailx instance sending emails over SMTP	2
IRC	Clients communicate via IRCd	2
BitTorrent	Download and seed torrents	3
SQL	Apache with MySQL	2
NTP	NTP client	2
Mopidy	Music Streaming	5
RTMP	Video Streaming Server	1
WAN Wget	Download websites	5
SSH B.force	Bruteforcing a password over SSH	3
URL Fuzz	Bruteforcing URL	1
Basic B.force	Bruteforcing Basic Authentication	2
Goldeneye	DoS attack on Web Server	1
Slowhttptest	DoS attack on Web Server	4
Mirai	Mirai botnet DDoS	3
Heartbleed	Heartbleed exploit	1
Ares	Backdoored Server	3
Cryptojacking	Cryptomining malware	1
XXE	External XML Entity	3
SQLi	SQL injection attack	2
Stepstone	Relayed traffic using SSH-tunnels	2

Table 1: Currently implemented traffic scenarios along with the number of implemented subscenarios

4 VALIDATION EXPERIMENTS

A framework that generates network traffic does not necessarily provide realistic and useful data. To evaluate the utility of our Docker framework, we construct a series of experiments. We have two goals in mind. First, we want to demonstrate that the traffic generated is sufficiently representative of real-world traffic. Second, we want to demonstrate that having a framework to continually generate data compared to static datasets benefits evaluating the efficacy of intrusion detection systems.

The first experiment provides a general verification of the reproducability of our framework, which is required for guarantee the ground truth of the produced data. The second experiment demonstrates that the WAN-characteristics we emulate for our data make it quasi non-distinguishable from real WAN traffic. Our third experiment then demonstrates the advantage of unlimited data generation capabilities for training ML-based traffic classification.

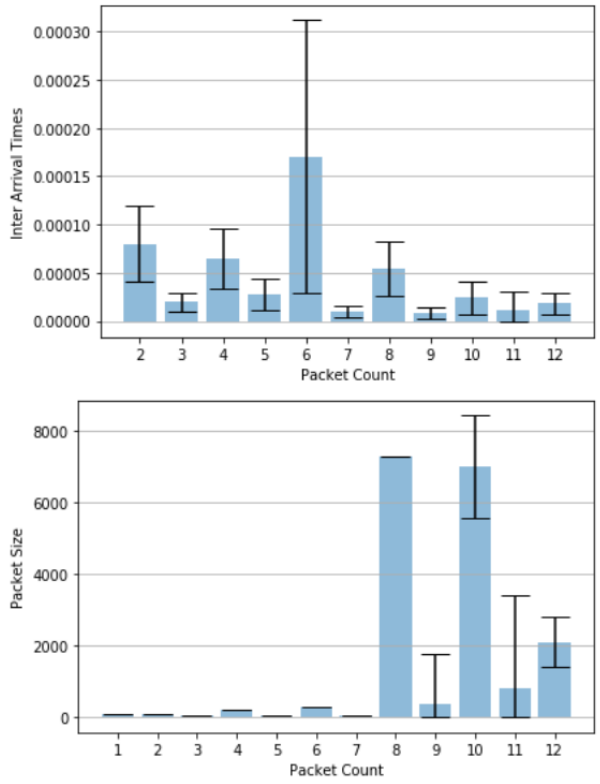

Figure 5: Means of IATs & packet sizes along with standard deviation bars for the first twelve packets in the Apache scenario.

4.1 Reproducible scenarios

To provide ground truth, we have to guarantee that our implemented scenarios and subscenarios are consistent and reproducible upon repeated execution. This applies both to consistency for external influences on the host, such as increased computational load, as well as internal consistency of the implemented script execution.

It is impossible to guarantee that each scenario will produce a truly 'deterministic', or repeatable, output due to differences in network conditions, computational times, or input. Instead, we aim for our data to be *reproducible up to networking and computational differences*. This means that when running a scenario multiple times, we expect the quantities of most packets to be largely identical. We do expect some packets to exhibit greater variation due to non-determinism in the underlying protocols, Fig. 5 outlines this behavior in terms of interarrival times and packet sizes.

To measure how consistent our scenarios are, we generate 500 `.pcap` files for three different implemented scenarios, namely the Apache, the VSFTPD, and the SSH scenario. These were generated consecutively under different host CPU load. We did not apply any delays or other NetEm traffic controls.

We assess the consistency of a scenario across different `.pcap` files by comparing all generated `.pcap` files pairwize. We measure this by the similarity of the connections captured.

To test the similarity of two connections, we extract the sample distributions of the packet interarrival times and packet sizes overall, upstream and downstream. We define two connections as similar if the two distributions for each of these quantities pass an equality test. We use the two-sample Kolmogorov-Smirnov (K-S) test, a non-parametric statistical test for the equality of two continuous one-dimensional distributions [22], with a p-value of 0.01.

As all tested files passed this similarity test, we conclude that these scenarios yield consistent and reproducible results. As other scenarios follow the same setup and launch commands, we expect the results to stay the same as long as the involved containers are consistent in their behavior.

4.2 Explorating Artificial Delays

Most traffic our framework generates is transported over Docker's virtual network and therefore does not succumb to problems associated with normal network congestion, such as packet loss, corruption and packets arriving out of order. A realistic dataset should include these phenomena, which is why we developed wrapping scripts that allow us to artificially add delays as well as packet loss and corruption, using NetEm. Choosing the parameters is not straightforward; it is not clear how close to real-world traffic such network emulation techniques are. This is especially true for packet delays, which are described by continuous distributions and often have temporal correlation.

Furthermore, the high effective bandwidth of the Docker virtual network resulted in traffic with extremely short inter-arrival times (IATs, defined as the time between two packet arrivals). Therefore, we devote considerable time to demonstrating that it is possible for traffic generated by our Docker framework to conform to real-world IAT distributions when altered using NetEm.

Datasets. We create two classes of datasets, one which is representative of 'real-world' traffic, and one which has been generated from our Docker framework. For simplicity, we only consider datasets consisting of FTP traffic.

For the real-world dataset, we set up a containerized VSFTPD server running on a *Google Compute* virtual machine located in the Eastern United States, and a containerized FTP client on our local host. We then ran a series of our scripted interactions between the two machines, generating 834 megabytes of data in 250964 packets. These interactions consisted of several FTP commands with various network footprints. We collect all data transmitted on both the server and the client. We call this data the *Non-Local* dataset.

We then repeat this process using the same container setup, but across the Docker virtual network on a local machine. We repeat this process several times, generating several *Local* datasets under a variety of emulated network conditions, discussed in Section 4.2. Our *Local* datasets vary slightly in size, but are all roughly 800 megabytes with 245000 packets.

Methodology. NetEm allows us to introduce packet delays according to a variety of distributions, namely uniform, normal, Pareto

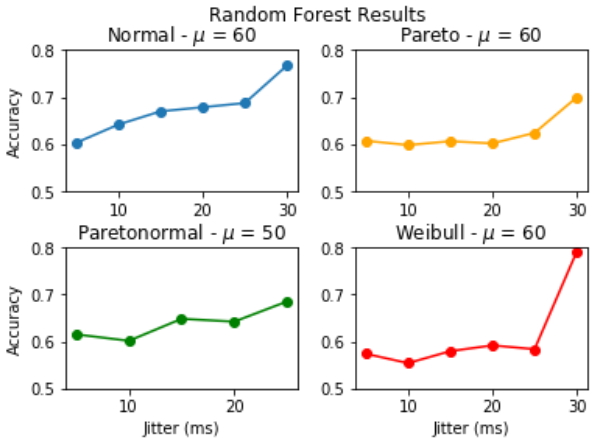

Figure 6: Results of Random Forest Classifier for a given distribution at the best performing delay mean μ. Note that a score of .5 indicates total indistinguishability.

and Paretonormal[1]. Furthermore, NetEm adds delays according to modifiable distribution tables, and so it is trivial for us to add a Weibull distribution, which along with Pareto distributions have been shown to closely model packet IATs [2, 27]. In total, we test the efficacy of four distributions to model inter-arrival times — normal, Pareto, Paretonormal and Weibull.

We generate several *Local* datasets by delaying traffic according these distributions, performing an exhaustive grid search over their means and standard deviations. Initial experiments revealed that introducing delays with a mean in the range of 40 ms to 70 ms produced the best results. Setting the jitter of the distribution too high resulted in the repeated arrival of packets out of order, therefore we further limit the grid search to jitter values in 5ms intervals up to half of the value of the mean. In total, we generate 88 *Local* datasets.

Our goal is to discover the *Local* dataset whose packet timings most closely resemble those of our *Non-Local* dataset. To do this, we extract the IATs and packet sizes from our datasets on a packet-by-packet basis and store these results in arrays. We measure the similarity between two of these arrays by training a Random Forest classifier to distinguish between them. We say that if the Random Forest correctly classifies each packet with a success rate of only 50% then it is no better than randomly guessing and, as such, the inter-arrival times of these two arrays are indistinguishable from one another for the Random Forest.

To perform this measurement, we concatenate one *Local* dataset array with our *Non-Local* dataset array, label the entries and then shuffle the rows. We proportion this data into a training set and a testing set using an 80-20 split. We then feed this training data into a Random Forest with 1000 trees and fixed seed, and then record the accuracy of this Random Forest on the test set. We repeat this process for every single *Local* dataset.

[1]This Paretonormal distribution is defined by the random variable $Z = 0.25 * X + 0.75 * Y$, where X is a random variable drawn from a normal distribution and Y is a random variable drawn from a Pareto distribution.

Results. Table 2 summarizes the values of the mean and jitter for a given distribution that produced the worst results from the random forest classifier.

Distribution	Mean	Jitter	RF Accuracy
No Delays (Baseline)	0	0ms	0.8176
Constant Delay	40ms	0ms	0.6730
Normal	60ms	5ms	0.6028
Pareto	60ms	10ms	0.5979
Paretonormal	50ms	10ms	0.6015
Weibull	60ms	10ms	0.5540

Table 2: Worst Random Forest accuracy rates for a given distribution

To establish a baseline, we compare the traffic generated from our Docker scenario to that of the *Google Compute* data with no added delays. In this case, the Random Forest was able to distinguish between the two datasets, achieving an accuracy of over 90%. The classification accuracy is worsened considerably by introducing network delays, with the best results being achieved using a Weibull distribution with a mean of 60 ms and a jitter of 10 ms, leading to an accuracy of just 55%. Results for Pareto and Weibull distributions seem to yield consistent results for differing jitter values. Although not completely indistinguishable, this proves that using NetEm we can emulate WAN properties very closely.

4.3 Advantages of Dynamic Dataset Generation

Having examined whether our Docker framework is capable of emulating real-world IATs, we explore their utility in traffic classification to demonstrate the advantages that our framework provides compared to static, unlabeled datasets.

Machine-learning techniques are a popular tool for traffic classification, with many successful published classifiers. Furthermore, inter-packet arrival times have been shown to be a discriminative feature [26, 38]. However, these methods considered datasets consisting of completed traffic flows, limiting their use in, say, a stateful packet inspector. On-the-fly classifiers are also successful. Jaber et al. [16] showed that a K-means classifier can classify flows in real-time solely based on IATs with precision exceeding 90% for most protocols within 18 packets. Similarly, Bernaille et al. [4] demonstrated that a K-means classifier can precisely classify traffic within five packets using only packet size as a feature.

However, Jaber et al. [16] only evaluated their traffic classifier with training and testing data drawn from the same dataset containing traces of a single network; there is no measure of how this model may generalize to other networks with differing conditions. Furthermore, they were limited to using unsupervised machine learning algorithms to classify their traffic as their datasets had no ground truth.

We attempt to replicate these results within our Docker framework with some adjustments. As we can generate a fully accurate ground truth, we attempt to segregate application flows based on their packet IATs using supervised learning techniques. Moreover, we then measure this model's ability to generalize by expanding our dataset to include traffic from networks with differing bandwidth and latency.

Data & Preprocessing. Our goal is to measure a classifier's ability to generalize across datasets. Therefore we construct two datasets using our Docker framework, both containing the same number of network traces from the same containers.

For our first dataset, we generate `.pcap`-files, each containing traffic from one of 16 different classes: HTTP (Client & Server), HTTPS (Client & Server), RTMP (Client, Server & Viewer), SSH (Client & Server), FTP (Client & Server), IRC (Client & Server), SMTP, SQLi and DoS traffic. To prevent class imbalance, we generate 200 `.pcap`-files for each of the 16 classes, resulting in 3200 total files. To more accurately emulate potential network conditions, we use our NetEm scripts to apply a unique delay to every container involved in a scenario. These delays follow a Pareto distribution with random mean between 0 and 100 milliseconds and random jitter between 0 and 10 milliseconds. We then preprocess this data by removing all but the first 12 packets of each `.pcap`-file. We extract the 11 inter-arrival times separating the 12 packets, which act as our feature vectors. We collect these feature vectors for each class along with a class label, and store collected feature vectors from all 3200 `.pcap`-files in a 12 x 3200 array. We call this our *Primary* dataset.

We then repeat this process to generate a second dataset, changing the properties of our emulated network. Again, we delay all traffic using a Pareto distribution, however, this time we select a random mean in the range of 100 to 500 milliseconds and random jitter between 0 and 50 milliseconds. The subsequent preprocessing of our data remains unchanged. We call this our *Secondary* dataset.

Methodology. First, we attempt to reproduce the results presented by Jaber et al. [16] by training a Random Forest with 100 trees to classify application flows based on packet IATs. We do this by proportioning our Primary dataset into training and testing sets using an 80-20 split. We then train and test our Random Forest repeatedly, first considering the classification accuracy based on the IATs of only the first two packets, then the first three packets and so on, up to 12 packets. We record the resulting confusion matrix for each round and calculate the precision and recall rates of our classifier.

Having trained the classifier, we measure its ability to generalize by repeating the above experiment, but replacing the test set with the Secondary dataset.

Results. After each run of our Random Forest on our Primary dataset, we gather the True Positive (T_P), False Positive (F_P) and False Negative (F_N) rate for each class. We then calculate their precision, defined as $\frac{T_P}{T_P+F_P}$, and recall, defined as $\frac{T_P}{T_P+F_N}$, values. in Fig. 7, we see that our average precision and recall across the classes exceeds 0.9 after 10 IATs. Furthermore, after 12 packets our DoS and SQLi data is classified with precision and recall rates of 1.0 and 1.0 and 0.9462 and 0.9322 respectively.

These results does not hold when we test the classifyer on our Secondary dataset. As seen in Fig. 7, we see a substantial decrease in our average precision and recall rates, achieving a maximum of 0.5923 and 0.5676 respectively. Moreover, after four packets, increasing the number of IATs in our dataset provides little additional benefit. Although some services generalized well, such as IRC-client and IRC-server, others failed to be classified, with every

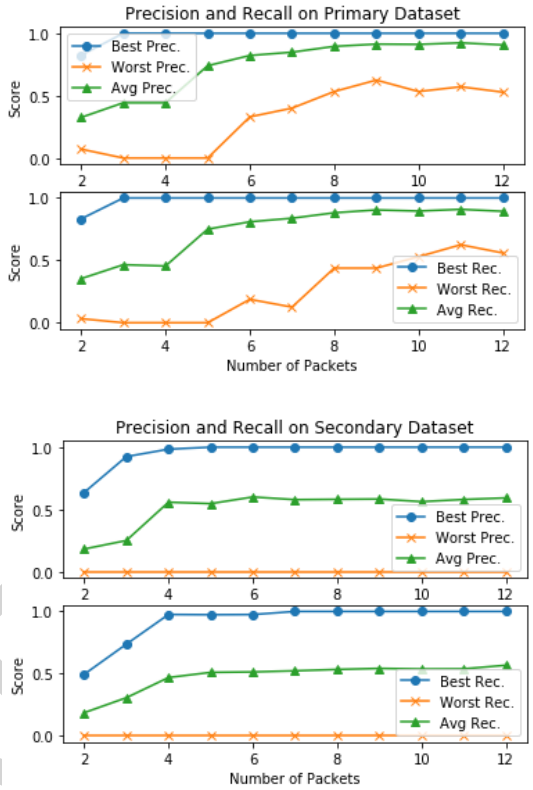

Figure 7: Results of Random Forest Classification on Primary dataset (Above) and Secondary dataset (Below)

single SMTP feature being classified as HTTP-client. We also see a substantial drop-off in the classification of malicious traffic, with the precision rates of DoS and SQLi data not exceeding 0.6.

These diverging results demonstrate the necessity of dynamic dataset generation for evaluation purposes. Researchers evaluating their methods only on a dataset with fixed properties such as the Primary dataset might receive overoptimistic results. The capability of generating two or more datasets with the same traffic classes, but otherwise differing properties, provides a more realistic evaluation.

5 CONCLUSIONS

In this paper, we outlined four requirements a modern dataset has to fulfil to strengthen the training of intrusion detection systems. We then proposed a Docker framework capable of generating network intrusion datasets that satisfy these conditions. The major design advantage of this framework are the isolation of traffic scenarios into separate container arrangements, which allows the extension of new scenarios and detailed implementation of subscenarios as well as the capture of ground truth of the computational origins of individual traffic events. Furthermore, containerization enables the generation of traffic data at scale due to containers being lightweight and easily clonable.

We verified the realism of the generated traffic and the corresponding ground truth information with two experiments, and

demonstrated the usefulness of the framework in another experiment. Presently, our framework consists of 29 scenarios capable of producing benign and malicious network traffic. Several of these scenarios, such as the *BitTorrent* or the *Stepping-Stone* scenario, provide novel traffic data of protocols or behaviours that has not been widely available to researchers previously.

5.1 Difficulties and limitations

Our framework is building network traffic datasets from a small-scale level up by coalescing traffic from different fine-grained scenarios together. While this provides great insight into small-scale traffic structures, our framework will not replicate realistic network-wide temporal structures, such as port usage distributions or long-term temporal activity. These quantities would have to be statistically estimated from other real-world traffic beforehand to allow our framework to emulate such behavior reliably. Other datasets such as UGR-16 use this approach to fuse real-world and synthetic traffic and are currently better suited to build models of large-scale traffic structures.

Working with Docker containers can sometimes complicate the implementation of individual scenarios compared to working with VMs. Although several applications are officially maintained Docker containers that are free from major errors, many do not. For instance, in the *BitTorrent* scenario, most common command line tools, such as `mktorrent`, `ctorrent` and `buildtorrent`, failed to actually produce functioning torrent files from within a container due to Docker's union filesystem. Furthermore, due to the unique way in which we are using these software packages, unusual configuration settings are sometimes needed.

Lastly, capturing `.pcap`-files from each container can quickly exceed available disc space when generating traffic at scale. Depending on specific research requirements, it is advizable to add filtering or feature extraction commands to the scenario execution scripts to enable traffic preprocessing in real-time.

5.2 Future work

Our traffic generation framework is designed to be expandable and there are many avenues for future work. The continual development of scenarios and subscenarios would improve the potential realism of datasets generated using the framework. The addition of more malicious scenarios would enable a more detailed model evaluation and improve detection rate estimation. Another future improvement for framework is to add scripts that emulate the usage activity of individual scenarios by a user or a network.

Although ground truth for particular traffic traces is provided by capturing `.pcap`-files for each container individually, we have not implemented a labelling mechanism yet for the dataset coalescence process. Though not technically difficult, some thought will have to be put how such labels would look like to satisfy different research demands. Furthermore, the Docker platform provides the functionality to collect system logs via the `syslog logging driver`. We plan on implementing their collection in the future, where they could act either as traffic labels providing more ground truth details, or act as a separate data source that complements the collected traffic.

We wish to publish this framework to a wider audience, allowing for further modification. This will be done using a GitHub repository, which contains both the implemented capture scenarios as well as the corresponding container images.

6 ACKNOWLEDGMENTS

We are grateful to British Telecommunications PLC who are supporting the PhD research of the first author in the UK EPSRC CASE scheme, giving invaluable guidance on the needs and possibilities of intelligent security tools and their evaluation. Part of this paper draws on the 2019 MSc dissertation of the second author. Nikola Pavlov helped with some of the earlier implementation. The third author was supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1 and the Office of Naval Research ONR NICOP award N62909-17-1-2065.

REFERENCES

[1] Kevin Allix, Tegawendé François D Assise Bissyande, Jacques Klein, and Yves Le Traon. 2014. *Machine Learning-Based Malware Detection for Android Applications: History Matters!* Technical Report. University of Luxembourg, SnT.

[2] Muhammad Asad Arfeen, Krzysztof Pawlikowski, D McNickle, and Andreas Willig. 2013. The role of the weibull distribution in internet traffic modeling. In *Proceedings of the 2013 25th International Teletraffic Congress (ITC)*. IEEE, 1–8.

[3] Shreyas Badiger, Shrey Baheti, and Yogesh Simmhan. 2018. VIoLET: A Large-scale Virtual Environment for Internet of Things. In *European Conference on Parallel Processing*. Springer, 309–324.

[4] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. 2006. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review* 36, 2 (2006), 23–26.

[5] Nevil Brownlee, Cyndi Mills, and Greg Ruth. 1999. *Traffic flow measurement: Architecture.* Technical Report.

[6] Jonathan Crussell, Jeremy Erickson, David Fritz, and John Floren. 2015. *minimega v. 3.0.* Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[7] Jonathan Crussell, Thomas M Kroeger, Aaron Brown, and Cynthia Phillips. 2019. Virtually the Same: Comparing Physical and Virtual Testbeds. In *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 847–853.

[8] Jonathan Crussell, Thomas M Kroeger, David Kavaler, Aaron Brown, and Cynthia Phillips. 2019. Lessons learned from 10k experiments to compare virtual and physical testbeds. In *12th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET} 19)*.

[9] Radek Fujdiak, Vaclav Uher, Petr Mlynek, Petr Blazek, Jan Slacik, Vladimir Sedlacek, Jiri Misurec, Marina Volkova, and Petr Chmelar. 2018. IP Traffic Generator Using Container Virtualization Technology. In *2018 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. IEEE, 1–6.

[10] Mark Gates and Alex Warshavsky. [n. d.]. Iperf Man Page. https://linux.die.net/man/1/iperf. ([n. d.]). Accessed: 2019-08-11.

[11] Stephen Hemminger et al. 2005. Network emulation with NetEm. In *Linux conf au*. 18–23.

[12] Docker Inc. [n. d.]. Best Practises for Writing Docker] Images. https://docs.docker.com/develop/develop-images/dockerfile_best-practices/. ([n. d.]). Accessed: 2019-08-11.

[13] Docker Inc. [n. d.]. Docker. https://www.docker.com/. ([n. d.]). Accessed: 2019-08-11.

[14] Docker Inc. [n. d.]. Docker Documentation. https://docs.docker.com/. ([n. d.]). Accessed: 2019-08-11.

[15] Docker Inc. [n. d.]. Docker Hub. https://hub.docker.com/. ([n. d.]). Accessed: 2019-08-11.

[16] Mohamad Jaber, Roberto G Cascella, and Chadi Barakat. 2011. Can we trust the inter-packet time for traffic classification?. In *2011 IEEE International Conference on Communications (ICC)*. IEEE, 1–5.

[17] Van Jacobson, Craig Leres, and S McCanne. 1989. The tcpdump manual page. *Lawrence Berkeley Laboratory, Berkeley, CA* 143 (1989).

[18] H Günes Kayacik, A Nur Zincir-Heywood, and Malcolm I Heywood. 2005. Selecting features for intrusion detection: A feature relevance analysis on KDD 99 intrusion detection datasets. In *Proceedings of the third annual conference on privacy, security and trust*, Vol. 94. 1723–1722.

[19] Alexander D. Kent. 2015. Cybersecurity Data Sources for Dynamic Network Research. In *Dynamic Networks in Cybersecurity*. Imperial College Press.

[20] Kirill Kolyshkin. 2006. Virtualization in linux. *White paper, OpenVZ* 3 (2006), 39.
[21] Gabriel Maciá-Fernández, José Camacho, Roberto Magán-Carrión, Pedro García-Teodoro, and Roberto Therón. 2018. UGR âĂŸ16: A new dataset for the evaluation of cyclostationarity-based network IDSs. *Computers & Security* 73 (2018), 411–424.
[22] Frank J Massey Jr. 1951. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association* 46, 253 (1951), 68–78.
[23] John McHugh. 2000. Testing intrusion detection systems: a critique of the 1998 and 1999 Darpa intrusion detection system evaluations as performed by Lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)* 3, 4 (2000), 262–294.
[24] Yisroel Mirsky, Asaf Shabtai, Lior Rokach, Bracha Shapira, and Yuval Elovici. 2016. Sherlock vs moriarty: A smartphone dataset for cybersecurity research. In *Proceedings of the 2016 ACM workshop on Artificial intelligence and security*. ACM, 1–12.
[25] N. Moustafa and J. Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*. 1–6. https://doi.org/10.1109/MilCIS.2015.7348942
[26] Thuy TT Nguyen and Grenville J Armitage. 2008. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials* 10, 1-4 (2008), 56–76.
[27] Vern Paxson and Sally Floyd. 1995. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on networking* 3, 3 (1995), 226–244.
[28] Scott Renfro. [n. d.]. Mergecap Man Page. https://www.wireshark.org/docs/man-pages/mergecap.html. ([n. d.]). Accessed: 2019-08-11.
[29] Iman Sharafaldin, Amirhossein Gharib, Arash Habibi Lashkari, and Ali A Ghorbani. 2018. Towards a reliable intrusion detection benchmark dataset. *Software Networking* 2018, 1 (2018), 177–200.
[30] Richard Sharpe. [n. d.]. Editcap Man Page. https://www.wireshark.org/docs/man-pages/editcap.html. ([n. d.]). Accessed: 2019-08-11.
[31] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security* 31, 3 (2012), 357–374.
[32] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*. IEEE, 305–316.
[33] CSL Sony and Kenjiro Cho. 2000. Traffic data repository at the WIDE project. In *Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track*. 263–270.
[34] Anna Sperotto, Ramin Sadre, Frank Van Vliet, and Aiko Pras. 2009. A labeled data set for flow-based intrusion detection. In *International Workshop on IP Operations and Management*. Springer, 39–50.
[35] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. 2009. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. IEEE, 1–6.
[36] M. J. M. Turcotte, A. D. Kent, and C. Hash. 2017. Unified Host and Network Data Set. *ArXiv e-prints* (Aug. 2017). arXiv:1708.07518
[37] C Walsworth, E Aben, K Claffy, and D Andersen. 2015. The caida ucsd anonymized internet traces 2012,âĂİ. (2015).
[38] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. 2005. Automated traffic classification and application identification using machine learning. In *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05) l*. IEEE, 250–257.

Malicious Behavior Detection based on Extracted Features from APIs for Windows Platforms

Dima Rabadi[†] Kar Wai Fok[*] Zhongmin Dai[*] Teo Sin Gee[†]
[†]Institute for Infocomm Research (I²R), A*STAR, Singapore
[*]ST Engineering Ltd, Singapore
[†]{dimadsr, teo_sin_gee}@i2r.a-star.edu.sg, [*]{fok.karwai, dai.zhongmin}@stengg.com

ABSTRACT

Many malicious behavior detection approaches rely on dynamic features that are extracted from Application Programming Interfaces (APIs), which represent the run-time behavior of programs. Most API-based malicious behavior detection techniques highly focus on measuring the statistical features of API calls such as finding the frequency (e.g., how many times a specific API is called) or recognizing the sequence pattern of API calls. However, such detectors can be easily evaded and bypassed by malware authors who would interrupt the sequence by basically hooking and shuffling the API calls or deleting/inserting the irrelevant calls. Also, most proposed API-based malicious behavior detectors would either consider only the API calls (e.g., function names) without taking into account their arguments information (e.g., function parameters) or incur a prohibitive cost, such as requiring complex operations to deal with the arguments (e.g., proficient knowledge about the types of the arguments and/or powerful computers to extract them). As relying on API calls alone is insufficient to understand the purpose of the program, we propose a low-cost malicious behavior detection approach that can extract APIs dynamic features by studying the API calls together with their arguments using machine learning. Experimental results show that our approach achieves an accuracy of over 98.24% with two different datasets, and outperforms the state-of-the-art malicious behavior detection techniques.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**; **File system security**.

KEYWORDS

API calls; feature generation; malicious behavior analysis; dynamic analysis; anomaly-based detectors; machine learning

1 INTRODUCTION

Despite the significant increase in malicious behavior detection methodologies and the variety of anti-virus vendors, it has been observed that malware detection has improved by only 3 % between 2017 and 2018 [13]. Such a small improvement in the detection is because while researchers keep building new detection techniques, attackers also keep continuously generating and spreading new malware variants and zero-day attacks. Additionally, the number of malware distributing websites is increasingly sophisticated, where the Internet infrastructure and the interconnectivity between devices and services make it easy for attackers to spread malware. Thus, the amount of malware is increasing rapidly. Malicious behavior detection becomes a priority and is a significant issue in every security-aware organization. Once the malicious behavior is detected, the presence of the attacker or her adversarial motive to harm the computer would be predicted. As long as the prediction occurs on an early stage, that would help in discovering the attack and stopping it promptly. It is known in cyber-security that '*prevention is ideal, but detection is a must* [6]'. Therefore, we focus on malicious behavior detection while prevention is out of concern in this paper. In literature, the term malware is synonymous to malicious code/software, malcode, harmful scripts, etc. In this paper, this term refers to any malicious program or file that achieves the adversarial motive of the attacker.

There are two broad categories of malware detection approaches namely static and dynamic analysis, respectively. The main difference between them is that the former investigates the structure of malware's binary code without executing it, while the latter seeks its behavior during the execution. However, static analysis is vulnerable to many shortcomings, such as dealing with packing, compression and obfuscation techniques, defeating polymorphic malware and detecting unknown and zero-day attacks, to name a few. Various studies [3, 7, 12] have proven that dynamic analysis gives more accurate detection performance than the static and is resilient to the vulnerabilities mentioned above. There are mainly two fundamental approaches to dynamic analysis. The first specifies a predefined system state at a specific point of time (e.g., before running the malware sample) and then compares the modification presented on the system before and after that state. The second approach observes malware behavior by executing the malware on a well-monitored and isolated environment (e.g., sandboxing). Both analyze different extracted information of the system,

which includes a list of kernel requested services and function calls, newly created processes, changes made to the registry and the accessed/dropped files, etc.

Application Programming Interface (API) calls play a prominent role in both static and dynamic analysis. Using extracted features from API calls to understand malicious behavior has received significant attention from researchers [11, 12, 17, 19–21, 30]. API, as being a standard interface that takes requests from a user and informs the system (kernel mode) about what should be done, then returns the response to the user, is capable of holding enough information about the program and its behavior. In literature, API-based malicious behavior detection techniques have made their advances during three distinct directions. The first direction has started with the static analysis by extracting API call sequences and creating static-signatures out of them [24]. The second direction has helped in developing the static analysis by extracting not only the signatures but also some statistical properties including the frequency of calling specific API functions[1] [1, 28]. Such techniques mentioned above are non-resilient to attackers who can insert redundant and noisy API. As a result, researchers expand the API-based detection topic by moving their attention to more efficient techniques based on dynamic analysis (third direction). For example, in [12], Longest Common Subsequence (LCS) alignment is applied to extract the common patterns from the API call sequences. Similarly, in [19], sequence alignment using n-gram and searching algorithms are used to find malicious API patterns. However, these techniques are time-consuming and require tremendous resources. API patterns can also be evaded by basically hooking and/or shuffling the API sequence and inserting irrelevant calls.

Recently, various machine learning techniques (e.g., Random Forest (RF) [23] and Sequential Minimal Optimization (SMO) [29]) are used to distinguish malicious from benign behavior using API-based extracted features. Such machine learning techniques achieved outstanding performance and high accuracy detection. However, as the API arguments have various hetrogenious types (e.g., integers, strings, address pointers, etc.), most of the available machine learning techniques ignore the API arguments in their feature extraction operations or try to gather domain knowledge about the arguments which requires tedious and complex operations to process them. As a result, to deal with the complex API arguments feature extraction, other researches apply deep learning models (e.g., convolutional or recurrent neural network (CNN) and (RNN), respectively) for malicious behavior detection [18, 32]. A powerful and expensive computer (e.g., a Graphics Processing Unit (GPU)) would be needed to run such models.

In this paper, we explore a new low-cost direction of API-based dynamic analysis by studying the API calls together with their arguments (e.g., function parameters). The main reason for proposing this direction is because relying on API calls alone is insufficient to understand the purpose of the program. For example, malware authors would fake the API call by replacing it with another API call that belongs to the same category and passing the same arguments with it. Such replacement achieves the same adversarial motive of the attacker and performs the same purpose. Thus, an important research question that we seek to answer is whether a low-cost feature extraction of the API arguments without any domain knowledge about the arguments is sufficient to distinguish malicious from normal behavior. To answer this question, we propose two methods using the extracted API arguments; the first method deals with all arguments of each API call as one feature, whereas the second method deals with each argument of each API call as one feature. To the best of our knowledge, no research has studied the feature extraction technique presented in the second method. We conduct extensive measurements on 27 different malware families using two sets of balanced and imbalanced datasets. We use 2,112 and 9,003, benign and malicious samples, respectively. Our malicious behavior detection approach gives an accuracy of over 98.24 % and outperforms the state-of-the-art API-based dynamic analysis methods.

We make the following main contributions in this paper:

- We analyze around 11,115 samples and extract their API-based dynamic features.
- Based on the extracted features from the API calls and their arguments, we design two methods to distinguish malicious from normal activities on the system. The first method relies on the entire list of arguments of each API call (*Method 1*), while the second method generalizes each argument of each API call separately (*Method 2*).
- Unlike the API-based malicious behavior detection methods presented in the state-of-the-art, our approach is resilient against sequence interruption, as we do not rely on the sequence (order) of API calls. We conduct extensive experiments to evaluate the efficacy of our proposed methods in detecting malicious behavior using both balanced and imbalanced datasets.
- We verify by using real malware samples that API arguments are capable of detecting the malicious behavior. Moreover, our methods outperform the state-of-the-art with an accuracy of over 98.24%.

The rest of the paper is organized as follows. Section 2 reviews related work. The principle of the proposed detection methods is presented in Section 3. Section 4 introduces the datasets used in this paper and presents detailed experimental evaluation results. Section 5 compares our proposed detection methods with other recent API-based malicious behavior detection methods. Section 6 further discusses the limitations as well as the future work of this paper. Section 7 concludes.

2 RELATED WORK

In the past decade, a considerable amount of research on API feature extraction for malicious behavior detection has been conducted. Both static and dynamic methods are used to extract these features. In static methods, the API information is extracted from the portable executable (PE) format of the executable files. For instance, frameworks for analyzing and categorizing PE files based on their relevant API calls and data mining techniques are presented in [30] and [24], respectively. Despite their high detection accuracy, it has been proven that static features, such as finding the Cosine similarity between two PE files [26], detecting sequence patterns and calculating the frequency of the API calls, are insufficient for

[1]The terms API functions and calls will be used interchangeably in this paper.

malicious behavior detection as they can be easily misled by applying obfuscation tools [16]. To overcome this issue, researchers are currently exploring the dynamic methods [2, 12, 19, 28] as being more promising than the static.

In [2], Alazab *et al.* have implemented a method that consists of online and offline modules to extract both the spatial and temporal statistical features from the arguments and the sequences of API calls, respectively. These statistical features (e.g., variances, means, and pointers addresses) have helped in achieving a detection accuracy of 96.3 %. Apart from the high complexity of their method, the malware authors can easily affect the temporal features by faking API call sequences. In [8], Gupta *et al.* have used Detours library by Microsoft to hook Windows API calls (limited to the user level) and correlate their functions with 26 categories based on their purposes. In [28], the frequency counters of both the API calls and their arguments are calculated, then WEKA library [9] is used to classify between benign and malicious samples. Again, frequency-based detection tools can be evaded by adding more redundant API calls.

In [5], the Multiple Sequence Alignment (MSA) algorithm is used to find the representative API pattern from the API call sequences. Such work can also be evaded by increasing the API call sequence as MSA is not practical for a sequence longer than a certain threshold. Furthermore, in [12], LCS alignment is applied to extract the common patterns from the API call sequences. Similarly, in [19], sequence alignment using n-gram and searching algorithms are used to find malicious API patterns. As previously stated, such techniques are time-consuming and require tremendous resources. Most importantly, pattern-based detection tools can be evaded by merely hooking and/or shuffling the API sequence and inserting irrelevant calls.

Recent works have studied the effectiveness of using both dynamic and static analyses together in detecting malicious behavior. In [17], Bruce *et al.* have proposed a model to construct API sequence features based on both static and dynamic analyses. Next, the features are fed into various similarity-based machine learning algorithms (e.g., LCS, Cosine similarity and Minkowski distance) to train the model and then classify the malicious and benign samples. In a seminal work [10], MalDAE, a malware detection framework that correlates dynamic and static features from the API call sequences is established. Experimental results of both works have shown that although the syntax of the dynamic and static API call features is different, there is a semantic mapping and clear relation between them. Nevertheless, the hybrid sequence features extracted in both works rely on the API calls alone. Thus, malware authors can easily modify such feature extraction models by interrupting the sequences.

Other works have used API calls and their related properties to classify malware to its family (malware classification). For example, in [4], Cheng *et al.* have used information retrieval theory and TF-IDF weighting to generate dynamic features using system calls and their arguments. Similarly, in [22], Rieck *et al.* have implemented the MIST framework to classify malware with similar behavior. The MIST framework has two levels, the first level uses prioritized system calls alone, where the second level uses both the prioritized systems calls and their arguments.

In contrast, the novel detection approach proposed in this paper extracts dynamic features from both the API calls and their arguments and applies machine learning techniques to distinguish malicious from benign behavior. The possibility of detecting malicious behavior based on both the API calls and their arguments has been slightly studied before [4, 23]. Such studies highly depend on either the order (sequence) or the frequency of API calls as will be explained in Section 5. However, our approach will no longer depend on the order of API calls. It will be resilient against malware mutation and obfuscation techniques (e.g., reordering and/or repeating certain API calls many times). A comparison between our approach and the aforementioned API-based malicious detection approaches will be presented in Section 5.

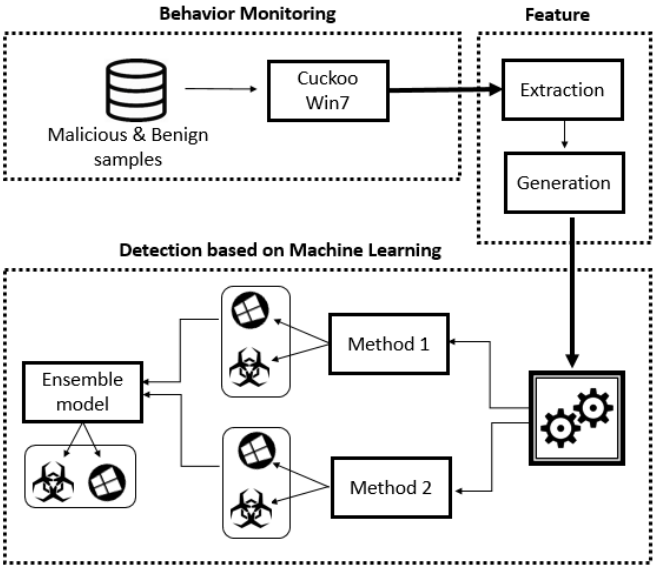

Figure 1: Schematic overview of the proposed malicious behavior detection approach. It has three main components, sample behavior monitoring, tokenization extraction and generation and lastly the machine learning detection methods.

3 METHODOLOGY

3.1 Overview

This section presents the working principles of the proposed approach. The basic operations of our approach are illustrated in Figure 1. As shown in the figure, our approach has three main components. In behavior monitoring, the behavior of the malicious and benign samples is monitored by executing them on Cuckoo Sandbox[2]. In this paper, Cuckoo's host and guest are Ubuntu 16.04 LTS and Windows 7 64-bit, respectively. In the second component, the sequence of API calls together with their arguments are extracted from Cuckoo's behavioral analysis report (e.g., report.json). As API arguments are highly dependent on system resources, feature generation methods are applied to generate the arguments in a way that would be efficiently processed by machine learning techniques; this component will be demonstrated via examples in Section 3.2.

[2]https://cuckoosandbox.org/

The generalized features are converted into bit-vector by utilizing two tokenization methods. The first method uses each API call and its arguments as one token (feature). Whereas the second uses each argument element of each API call as one token. Finally, in the third component, machine learning detection methods based on Support Vector Machine (SVM) are trained, resulting in classifying samples as benign or malicious.

3.2 Feature Generation (tokenization)

After executing each sample (e.g., benign and malicious) in Cuckoo, a behavioral analysis report for each sample will be collected. Then, the API calls and their arguments are extracted from the report. For each call, two methods will be used to generate the features (tokenizations) by using the following processes:

(1) We remove API suffixes such as *Ex, A, W, ExA,* and *ExW* from the API function names, thereby ensuring the resilience of the extracted features against the conflict of using multiple versions of the same API call. For example, the two API calls *FindFirstFileExW* and *FindFirstFileExA* will be merged under the same API call, *FindFirstFile*. Mainly these APIs perform the same functionality where the difference is in the format of their arguments. For example, the former receives its arguments in Unicode format (*ExW*), while the latter receives its arguments in ANSI coded format (*ExA*).

(2) The API arguments are generalized by performing different processes on different argument types. To name a few of these processes, (i) If the directory path found in the argument is System32, then the argument will be categorized as *sys_dir*; otherwise, it will be added to the *other_dir* category. (ii) The extension of the file is checked whether it is one of the most popular Windows file types used by malware [27]. (iii) Similarly, if the argument is registry key (e.g., *regkey* such as *BootExecute, Winlogon, Run, RunOnce keys*, etc.), then we check whether it is one of the most common registry keys exploited by malware.

(3) We convert API arguments with integer type to logarithmic bins. Thus, actual numbers are abstracted. For example, if the argument value represents an integer larger than 0, a prefix *numB* plus the logarithmic bin of the value is added to the feature. However, if the argument value is a negative number, a prefix of *neg_* is added before the logarithmic bin in the feature. In addition, to distinguish 0 from the number in the range of $(0-10)$, 0 is tagged as *num*0 instead of *numB*0. Moreover, arguments with concrete values will not be changed in the feature. For example, *allocation_type* argument in *NtAllocateVirtualMemory* API call has a concrete number equals 8192 that represents a *MEM_RESERVE*. Such constant value will not be changed in this process.

These above processes enhance the generated features by generalizing the arguments in a way that would be easier to be processed by machine learning.

After applying the previous processes, the API calls will be abstracted as a set of unique strings in the form of an API function name and its argument values. In the first method, each API call and the list of its arguments are considered as one feature. Whereas in the second method, each API call and each element of its arguments are considered as one feature. In other words, *Method 1* and *Method 2* features are constructed using the following formulas, respectively.

- ***Method 1***: *"API(0): arg(0); arg(1)...; arg(j0)", "API(1): arg(0); arg(1)...; arg(j1)", ..., "API(i): arg(0); arg(1); ...; arg(ji)"*
- ***Method 2***: *"API(0): arg(0)", "API(0): arg(1)", ..., "API(0): arg(j0)", "API(1): arg(0)", "API(1): arg(1)", ..., "API(1): arg(j1)", ..., "API(i): arg(0)", "API(i): arg(1)", ..., "API(i): arg(ji)"*

Where i and j are the numbers of API calls and arguments of each API call, respectively. Listing 1 illustrates a numerical example that shows a pair of API calls ($NtQueryValueKey$ and $NtAllocateVirtualMemory$) and their arguments values extracted from Cuckoo's analysis report (using JSON format). As shown, *reg_type* and $information\_class$ arguemnts have values of 0 and 1, respectively. The feature generation processes presented above convert these values to *num*0 and *numB*0 (e.g., numB+ $\log_{10}$ 1), respectively. Table 1 shows the generated features of both methods for the entire list of arguments shown in Listing 1.

Listing 1: Two API calls and their arguments snippet from Cuckoo's analysis report (JSON format).

```
\\report.json
"api": "NtQueryValueKey",
"return_value": 3221225524,
"arguments": {
"value": " ",
"reg_type": 0,
"information_class": 1,
"regkey": "HKEY_LOCAL_MACHINE/SYSTEM/
ControlSet001/Control/NlsExtendedLocale
/en-US"},

"api": "NtAllocateVirtualMemory",
"return_value": 0,
"arguments": {
"process_identifier": 3336,
"region_size": 524288,
"stack_dep_bypass": 0,
"stack_pivoted": 0,
"heap_dep_bypass": 0,
"protection": 4,
"allocation_type": 8192},
```

3.3 Detection based on Machine Learning Techniques

After extracting and generating the API-based dynamic features in the previous component, the Hashing Vectorizer function [14] is used to generate a bit-vector out of these features (e.g., the bit-vector length equals 2^{20}). As discussed before, *Method 1* and *Method 2* serve the same purpose but the difference between them lies in the tokenization strategies (i.e., using the entire list of arguments versus each argument separately).

To this end, our proposed methods are evaluated using the SVM learning. Our performance metrics include precision, recall and F-measure metrics. An ensemble method [15] is also implemented.

Table 1: The generated features (tokenizations) from the API calls and their arguments presented in Listing 1 . *Method 1* uses each API call and its entire list of its arguments as one token, where *Method 2* uses each argument of each API call as one token separately.

Method	Features
Method 1	"NtQueryValueKey:0=numB0;NtQueryValueKey:1=0;NtQueryValueKey:2=other_reg;NtQueryValueKey:3=num0", "NtAllocateVirtualMemory:0=8192;NtAllocateVirtualMemory:1=num0;NtAllocateVirtualMemory:2=numB3; NtAllocateVirtualMemory:3=numB0;NtAllocateVirtualMemory:4=numB5;NtAllocateVirtualMemory:5=num0; NtAllocateVirtualMemory:6=num0"
Method 2	"NtQueryValueKey:0=numB0", "NtQueryValueKey:1=0", "NtQueryValueKey:2=other_reg", "NtQueryValueKey:3=num0", "NtAllocateVirtualMemory:0=8192", "NtAllocateVirtualMemory:1=num0", "NtAllocateVirtualMemory:2=numB3", "NtAllocateVirtualMemory:3=numB0", "NtAllocateVirtualMemory:4=numB5", "NtAllocateVirtualMemory:5=num0", "NtAllocateVirtualMemory:6=num0"

The ensemble method uses both features of *Method 1* and *Method 2* and combines their performance measures scores (e.g., averaging the scores). The evaluation results of these proposed methods are discussed in detail in Section 4.

4 PERFORMANCE EVALUATION AND DISCUSSION

4.1 Datasets

In this section, we introduce the malicious and benign datasets used in the evaluation of the proposed malicious behavior detection methods in this paper. The datasets consist of 9,003 malicious samples belonging to 27 distinct families and 2,112 benign samples.

Benign Samples: The benign samples are obtained from six different sources as follows: (1) After immediately installing a fresh version of Windows 7, we extract the Native Windows executable and (2) the Dynamic Link Library (DLL) files located in $C:\backslash Windows\backslash Systems32$ directory. (3) We download the APIMDS dataset[3]. (4) We use websites for free-to-try legal downloads (e.g., download.cnet.com and softpedia.com). We download (5) the top 300 portable applications for Windows [4] and (6) the top 43 applications from the File Hippo Website [5].

Malicious Samples: The malicious samples are obtained using daily downloaded samples from the Malshare website[6]. The period of collection was from February to May 2019.

All samples are validated using VirusTotal [31]. Table 2 describes the number of samples for each benign and malicious sources and a summary of the malware families/categories distribution, including their percentages in the total dataset.

4.2 Experimental Results

In this section, we investigate the performance of the proposed methods and the efficacy of their generated features in detecting malicious behavior. Two sets of experiments are performed. In the first set, we use a balanced dataset with 4,224 samples equally divided between malicious and benign samples. In the second set of

[3]http://ocslab.hksecurity.net/apimds-dataset
[4]https://portableapps.com/
[5]https://filehippo.com/software/desktop/
[6]https://malshare.com/

Table 2: Description of the datasets used in our experiments.

Type	Category	No. of samples	(%)
Malicious	Adwares	104	0.94
	Agent	1019	9.17
	Andromeda	66	0.59
	AutoIt	92	0.83
	Autorun	10	0.09
	Backdoor	360	3.24
	CoinHive	101	0.91
	CoinMiner	231	2.08
	Cryxos	223	2.01
	Delf	133	1.2
	Emotet	156	1.4
	Fareit	1158	10.42
	Gandcrab	348	3.13
	Javascript Trojans	248	2.23
	Kryptik	216	1.94
	MagentoStealer	43	0.39
	Mansabo	49	0.44
	Midie	50	0.45
	Noon	55	0.49
	Parite	14	0.13
	Ponystealer	59	0.53
	Razy	1801	16.2
	Symmi	61	0.55
	Ulise	97	0.87
	Ursnif	314	2.83
	Zusy	78	0.7
	Other	1917	17.25
	Sub Total	9003	81.01
Benign	Windows executable	327	2.94
	DLL files	1069	9.62
	APIMDS	200	1.8
	CNET	173	1.56
	Portable applications	300	2.7
	File Hippo	43	0.39
	Total	11115	-

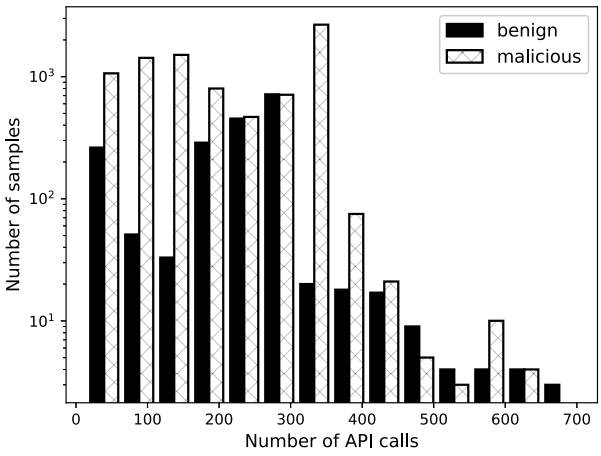

Figure 2: Number of API calls in benign and malicious datasets.

experiments, we use 9,003 and 2,112 malicious and benign samples, respectively. In Figure 2, we plot the number of API calls in benign and malicious datasets. As shown in the figure, there is no direct relation between the number of API calls and the classification of the samples as being benign or malicious. Some malicious samples need less than 50 API calls to achieve their harmful goals, whereas, in the regular operation of Windows, some benign samples need more than 500 calls to complete their purposes.

In each experiment, 80% of the dataset is used for training and the remaining 20% for testing the SVM methods (*Method 1*, *Method 2* and the ensemble method). We start the evaluation by measuring how well *Method 1* and *Method 2* are able to distinguish the malicious from the benign samples. In the first set of experiments (balanced dataset), we report the F-measure, precision, recall and the misclassification scores (e.g., false positives and negatives, FP and FN, respectively) for the benign and malicious samples. Table 3(a) shows a comparison between *Method 1* and *Method 2* performance results. We also show the performance of the ensemble method, which uses both *Method 1* and *Method 2* features on the same dataset. As shown in the table, the ensemble method helps in improving the detection of malicious behavior and increases the precision to 98.12%. Hence, using the features of both methods together is considered more effective than using each method separately.

Similarly, for the second set of experiments (imbalanced dataset), Table 3(b) shows a comparison between *Method 1* and *Method 2* and their ensemble method. Interestingly, the ensemble method increases the detection to 98.24%. At this end, under both the balanced and imbalanced datasets, the ensemble method, where the features of both *Method 1* and *Method* 2 are used, could reduce the false positives and negatives. A comparison between the performance of the two methods and discussion on the misclassification of our approach will be presented shortly.

4.3 *Method 1* versus *Method 2*

As shown in Tables 3(a) and 3(b), the performance of the two methods (*Method 1* and *Method 2*) is different. As described in Section 3, *Method 1* and *Method 2* serve the same purpose where the difference between them lies in the tokenization strategies. *Method 1* uses the entire list of arguments of each API call, where *Method 2* uses each argument separately. These tokenization strategies would enhance each method differently. For example, if the number of API calls is small, but the number of the passed arguments of each API call is high, then *Method 2* would provide a better performance, as the large number of arguments of each call compensates the small number of calls. However, if both the number of API calls and their arguments are small, then using features of both methods together would provide better detection. For example, in Table 3(b), *Method 1* wrongly classified few benign samples as malicious which caused a false positive of 0.056 %. We study the features of these misclassified samples. The number of API calls of one of these samples is only 7. Thus, only 7 features are generated using *Method 1*. In contrast, *Method 2* could generate around 60 features out of the 7 calls and correctly classify this sample as benign (FP of *Method 2* = 0.054).

4.4 Misclassifications (FP & FN)

Now we discuss the misclassifications of our approach. The experimental results presented above show that our approach distinguishes benign from malicious behavior with high accuracy. Where the ensemble method which uses the features of both *Method 1* and *Method 2*, could reduce the false positives to 0.054. However, our approach, like any other detection approaches, misclassifies a small number of samples that causes the FN as presented in Tables 3(a) and 3(b). We check the misclassified malicious samples, which caused the FN of our approach. Interestingly, the number of API calls is only two, where each API call takes only one argument (e.g., "__exception__:0=numB0", "NtTerminateProcess:0=numB2"). Thus, both *Method 1* and *Method 2* have only two features. Due to the small numbers of extracted features from these samples, our approach fails to correctly classify them as malicious, which provides the small FN values, as shown in Tables 3(a) and 3(b).

5 COMPARISON WITH STATE-OF-THE-ART

In this section, we compare our approach with other recent API arguments-based malicious detection approaches (e.g., [1, 4, 20, 22, 23, 28]) in terms of accuracy, limitations and required APIs information. Researchers have used API calls and their arguments related properties to establish behavior detection models and classify malware to its family [4, 22, 23]. For malware families classification, in [4], Cheng *et al.* have used information retrieval theory and TF-IDF weighting[7] to generate dynamic features using system calls and their arguments. Similarly, in [22], Rieck *et al.* have implemented the malware instruction set (MIST) framework to classify malware with similar behavior. The MIST framework has two levels; the first level uses prioritized system calls alone, where the second level uses both the prioritized systems calls and their arguments.

Moreover, in [23], Salehi *et al.* have used a subset of 126 API calls, their arguments and/or return values to detect malicious activity.

[7] https://en.wikipedia.org/wiki/Tf–idf

Table 3: The evaluation results of the balanced and imbalanced datasets including the precision, recall, F-measure and the false positives and negatives (FP and FN, respectively) of the benign and malicious samples.

(a) The evaluation results of the balanced dataset

Method	Type	FP	FN	Precision	Recall	F-m
	Benign	0.028	-	98.56	97.16	97.86
Method 1	Malicious	-	0.014	97.20	98.58	97.88
	Total	**0.028**	**0.014**	**97.88**	**97.87**	**97.87**
	Benign	0.026	-	98.56	97.40	97.98
Method 2	Malicious	-	0.014	97.42	98.58	98.00
	Total	**0.026**	**0.014**	**97.99**	**97.99**	**97.99**
	Benign	0.026	-	98.80	97.40	98.10
Ensemble	Malicious	-	0.012	97.43	98.82	98.12
	Total	**0.026**	**0.012**	**98.12**	**98.11**	**98.11**

(b) The evaluation results of the imbalanced dataset

Method	Type	FP	FN	Precision	Recall	F-m
	Benign	0.056	-	96.14	94.31	95.22
Method 1	Malicious	-	0.008	98.67	99.11	98.89
	Total	**0.056**	**0.008**	**98.19**	**98.20**	**98.19**
	Benign	0.054	-	95.68	94.55	95.11
Method 2	Malicious	-	0.009	98.73	99.00	98.86
	Total	**0.054**	**0.009**	**98.15**	**98.16**	**98.15**
	Benign	0.054	-	96.14	94.55	95.34
Ensemble	Malicious	-	0.008	98.73	99.11	98.92
	Total	**0.054**	**0.008**	**98.24**	**98.25**	**98.24**

As previously stated in Section 2, [4, 23] use pattern recognition techniques to find a mutual sequence of APIs and their arguments. However, the sequence of API calls and their arguments (i.e., pattern) can be interrupted by inserting and/or deleting API calls.

In [28], the frequency metric of specific API calls and their arguments has been used to differentiate between benign and malicious behavior. Also, Yong *et al.* [20] have used frequent itemsets of API calls and their arguments to detect malicious behavior. Similarly, Faraz *et al.* [1] have used statistical features of API calls and their arguments including frequency, mean and size of parameters, to detect the malicious behavior of programs. Malware authors can easily bypass these frequency-based approaches by adding and/or removing API calls hence altering the frequency value.

Our approach is different from the aforementioned studies. First, we use the occurrence of the API calls and their arguments instead of the sequence (e.g., order). Therefore, it is resilient to obfuscation techniques and malware mutation (e.g., reordering or repeating certain API calls and/or arguments many times). Second, the statistical features such as frequency are not involved in our approach. Third, we use feature generation functions to enhance the extracted features by generalizing the arguments for processing by machine learning rather than using specific argument values. Finally, none of the existing approaches have studied the possibility of using each argument element of each API call separately as presented in *Method 2*. These advantages render our approach against the scalability issue of using high dimensional feature space which requires high computational complexity and memory consumption. Our approach outperforms the state-of-the-art API arguments-based malicious detection methods. We summarize the comparison between our approach and the above-mentioned related studies in Table 4.

Table 4: Comparison with similar API arguments-based detection methods.

Approach	F-measure(%)	Method	API information	Limitations
Cheng *et al.* [4]	97.7	IR (TF-IDF)	sequential Windows API calls and parameters	sequence[1]
Salehi *et al.* [23]	97.9	machine learning (RF, J48, SMO, BLR)	sequence of API calls, parameters and/or return value	only a subset of 126 API calls were used, sequence[1]
Rieck *et al.* [22]	96.7	n-gram	system calls and their arguments	system calls only were used
Tian *et al.* [28]	97.0	Weka library (SVM, random forest, decision table and instance-based classifier IB1)	frequency of API calls and arguments traces separately	frequency[2]
Yong *et al.* [20]	94.7	n-gram	frequent itemsets API calls sequences and/or their arguments	frequency[2]
Faraz *et al.* [1]	96.6	SMO, RIPPER, NB, J48, IBk	statistical features of API calls and their arguments (e.g., frequency, mean and size parameters)	frequency[2]
Our approach	*Method 1:* 98.12 *Method 2:* 98.24	SVM	*Method 1*: API call and its entire list of arguments, *Method 2*: API call and each of its argument element separately	Section 6

[1] Sequence: malware authors can bypass sequence-based malicious detection techniques by inserting and/or deleting API calls.

[2] Frequency: the frequency of API calls and their arguments can be interrupted by adding and/or removing API calls hence altering the frequency value.

6 LIMITATIONS AND FUTURE WORK

This section discusses the limitations and the future work of our proposed malicious behavior detection methods. First, the experiments are conducted using the Cuckoo sandbox. Thus, the APIs studied in this paper are limited to the list of API calls extracted by Cuckoo [25]. In the future, we plan to design a real-time malicious behavior detector that can extract the API calls simultaneously while the samples are executed on the system.

Second, we have analyzed and monitored the behavior of the samples only on Windows 7. In the future, we plan to increase the scale of the experiments to cover the new versions of Windows (e.g., Windows 10).

Third, we have used the SVM machine learning technique to evaluate our methods on both balanced and imbalanced datasets. Although SVM provides a very promising detection accuracy of over 98.24 %, we plan to evaluate our methods on different machine learning techniques such as Naive Bayes and Random Forest.

Lastly, as Android platforms also work based on API calls, we also plan to test the performance of our proposed methods in detecting malicious behavior on Android platforms.

7 CONCLUSIONS

This paper investigated a new low-cost direction of extracting API-based dynamic features by studying the API calls together with their arguments. We designed a new model that relies on two different methods of extracting APIs arguments; the first method deals with the entire list of arguments of each API call as one feature, whereas the second novel method deals with each argument of each API call as one feature.

We verified the performance of the proposed methods and the efficacy of their generated features in detecting malicious behavior by using a reasonably large dataset of 9,003 malicious samples belonging to 27 distinct families and 2,112 benign samples, respectively. We implemented an ensemble method that uses the generated features of both methods to enhance the performance of our approach and increase its accuracy.

We compared our approach with other recent API arguments-based malicious detection approaches in terms of accuracy, limitations and required APIs information. Our experimental results showed that our malicious behavior detection approach gave an accuracy of over 98.24 % and outperformed the state-of-the-art. Our

approach has promise for wide adoption in API-based malicious behavior detection for Windows platforms.

8 ACKNOWLEDGMENT

The authors wish to thank Dr. Micheal Clifford and Dr. Micheal Collins, and the anonymous reviewer for providing valuable feedback on this work. We thank the KPMG Forensic team and the I^2R Cyber Security Department for their helpful discussion on this work.

REFERENCES

[1] Faraz Ahmed, Haider Hameed, M Zubair Shafiq, and Muddassar Farooq. Using spatio-temporal information in api calls with machine learning algorithms for malware detection. In *Proceedings of the 2nd ACM workshop on Security and artificial intelligence*, pages 55–62. ACM, 2009.

[2] Mamoun Alazab, Sitalakshmi Venkataraman, and Paul Watters. Towards understanding malware behaviour by the extraction of api calls. In *2010 Second Cybercrime and Trustworthy Computing Workshop*, pages 52–59. IEEE, 2010.

[3] Silvio Cesare and Yang Xiang. *Software similarity and classification*. Springer Science & Business Media, 2012.

[4] Julia Yu-Chin Cheng, Tzung-Shian Tsai, and Chu-Sing Yang. An information retrieval approach for malware classification based on windows api calls. In *2013 International Conference on Machine Learning and Cybernetics*, volume 4, pages 1678–1683. IEEE, 2013.

[5] In Kyeom Cho and Eul Gyu Im. Extracting representative api patterns of malware families using multiple sequence alignments. In *Proceedings of the 2015 Conference on research in adaptive and convergent systems*, pages 308–313. ACM, 2015.

[6] Eric Cole. Taming the beast: Preventing/detecting insider threat. URL: https://www.darkreading.com/risk/taming-the-beast-preventing-detecting-insider-threat/d/d-id/1134832.

[7] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, 44(2):6, 2012.

[8] Sanchit Gupta, Harshit Sharma, and Sarvjeet Kaur. Malware characterization using windows api call sequences. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 271–280. Springer, 2016.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[10] Weijie Han, Jingfeng Xue, Yong Wang, Lu Huang, Zixiao Kong, and Limin Mao. Maldae: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *Computers & Security*, 83:208–233, 2019.

[11] Kazuki Iwamoto and Katsumi Wasaki. Malware classification based on extracted api sequences using static analysis. In *Proceedings of the Asian Internet Engineeering Conference*, pages 31–38. ACM, 2012.

[12] Youngjoon Ki, Eunjin Kim, and Huy Kang Kim. A novel approach to detect malware based on api call sequence analysis. *International Journal of Distributed Sensor Networks*, 11(6):659101, 2015.

[13] Malwarebytes Labs. Malwarebytes labs. URL: https://resources.malwarebytes.com/files/2019/01/Malwarebytes-Labs-2019-State-of-Malware-Report-2.pdf.

[14] Malwarebytes Labs. Scikit-learn machine learning in python. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.HashingVectorizer.html.

[15] Scikit learn Developers. Ensemble methods. URL: https://scikit-learn.org/stable/modules/ensemble.html.

[16] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 421–430. IEEE, 2007.

[17] Bruce Ndibanje, Ki Hwan Kim, Young Jin Kang, Hyun Ho Kim, Tae Yong Kim, and Hoon Jae Lee. Cross-method-based analysis and classification of malicious behavior by api calls extraction. *Applied Sciences*, 9(2):239, 2019.

[18] Razvan Pascanu, Jack W Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1916–1920. IEEE, 2015.

[19] Abdurrahman Pektaş and Tankut Acarman. Malware classification based on api calls and behaviour analysis. *IET Information Security*, 12(2):107–117, 2017.

[20] Yong Qiao, Yuexiang Yang, Lin Ji, and Jie He. Analyzing malware by abstracting the frequent itemsets in api call sequences. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 265–270. IEEE, 2013.

[21] Chandrasekar Ravi and R Manoharan. Malware detection using windows api sequence and machine learning. *International Journal of Computer Applications*, 43(17):12–16, 2012.

[22] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[23] Zahra Salehi, Ashkan Sami, and Mahboobe Ghiasi. Maar: Robust features to detect malicious activity based on api calls, their arguments and return values. *Engineering Applications of Artificial Intelligence*, 59:93–102, 2017.

[24] Ashkan Sami, Babak Yadegari, Hossein Rahimi, Naser Peiravian, Sattar Hashemi, and Ali Hamze. Malware detection based on mining api calls. In *Proceedings of the 2010 ACM symposium on applied computing*, pages 1020–1025. ACM, 2010.

[25] Cuckoo Sandbox. Hooked apis and categories in cuckoo. URL: https://github.com/cuckoosandbox/cuckoo/wiki/Hooked-APIs-and-Categories/.

[26] Igor Santos, Felix Brezo, Javier Nieves, Yoseba K Penya, Borja Sanz, Carlos Laorden, and Pablo G Bringas. Idea: Opcode-sequence-based malware detection. In *International Symposium on Engineering Secure Software and Systems*, pages 35–43. Springer, 2010.

[27] SensorsTechForum. Most popular windows file types used by malware (2018). URL: https://sensorstechforum.com/popular-windows-file-types-used-malware-2018/.

[28] Ronghua Tian, Rafiqul Islam, Lynn Batten, and Steve Versteeg. Differentiating malware from cleanware using behavioural analysis. In *2010 5th international conference on malicious and unwanted software*, pages 23–30. IEEE, 2010.

[29] Dolly Uppal, Rakhi Sinha, Vishakha Mehra, and Vinesh Jain. Malware detection and classification based on extraction of api sequences. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2337–2342. IEEE, 2014.

[30] R Veeramani and Nitin Rai. Windows api based malware detection and framework analysis. In *International conference on networks and cyber security*, volume 25, 2012.

[31] VirusTotal. Hooked apis and categories in cuckoo. URL: https://www.virustotal.com/#/home/upload.

[32] Zhaoqi Zhang, Panpan Qi, and Wei Wang. Dynamic malware analysis with feature engineering and feature learning. *arXiv preprint arXiv:1907.07352*, 2019.

Generating targeted e-mail at scale using neural machine translation

Chumki Basu
Perspecta Labs
Basking Ridge, NJ
cbasu@perspectalabs.com

Sridhar Venkatesan
Perspecta Labs
Basking Ridge, NJ
svenkatesan@perspectalabs.com

Cho-Yu Jason Chiang
Perspecta Labs
Basking Ridge, NJ
jchiang@perspectalabs.com

Nandi Leslie
US CCDC Army Research Laboratory
Adelphi, Maryland
nandi.o.leslie.ctr@mail.mil

Charles Kamhoua
US CCDC Army Research Laboratory
Adelphi, Maryland
charles.a.kamhoua.civ@mail.mil

ABSTRACT

Advances in deep learning, specifically in Neural Machine Translation (NMT) [1, 20, 28], have opened up new application opportunities for generating human-readable text by machines. In this paper, we discuss one such application – the generation of spearphishing e-mail at scale – that has ramifications for cybersecurity. To this end, the ability to characterize text generated by people and by machines is important, and specific characteristics of text could be used as differentiating features. For example, we could evaluate whether the generated text is grammatically well-formed or semantically relevant to some topic. However, existing metrics for evaluating NMT models do not adequately address these needs. As a result, we propose new metrics that capture both well-formedness and topical relevance to evaluate the quality of text. We show that the proposed metrics corroborate the findings of existing metrics and provide additional insights. Although the metrics consistently indicate that text generated by state-of-the-art NMT models possess grammatical structure, the machine-generated text differ in performance with respect to the metrics when compared to human-generated text. By using these new metrics, we assessed whether the automatically-generated content would pass a spam/spearphishing filter [6]. The filter performs poorly in distinguishing between human-generated and machine-generated text indicating the robustness of existing models.

KEYWORDS

machine learning-driven cyber offense, machine learning-based defense and response, phishing, neural machine translation, metrics

ACM Reference Format:
Chumki Basu, Sridhar Venkatesan, Cho-Yu Jason Chiang, Nandi Leslie, and Charles Kamhoua. 2019. Generating targeted e-mail at scale using neural machine translation. In *DYNAMICS '19: DYnamic and Novel Advances in Machine Learning and Intelligent Cyber Security (DYNAMICS) Workshop, December 09–10, 2019, San Juan, Puerto Rico.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Human beings are adept at noticing important characteristics of language. We use semantic cues to follow and respond to a thread of conversation. We recognize sentences that are grammatically correct or nearly correct. However, designing computer programs to perform these tasks remains a challenging task.

Recent advances in the field of natural language processing using deep learning techniques include Neural Machine Translation (NMT) approaches for language translation, generation, summarization, etc. Advances in the automated generation of text by machines necessitate the design of approaches to evaluate the content and structure of the generated text. With burgeoning new applications such as automated news annotations [32], researchers are calling for the responsible use of this technology – e.g., disclosing whether annotations are generated by people or by machines.

We are interested in two characteristics of text that pertain to realism: whether the text is human-generated, and in the case of e-mail, whether it also conforms to the interests of the intended recipients. For the former, we compare, collectively, the performance of automated methods using new and existing structural and semantic similarity metrics to human performance using the same evaluation metrics; for the latter, we propose a semantic similarity metric that measures the pairwise intersection of topic-bearing terms between the generated text and the source text.

In this paper, we discuss an application of automated text generation as a cybersecurity offensive capability: the generation of spearphishing e-mail at scale customized to the interests of individuals. Consider an example scenario where an attacker initiates a conversation on a recent topic of interest to an individual in an e-mail, which contains a link to a seemingly benign but malicious website or a malware attachment. We show an attacker may train an NMT model to automatically generate topically-relevant content, given an initial collection of similar content pairs or paraphrases, and we evaluate the performance of three NMT models on this task. The content should not duplicate any previous e-mail.

In addition to the offensive capability, our approach to generating new content from paraphrases can also be used by network administrators in corporate settings to improve the security awareness of their employees. They can use these models to generate and send

phishing e-mail to their employees and evaluate the employees' awareness in identifying and handling spearphishing e-mail.

Since most e-mail threads are comprised of questions and responses, in this research, we consider the problem of generating grammatically-correct, topically-relevant questions in spearphishing e-mail. How well we achieve our objective of generating questions determines whether the spearphishing attack would be viable. How we characterize the generated content, i.e., how we measure 1) the semantic similarity between the generated question and the content of the source text and 2) the well-formedness of the question, will determine if these measures could be used to determine if the content was generated automatically. In order to characterize the content, we need to go beyond the traditional metrics used to evaluate machine translation, which are based on measures of word overlap between the generated content and some reference text.

Our contributions in this paper include (1) learning how to generate sentences from paraphrases and (2) developing new metrics for evaluating the generated sentences with respect to realism. To the best of our knowledge, this is the first work that requires very little text input for training a "text generator" that generates variations of the input text. We show that our approach can generate spearphishing e-mail at scale and the proposed metrics provide insight on evaluating text generation performance. We also show that more than 87% of the generated e-mail is flagged as benign when we use the existing NMT models to generate spearphishing e-mail. The text generated by state-of-the-art NMT models received similar scores across the metrics we used, but the scores are inferior to those received by human-generated text.

2 PROBLEM DEFINITION

We define the problem of sentence generation as a learning problem:

$$f(\textit{key sentence or key words}) \rightarrow \textit{sentence}$$

Given a key sentence or keywords, our goal is to predict a sequence of words that are previously unseen and constitute a sentence. Since questions in an e-mail are meant to engage the recipient(s), we initially focus our work on question generation. Specifically, we restrict our input and output sentences to questions. Question generation using NMT is a topic of recent research [8] with relevance to the generation of educational content, starting up chatbot conversations, etc. However, the models we evaluate are agnostic to sentence type – they, simply, map an input sequence of words to an output sequence of words.

Since we are using NMT models, and translation is a type of paraphrasing, we believe it is natural to model sentence generation using paraphrases. This also mimics how we learn. Young children learn to imitate what they hear, directly repeating words and phrases. We can formulate this as a retrieval problem. Over time, however, we learn to understand by reading text and paraphrasing the contents. We formulate this as a generative problem, in which we learn to generate new, unseen text that is a function of what we have read and learned.

3 USE CASE

In recent years, spearphishing attacks have migrated from broadcast spam e-mail to personalized e-mail targeted at individuals. Inherently, it is difficult for spam filters to detect these kinds of attacks. In order to come up with alternative defenses for such an attack, we must, first, understand the different attack scenarios. Then, we propose a set of security requirements for realistic exchange of text, which would enable us to identify automatically-generated content.

Interest in the generation of targeted content for individuals has been studied for decades, particularly in the field of recommender systems. Types of recommended content range from targeted advertising based on observation of online activity to very specific recommendations of technical content. To generate very specific content, the attacker must dig deeper than just social networks. Rather, an attacker who has access to the e-mail or web pages of an individual could extract sentences containing key topics of interest.

By clustering either the individual's e-mail log or content from an online profile, the attacker uses unsupervised learning and *tf-idf*-based (term frequency-inverse document frequency) [24] techniques to find related sentences or paraphrases. Given just this data, the attacker can apply NMT tools to automatically generate unseen sentences, possibly fooling the targeted individual into thinking the sentences originated from someone who knows the individual.

We posit that content from a "real" source should have certain characteristics. First, the content should be topically relevant to an individual. If the content is off-topic or on a general topic (e.g., a distraction e-mail), then it is most likely to be ignored by the individual. Second, the content should be well-formed. Consider, for example, that grammatical and spelling mistakes are indicators of spam. From a security perspective, satisfying these characteristics are requirements that increase the chances of successful spearphishing. To the best of our knowledge, formal metrics for measuring these characteristics is an open research problem in the spam/spearphishing e-mail detection literature [6, 14].

4 BACKGROUND AND RELATED WORK

Our work spans multiple disciplines including cybersecurity, artificial intelligence, and natural language processing. Therefore, we present our work in the context of research on spam/spearphishing filtering as well as sequence to sequence learning for NMT. We also discuss question generation, one of its application areas – conversational agents, and evaluation metrics.

4.1 Spam/spearphising content generation and filtering

Generative approaches for the creation of spearphishing content have been discussed in recent literature [2, 5, 25, 26]. Seymour and Tully [25, 26] present an end-to-end spear phishing campaign generator for Twitter. They train both long short-term memory (LSTM) [10] models and hidden Markov models (HMMs) [4] on Twitter user posts, seeded on a topic from the user's timeline. They evaluate their approach using the click-rates of IP-tracked links. We present a more detailed framework for the generation and evaluation of spearphishing e-mail content: we compare multiple models including RNNs with LSTM units [28] and Transformer [30]; compare these models to human performance; and propose

evaluation metrics to assess the realism of the generated content along syntactic and semantic dimensions.

Spearphishing e-mail detection approaches include machine learning as well as information retrieval techniques. Dewan, et al. [6] built a random forest classifier using both stylometric and social features. Stylometric features (e.g, "number of words") were extracted from the body of an e-mail while social features (e.g., "location") were extracted from LinkedIn profiles. Kotson, et al. [14] applied *tf-idf* on the bag-of-words retrieved from the curriculum vitae of the e-mail sender and the LinkedIn profile of the e-mail recipient. They developed metrics to determine whether an e-mail is either a spearphishing or a phishing e-mail. We evaluate the performance of Dewan et al.'s [6] phishing filter at differentiating real from automatically-generated content.

Existing tools such as Social Engineering Toolkit (SET)[1] provide a menu-driven system for network administrators to set up a phishing website and an interface to compose e-mail that will be distributed to their targets. Typically, the content of such e-mail are created manually and have a broad-range of topics in order to cast a wide net (e.g., change password immediately, check your corporate bank account, etc.). Studies have shown that the click-through rate for such e-mail varies between 0% to 40% [27]. However, network administrators can leverage their access to employees' e-mail to build additional context (e.g., in terms of complete sentences from their Inbox) and compose phishing e-mail that are tailored to each employee. Therefore, the approach presented in this paper enables an administrator to use an employee's e-mail contents and automatically create phishing content at scale.

4.2 Sequence to sequence learning

Bahdanau, et al. [1] define the problem of translation in probabilistic terms as $argmax_y p(y|x)$ or the problem of finding a target sentence, y, that maximizes the conditional probability of y, given a source sentence, x. The NMT problem is that of fitting a parameterized model that maximizes the conditional probability of sentence pairs given a parallel training corpus [1]. The basic NMT approach is one of encoding an input sequence or source sentence, and then, decoding it to an output sequence or target sentence. This mapping from an input sequence to an output sequence is also known as sequence to sequence learning.

Sutskever, et al. [28] proposed an encoder-decoder framework for NMT, which used recurrent neural networks (RNNs) with long short-term memory (LSTM) units. They took advantage of the property that LSTMs learn to map a variable length input sequence to a fixed-dimensional vector representation. (Sentences that are close in meaning will tend to be close to one another in the vector space.) To meet the objective of translating, the authors state that LSTMs learn to find sentence representations that can capture meaning. They also show that their model is aware of word order, providing qualitative support for their claim. Since translating is a type of paraphrasing, we were inspired to apply sequence to sequence models for the sentence generation problem. We were also motivated to consider whether fine-grained metrics could quantify the model's performance at capturing meaning in the output sentences.

Improving on the original sequence to sequence models, Vaswani et al. [30] noted that better performance could be achieved by connecting the encoder and decoder using an attention mechanism. An attention mechanism such as self-attention relates different positions of the same sequence to compute the sequence's representation. The authors proposed a new model, called the Transformer, which was based solely on attention. By inspecting some of the outputs of attention, the authors noted behavior related to the syntactic and semantic structure of the sentences. Again, this provided motivation for us to quantify the outputs of this model in terms of both syntactic and semantic structure.

4.3 Question Generation

Question generation has been explored over the years in prior works [21, 23], primarily implementing rule-based approaches. The most relevant recent work is by Du, et al.[8] on an end-to-end sequence to sequence approach to generating questions for the task of reading comprehension. Du, et al. [8] introduced a sequence to sequence learning model, applying the attention mechanism to an RNN encoder-decoder architecture. Given a text passage as input, the authors generated a question that would test a user's understanding of that passage. They compared the effects of encoding sentence- vs. paragraph-level information for this task. Since their work had ramifications for question answering, they used the Stanford Question Answering dataset[2]. To the best of our knowledge, the work discussed in this paper is the first to explore the use of a paraphrase dataset for generating questions.

4.4 Conversational agents

A practical application of sentence generation has been the development of dialogs for conversational agents or chatbots, dating back to ELIZA [31]. State-of-the-art learning architectures for dialog systems include retrieval-based models [11, 18, 19, 29] (e.g., dual encoder) [19], which select the next best response from a list of candidates. Retrieval-based systems apply machine learning classifiers or information retrieval techniques to choose pre-defined responses from a collection of responses. Most production systems are retrieval-based. The primary limitation of this approach is that these models do not generate new text. Therefore, we do not consider it as a viable approach for generating personalized content at scale in an attack scenario.

Generative approaches, e.g., using sequence to sequence learning, generate a response given an utterance. These systems typically do not rely on pre-defined responses and are based on machine translation techniques. Primary limitations include models generating generic responses or lacking an understanding of context. They do not work very well in practice since responses suffer from mistakes, inconsistency, and irrelevancy. However, by developing tools that give us better visibility into its shortcomings, we assess whether a sequence to sequence model would be suited for the stated attack scenario.

Google's SmartReply [12] is an example of a system that combines retrieval-based and generative approaches. It is trained to predict whether someone might write a brief reply to an e-mail, and if so, generate three possible responses. One of the unique

[1] https://www.trustedsec.com/social-engineer-toolkit-set

[2] https://stanford-qa.com

features of SmartReply is that it qualifies responses as being "high quality" if they deliver a positive user experience. However, this system is mostly geared to "yes"/"no" questions, rather than the more free-form questions an attacker may pose.

4.5 Evaluation metrics

Commonly used metrics for evaluating machine translation systems include BLEU [22], NIST [7], and ROUGE [16]. Our work extends both the BLEU and ROUGE metrics with syntactic as well as semantic analyses. With regard to the former, our work is similar to the work of Ding, et al. [17], in that we incorporate syntactic information in the computation of similarity.

Ding, et al., [17] use syntax trees to find matches – they compute the fraction of the candidate subtree that also appears in the complete syntax trees of the reference sentences. We view this a top-down approach, whereas the approach we take is bottom-up. For one of our syntax-based or structural metrics, we compute similarity based only on part-of-speech information. This is the syntactic information that is closest to the leaf level or lexical information (words) of the parse trees of the candidate and reference sentences. For dependency parses, Ding, et al. [17] implement a headword chain metric using *n*-grams of dependency chains rather than *n*-grams in the linear order of the sentence, as well as a dependency subtree metric and a dependency tree kernel metric.

For our second structural metric, we focus on relationships between words. Specifically, we consider dependency information [17], but in a bottom-up manner – generalizing from the part-of-speech information, we extract noun chunks – i.e., nouns and the words (verbs and prepositions) surrounding them. We select noun chunks since they often contain a sentence's topical information such as keywords and phrases. We form triples that capture relationships – e.g., subject, object – between the (head) nouns, verbs, and prepositions and match *n*-grams of triples in the linear order of the sentence.

Our semantic similarity metric computes topical similarity or topical intersection. It was motivated by the embedding average score [19], which estimates a compositional embedding of a sentence by averaging the word vectors that constitute the sentence. As in the work of [19], we compute the cosine similarity between two sentence embedding vectors, $\mathbf{D_A}$ and $\mathbf{D_B}$, as a final score:

$$sim(\mathbf{D_A}, \mathbf{D_B}) = \mathbf{D_A} \cdot \mathbf{D_B} / |\mathbf{D_A}||\mathbf{D_B}|$$

Although averaging embedding vectors has been suggested in the literature as a way to capture the semantics of sentences, we introduce additional constraints such as removing stop words in the sentences to focus on content-bearing words and computing the topical intersection of two sentences. We compare our results on topical similarity to a benchmark that computes the distance between two probability distributions, word mover's distance [15]. Unlike our topical similarity metric, word mover's distance does not take into account word position information, and therefore, provides a complementary point of view. We do note that recent work has extended word mover's distance to account for word position [9].

5 EXPERIMENTAL SETUP

We train and evaluate three NMT models on sentence generation given a corpus of sentence pairs. We also compare these results to human paraphrasing. Finally, we test the hypothesis that the generated sentences will also pass a spam/spearphishing filter with reasonable performance.

5.1 Models

For our NMT experiments, we used OpenNMT-py, a PyTorch implementation of an open-source toolkit for NMT [13]. We chose this toolkit for a variety of reasons. First, we chose a standardized implementation of NMT models for the purposes of benchmarking our sentence generation task and building on prior results. Second, since NMT tasks could take days to weeks to train, efficiency was key, and this system was designed to prioritize efficiency during both the training and testing phases. We applied and evaluated three variants of OpenNMT's sequence to sequence models for translation. The first model is our baseline – it is a basic implementation of the sequence to sequence model with LSTMs [28], which we refer to as *lstm-1*. It implements a 2-layer LSTM with 500 hidden units on both the encoder and decoder without the attention mechanism. The second model is OpenNMT's sequence to sequence model with global attention [1], which we refer to as *lstm-2*. Our third model is an implementation of Vaswani et al.'s [30] attention-only Transformer network.

5.2 Dataset

We chose to develop and test our sentence generation approach on the Quora question pairs dataset, which is available from Kaggle[3]. Quora is a knowledge-sharing platform where individuals congregate to ask and provide answers to questions. We think of this as present-day version of a newsgroup platform. Given that Quora has 100 million users, it is natural that some of the questions will have similarly worded or "duplicate" content. Quora uses a random forest classifier to label these pairs of duplicates. The dataset has more than 404K question pairs of which more than 149K have been labeled as duplicates. We refer to the latter as the paraphrase dataset.

Since the text of the source and target sentences in a sentence pair can contain multiple questions, as a pre-processing step, we scan the text from the start until we reach the end of the first complete question. We try to achieve a one-to-one mapping from a question to its paraphrase in each pair. This also makes training more efficient.

To ensure that we use as much of the data as possible for training, we set aside 10% of the paraphrase dataset for testing. Of the remaining 90% of the data, 10% is used for validation. In our experiments, we also use traditional natural language pre-processing steps on the data such as lower-casing, word tokenization, and punctuation removal. We did not perform any pre-processing to check the grammar of the source or target sentences.

[3]https://www.kaggle.com/quora/question-pairs-dataset

Table 1: Sentences generated by the Transformer model.

Source sentence	Generated (candidate) sentence
(1) how does whatsapp makes money	(1) how does whatsapp make money
(2) how does eukaryotic and prokaryotic cells differ	(2) how do eukaryotic and prokaryotic cells differ
(3) which test ielts or pte is easy	(3) which test would be easier ielts or pte pearson test of english
(4) what is the best twitter management program	(4) what is the best software to manage your twitter account

5.3 Experiments

We generate questions using three NMT models trained on the Quora paraphrases: (1) a basic lstm-based sequence to sequence model without attention, (2) an lstm-based sequence to sequence model with attention, and (3) the Transformer network. We also compare performance with human paraphrasing. For our final experiment, we implement state-of-the-art spam/spearphishing filters. We then test how well these filters differentiate between the original, human-generated questions and questions automatically-generated by the sequence to sequence models.

We performed the above experiments on a single GPU. We ran all of our models for 50K iterations. We evaluate the behavior of the models using both traditional metrics as well as new metrics that we describe in this paper. As Lowe, et al. [19] note, to preserve the statistical independence between the task and performance metrics, word embeddings should be trained on a corpus that is different from the task corpus. For our metrics, we trained our word embeddings on Google News data[4].

5.4 Examples of generated questions

In Table 1, we present some example sentences generated by the Transformer network. The quality of the generated sentences depends on the paraphrases used to train the model. The first two sentences are examples where the model generates sentences that are grammatically correct whereas the source sentences have errors in subject-verb agreement. This is a by-product of our training data – there are also cases where the model learns ungrammatical sentences when the errors appear in the target sentences. The third sentence is an example where the source sentence is more fluent than the candidate sentence generated by the model. Both the third and fourth sentence introduce new terms, which elaborate on the original content of the source sentences, contributing to output variety.

6 EVALUATION

6.1 Existing metrics and their limitations

Bilingual Evaluation Understudy Score (BLEU) [22] is the most widely used metric for evaluating dialogue systems; it is based on word overlap. It analyzes co-occurrences of n-grams in the candidate sentence vs. some reference sentence(s). N represents the order of the n-gram. For example, the sentence, "The cat ran", has three n-grams of order 1 or unigrams, "the", "cat", and "ran". It has two n-grams of order 2 or bigrams, "the cat" and "cat ran", and one n-gram of order 3 or trigram, "the cat ran". In practice, we compute BLEU scores by taking a weighted average of truncated (clipped) n-gram precision for n=1, 2, 3, and 4. For our dataset, we compute corpus-level BLEU, a modified n-gram precision based on clipped n-gram counts per candidate sentence (numerator) and n-gram' counts per corpus (denominator):

$$\frac{\sum_{C \in Candidates} \sum_{n\text{-}gram \in C} Count_{clip}(n\text{-}gram)}{\sum_{C' \in Candidates} \sum_{n\text{-}gram' \in C'} Count(n\text{-}gram')}$$

The final score also incorporates a multiplicative factor or penalty for short translations. As Papineni, et al., [22] state, a good candidate sentence must match the reference sentences in word choice, word order, and in length. To provide some context to our results, we refer to an interpretation of BLEU scores in Table 2[5].

It is common practice to report all BLEU scores (for n=1, 2, 3, and 4) for an evaluation. By varying n, we may get very different BLEU scores. Given a candidate sentence, "The hat in the cat is brown", and the reference sentence, "The cat in the hat is brown", the BLEU-1 score is 1, in spite of the surface dissimilarity between the sentences. Even though a BLEU score is calculated as a weighted geometric mean, since there are no 4-gram overlaps between the two sentences, BLEU-4 defaults to 0.

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [16] computes recall or how much of the reference text is captured by the candidate sentence. For our dataset, we compute the average recall score for words, w, across all candidate and reference sentences:

$$\frac{|\ \{w \mid w \in Reference \cap w \in Candidate\}\ |}{|\ \{w \mid w \in Reference\}\ |}$$

Since we strive for variety in the sentences generated rather than strict duplication, we need both measures of recall and precision for a balanced evaluation. However, these metrics are based on strict word overlap and are not meant to generalize. We need to generalize from words to higher-order structures to assess grammatical well-formedness. We also need to handle sentences that have different words but are similar in meaning.

6.2 Structural similarity metrics

We develop new metrics that extend BLEU and ROUGE by giving us greater visibility into the grammatical structure of the generated text. We introduce a new metric, Part-of-speech-BLEU (POS-BLEU), which extends the BLEU metric to account for grammatical structure more precisely. To compute POS-BLEU, we replace each word

[4]https://code.google.com/archive/p/word2vec

[5]https://cloud.google.com/translate/automl/docs/evaluate

Table 2: Understanding BLEU scores(courtesy Google).

BLEU Score	Interpretation
<10	Almost useless
10-19	Hard to get the gist
20-29	The gist is clear, but has significant grammatical errors
30-40	Understandable to good translation
40-50	High quality translations
50-60	Very high quality, adequate, and fluent translations
>60	Quality often better than human

token in a sentence with its part-of-speech tag - e.g., "noun", "verb", etc. First, this metric has the effect of generalizing both the reference text and the candidate response generated by the model. For example, two words in a reference sentence and candidate response may be dissimilar on the surface. However, if they are both examples of the same part-of-speech, our metric will consider this a match. Second, this metric helps to characterize the behavior of our models in generating grammatical structures. By considering n-grams (where n is in the range of 1 to 4) of POS tags, we focus, primarily, on syntactic co-occurrences – e.g., the collocation of determiners and nouns in a sentence. In this sense, we consider POS-BLEU as a smoothing technique that helps us determine underlying co-occurrences in text. As a smoothing technique, we expect POS-BLEU scores to be higher than BLEU scores.

With POS-BLEU, we are less concerned with capturing long-range dependencies – i.e., relationships between words that may be spaced farther apart in a sentence. For example, we expect every sentence to have a subject and verb, but the subject and verb may be spaced far apart in a given sentence. To capture long-range dependencies or syntactic relationships between keywords in a sentence, we introduce the Relation-BLEU (REL-BLEU) metric. To compute REL-BLEU, we first parse a sentence to identify noun chunks. Then, we iterate through each noun chunk, forming triples that capture relationships between nouns, verbs, and prepositions – e.g., some sample relation triples are:

(1) <subject, noun, verb>
(2) <direct-object, noun, verb>
(3) <preposition-object, preposition, noun>

The REL-BLEU metric introduces another level of generalization, and therefore, is a smoothing technique that helps us determine trends or dependencies in sentence structure. For both REL-BLEU and POS-BLEU, we build on the underlying modified n-gram precision computation of BLEU. Applying the same smoothing techniques in the computation of recall, we introduce POS-ROUGE and REL-ROUGE. These metrics also substitute POS tags and relation triples for word tokens, respectively.

6.3 Semantic similarity metrics

Semantic similarity is a broad term that describes overlap in semantic information across data items. Various measures of semantic similarity have been developed in the natural language processing and information retrieval communities. However, a single metric is not likely to address all aspects of semantic overlap. For example, if we compute the cosine similarity of the word embedding vectors for the two sentences, "The cat in the hat is brown" and "The hat in the cat is brown", we will get a similarity of 1 even though the sentences have very different meanings.

We assume the reference sentence is a representation of the interests of the individual. Using an NMT model, we generate a candidate sentence, and we would like to determine how closely we have matched the topics in the reference sentence. To do this, we ask a specific question about semantic overlap – what is the intersection of content-bearing terms in the candidate and reference sentences? We develop a new vector-based generative metric, topical intersection, for measuring semantic similarity [19] across candidate and reference sentences. This metric averages word embedding vectors at different levels of granularity (n-grams) for both the reference text and the candidate response and computes cosine similarity to measure relatedness between the two.

Topical intersection computes the average pairwise cosine similarity across average embedding vectors of n-grams (for n=1, 2) in both the candidate and reference sentences. First, we remove stop words from the sentences. We note that removal of stop words is a step towards standardizing sentences to sequences of content-bearing words such as nouns, verbs, adjectives, and adverbs. Second, cosine similarity comparisons are computed across n-grams, enabling us to preserve word order across the comparisons.

Documents such as technical papers have been compared by matching their titles, keywords and abstracts [3]. Here, our semantic unit of comparison is a sentence or a set of keywords – nevertheless, we break this unit into constituent n-grams for pairwise comparisons. Finally, since our goal is to assess common semantic information or the intersection of content-bearing terms between two sentences, pairwise comparisons are constrained by the length of the shorter, standardized sentence.

For our second metric, we implement the word mover's distance [15] between two sentences. Word mover's distance, a derivative of earth mover's distance, predicates the minimum distance between two documents on the basic unit of the "travel cost" between two words. The assumption is that a word (or its part) can be transformed from its representation in document A to a representation in document B. For example, consider moving words from the candidate to reference sentence. To transform all words from A to B, we induce an outgoing flow from A. This flow must match the incoming flow to B. From these constraints, the word mover's distance

between two documents is defined as the minimum weighted cost to move all words from *A* to *B*.

7 RESULTS AND DISCUSSION

The results of our first experiment are shown in Figure 1. We compare the baseline model *lstm-1* after 50K iterations with *lstm-2* after 50K iterations. The literature shows that models with attention outperform models without attention. As expected, we see the performance gap in BLEU scores, achieving better performance with *lstm-2*, a model implemented with attention. Comparing the two attention-based models, we show *lstm-2* after 50K iterations and the Transformer model after only 5K iterations – we find that *transformer* achieves BLEU scores that are at par with *lstm-2* at 1/10th the number of iterations.

We see the smoothing effects of the POS-BLEU metric in Figure 2. We see a significant increase in scores compared to BLEU1-4 in Figure 1 across all the models, suggesting a high degree of syntactic co-occurrence in the generated text compared to the reference text. Similarly, we see an increase in REL-BLEU scores compared to BLEU, in Figure 3. However, the differences are not as marked, suggesting that the generated sentences are more diverse in their longer-term syntactic dependencies when compared to the reference text.

In Figure 3, we note that all three of the models are comparable in their performance at capturing longer-term dependencies. Even though *lstm-1* does poorer than the other models in terms of word overlap (BLEU) and syntactic co-occurrences (POS-BLEU), it does pretty well in learning the overall structure of sentences compared to the other models.

In Figure 4, we see that when measuring average recall using the ROUGE metric, with our smoothing techniques, we are able to measure scores above 75% for POS-ROUGE and for REL-ROUGE, for all of our models. For example, this suggests that the structure of the generated sentences have a high degree of similarity with the short-range and long-range structure of the reference sentences.

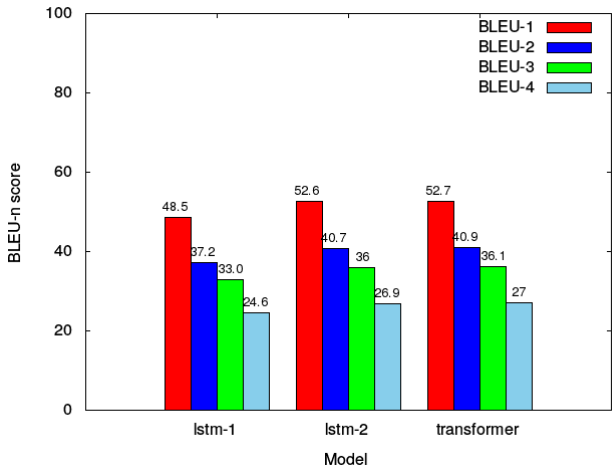

Figure 1: Comparison of two LSTM and the Transformer models using BLEU.

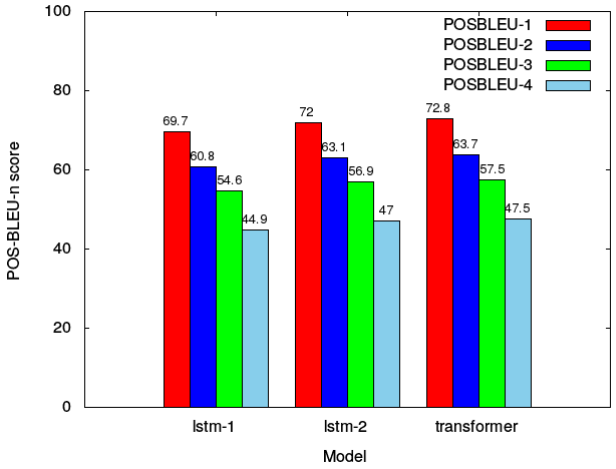

Figure 2: Comparison of two LSTM and the Transformer models using POS-BLEU.

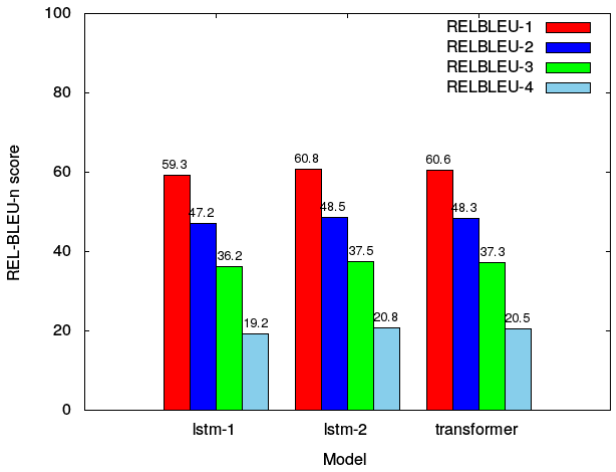

Figure 3: Comparison of two LSTM and the Transformer models using REL-BLEU.

We also note that all of the metrics discussed so far produced very consistent results for *lstm-2* and *transformer*. Although this does not give us additional information to differentiate between them, by analyzing grammatical structure independently of the BLEU metric, we have an additional tool to judge the grammatical realism of the generated sentences.

By developing a fine-grained metric for assessing semantic similarity of sentences, in Table 3, we see that all the models perform comparably when assessing the intersection of content-bearing terms (unigram topical intersection). In other words, they are all able to model the key terms, and hence, topics, well. We also notice

Table 3: Comparison of semantic similarity and distance metrics for two LSTM and the Transformer models.

Model	Unigram Topical Intersection	Bigram Topical Intersection	Word Mover's Distance
lstm-1	0.9	0.6	0.63
lstm-2	0.9	0.6	0.56
transformer	0.91	0.63	0.57

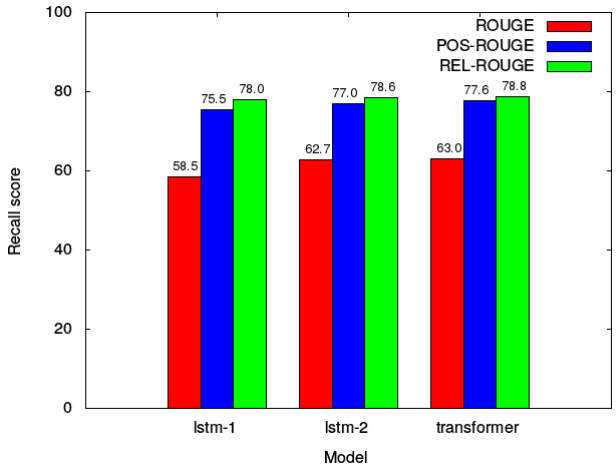

Figure 4: Comparison of two LSTM and the Transformer models using ROUGE, POS-ROUGE, and REL-ROUGE.

differences between *transformer* and the other models – we observe that *transformer* has a slight advantage over the other models, specifically when computing bigram topical intersection. This suggests that *transformer* attains a better intersection of key phrases. This is also an example where the traditional BLEU and ROUGE metrics do not provide visibility into measuring this difference in the behavior of these two models.

Examining the results for word mover's distance, we find that both *lstm-2* and *transformer* are competitive in their scores (note that since we are measuring distance, lower scores indicate greater similarity). We also note that this metric does not account for word order, so it provides a different perspective on the semantic similarity of the results.

7.1 A comparison to human paraphrasing

Ideally, we would compare our findings with how well humans paraphrase. Since both source and target Quora sentences were generated by humans, independently, and since they were, consequently, labeled as paraphrases, we estimate human paraphrasing by considering source test sentences as reference text and target test sentences as candidate text. Results of the behaviors of the metrics are shown in Figures 5 through 8 and in Table 4.

We make a few observations about structural similarity. First, the results shown in Figures 5 through 7 provide an upper bound based on our estimate of human performance for this dataset – all of the automated methods have yet to reach the limits for BLEU, POS-BLEU, and REL-BLEU for human-generated paraphrases. Therefore, the values of these scores helps us differentiate between human-generated and automatically-generated text.

In the same figures, we compare the results, side-by-side, for both *lstm-2* and *transformer* on Quora along with the results of human paraphrases as we vary the granularity n-gram value) of the analysis. For each model, we plot as its behavior curve, the n-gram value, N, on the x-axis and the resulting metric value on the y-axis (e.g., BLEU score), across all structural similarity metrics.

First, we reiterate the consistency in the behavior curves of *lstm-2* and *transformer* – they are barely distinguishable from each other. We see the human paraphrasing behavior curves are inconsistent in this regard. While we note that the human behavior curves dominate the rest, as we would expect, we also see in Figures 5 and 6 that the human behavior curves have steeper gradients, along multiple points on the curves, unlike the automated methods. It is possible that there is greater regularity in the mistakes made by the models compared to humans, which could be used for defense purposes – we leave this as an area for future research.

The ROUGE scores shown in Figure 8 provide additional differentiating characteristics. We note that the range of the ROUGE scores is much smaller than that observed for the automatically-generated text as shown in Figure 4. We also note that the POS-ROUGE and REL-ROUGE scores converge. These characteristics of text as measured by the ROUGE metrics help us differentiate between human-generated and automatically-generated text for this task.

For semantic similarity, we find that automatically-generated sentences do reasonably well at representing key terms (unigram topical intersection), followed by key phrases (bigram topical intersection), when compared to their human-generated counterparts as shown in Table 4. In fact, for the intersection of key terms, which is a constrained metric, we note that our models have converged with human performance.

However, we note the difference in semantic distance as measured by word mover's distance highlights the gap in performance between human and automatically-generated text. Word mover's distance for the human-generated sentences is 0.49 compared to the smallest distance for an automated method (0.56 for *lstm-2*).

An interesting question for future research is whether metrics can capture if people are able to distinguish between human-generated and automatically-generated text. For example, people write text that is not grammatically correct and has misspellings. Yet, readers routinely compensate for errors and omissions in text without attributing them to automated methods. When it comes to semantics, we do not expect contiguous text in an e-mail to drift from topic to topic, so if automatically-generated text exhibited this property, it may be easier to detect.

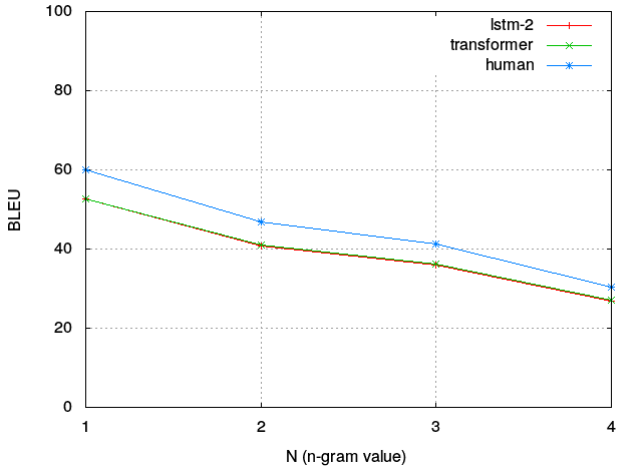

Figure 5: Comparison of BLEU-n scores across lstm-2, transformer, and human paraphrases.

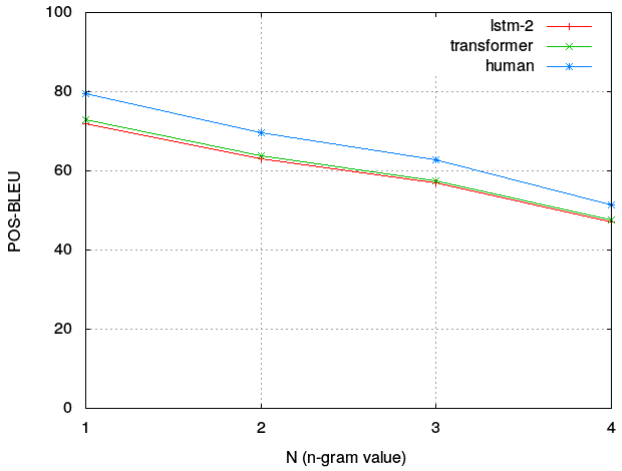

Figure 6: Comparison of POS-BLEU-n scores across lstm-2, transformer, and human paraphrases.

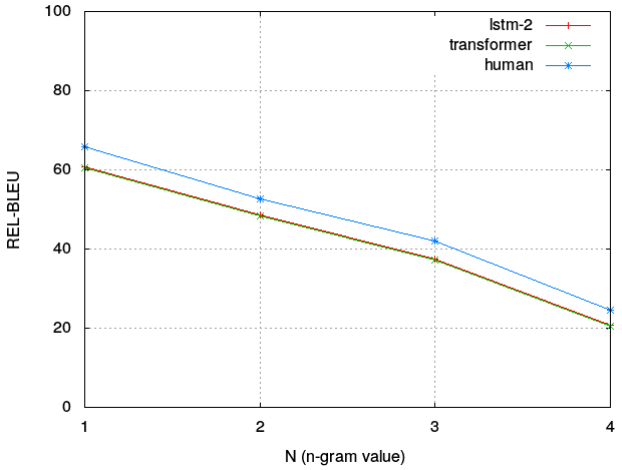

Figure 7: Comparison of REL-BLEU-n scores across lstm-2, transformer, and human paraphrases.

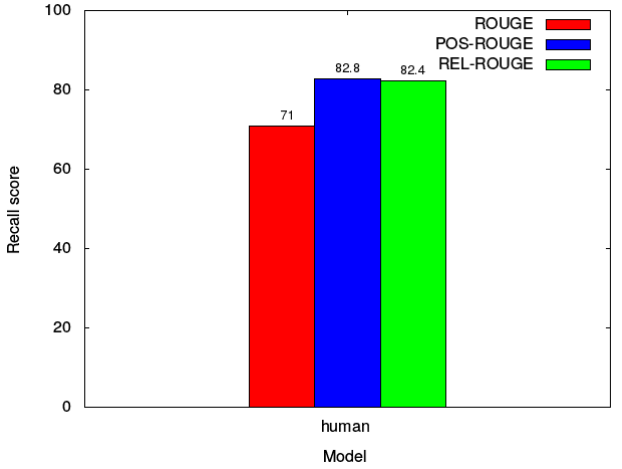

Figure 8: Recall (ROUGE, POS-ROUGE, and REL-ROUGE) scores for human paraphrases.

7.2 Spam/spearphishing filtering results

We test the quality of our automatically-generated content using *transformer* by running the content through spam/spearphishing filters. Our primary finding is that using these filters, it is difficult to differentiate between human-generated and automatically-generated content, which increases the threat posed by the kind of attack described in this paper.

In our first experiment, we implemented a random forest classifier for a three-class problem: classifying e-mail either as (1) ham (real), (2) phishing, or (3) spam. For the ham and spam e-mail, we train and test the classifier using Enron e-mail data, and for the phishing e-mail, we used a dataset from Kaggle[6]. As in [6], we train our classifier using 6 features: number of words, number of characters, number of newlines, number of unique words, richness (number of words/number of characters) and number of function words present.

Our dataset had 900 data points (300 per class). We split the data, using 75% for training and 25% for testing. We show the performance of the random forest classifier on the test set in Table 5. Using this classifier, we predicted the category of our questions. More than 80% of the human-generated questions were classified

[6]https://www.kaggle.com/rtatman/fraudulent-email-corpus

Table 4: Comparing human performance with automated models using semantic similarity and distance metrics.

Model	Unigram Topical Intersection	Bigram Topical Intersection	Word Mover's Distance
lstm-1	0.9	0.6	0.63
lstm-2	0.9	0.6	0.56
transformer	0.91	0.63	0.57
human	0.91	0.65	0.49

Table 5: Performance of random forest classifier trained on e-mail.

Class	Precision	Recall	F1 Score	Support
ham	0.55	0.57	0.56	83
phishing	0.80	0.88	0.83	72
spam	0.49	0.43	0.46	70
Average/total	0.61	0.62	0.62	225

Table 6: Performance of random forest classifier trained on automatically-generated sentences.

Class	Precision	Recall	F1 Score	Support
ham	0.88	0.90	0.89	78
phishing	0.98	0.96	0.97	68
spam	0.86	0.86	0.86	79
Average/total	0.90	0.90	0.90	225

as ham and the remaining as spam (i.e., 0% were classified as phishing). The results were also true for the *transformer*-generated questions. This implies that the characteristics of human-generated and automatically-generated text were similar to legitimate content.

In the second experiment, we replaced ham e-mail with human-generated questions and trained another random forest classifier using the same train/test split. The performance of the random forest classifier on the test set is shown in Table 6. Using this classifier, we predicted the category of the automatically-generated questions and observed that 87.7% of them were classified as ham, 12% were classified as spam and 0.3% were classified as phishing. Note that since automatically-generated questions are intended to serve as phishing content, these results imply that the phishing filter had difficulty in distinguishing between human-generated and automatically-generated questions.

8 CONCLUSIONS

In conclusion, in this paper, we have shown how state-of-the-art NMT models have made great strides in generating paraphrases, while not yet reaching levels of human performance on this task. We describe how we characterize the text generated by these models for realism, specifically along the dimensions of structural and semantic similarity of text. We find the text that is automatically-generated by top-performing NMT models show consistent behaviors across a number of metrics but differ in these behaviors when compared to human paraphrases. These models also do not converge on some values of recall when compared to human paraphrasing. Finally, we have shown that current state-of-the-art spam and spearphishing filtering techniques have a difficult job differentiating between text that is generated by humans versus NMT approaches. While we have shown that NMT approaches are viable for the generation of spearphishing content at scale, our metrics for characterizing the realism of text also provide an initial direction for defending against such attacks.

ACKNOWLEDGMENTS

This research was sponsored 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.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[2] Alejandro Correa Bahnsen, Ivan Torroledo, Luis David Camacho, and Sergio Villegas. 2018. DeepPhish: Simulating Malicious AI. In *Proceedings of the Symposium on Electronic Crime Research, San Diego, CA, USA*. 15–17.

[3] Chumki Basu, Haym Hirsh, William W Cohen, and Craig Nevill-Manning. 2001. Technical paper recommendation: A study in combining multiple information sources. *Journal of Artificial Intelligence Research* 14 (2001), 231–252.

[4] Leonard E Baum and Ted Petrie. 1966. Statistical inference for probabilistic functions of finite state Markov chains. *The annals of mathematical statistics* 37, 6 (1966), 1554–1563.

[5] Joseph Bullock and Miguel Luengo-Oroz. 2019. Automated Speech Generation from UN General Assembly Statements: Mapping Risks in AI Generated Texts. *arXiv preprint arXiv:1906.01946* (2019).

[6] Prateek Dewan, Anand Kashyap, and Ponnurangam Kumaraguru. 2014. Analyzing social and stylometric features to identify spear phishing emails. In *2014 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 1–13.

[7] George Doddington. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*. Morgan Kaufmann Publishers Inc., 138–145.

[8] Xinya Du, Junru Shao, and Claire Cardie. 2017. Learning to ask: Neural question generation for reading comprehension. *arXiv preprint arXiv:1705.00106* (2017).

[9] Hiroshi Echizen-ya, Kenji Araki, and Eduard Hovy. 2019. Word Embedding-Based Automatic MT Evaluation Metric using Word Position Information. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 1874–1883.

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

[11] Michimasa Inaba and Kenichi Takahashi. 2016. Neural utterance ranking model for conversational dialogue systems. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. 393–403.

[12] Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, Laszlo Lukacs, Marina Ganea, Peter Young, et al. 2016. Smart reply: Automated response suggestion for email. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 955–964.

[13] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810* (2017).

[14] Michael C Kotson and Alexia Schulz. 2015. Characterizing phishing threats with natural language processing. In *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 308–316.

[15] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. 2015. From word embeddings to document distances. In *International conference on machine learning*. 957–966.

[16] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[17] Ding Liu and Daniel Gildea. 2005. Syntactic features for evaluation of machine translation. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. 25–32.

[18] Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. 2015. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *arXiv preprint arXiv:1506.08909* (2015).

[19] Ryan Thomas Lowe, Nissan Pow, Iulian Vlad Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. 2017. Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue & Discourse* 8, 1 (2017), 31–65.

[20] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).

[21] Jack Mostow and Wei Chen. 2009. Generating Instruction Automatically for the Reading Strategy of Self-Questioning.. In *AIED*. 465–472.

[22] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 311–318.

[23] Vasile Rus, Brendan Wyse, Paul Piwek, Mihai Lintean, Svetlana Stoyanchev, and Cristian Moldovan. 2010. The first question generation shared task evaluation challenge. (2010).

[24] Gerard Salton and Michael J McGill. 1983. *Introduction to modern information retrieval*. mcgraw-hill.

[25] John Seymour and Philip Tully. 2016. Weaponizing data science for social engineering: Automated E2E spear phishing on Twitter. *Black Hat USA* 37 (2016).

[26] John Seymour and Philip Tully. 2018. Generative Models for Spear Phishing Posts on Social Media. *arXiv preprint arXiv:1802.05196* (2018).

[27] Hossein Siadati, Sean Palka, Avi Siegel, and Damon McCoy. 2017. Measuring the effectiveness of embedded phishing exercises. In *10th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET} 17)*.

[28] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[29] David Traum, Kallirroi Georgila, Ron Artstein, and Anton Leuski. 2015. Evaluating spoken dialogue processing for time-offset interaction. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. 199–208.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[31] Joseph Weizenbaum et al. 1966. ELIZA—a computer program for the study of natural language communication between man and machine. *Commun. ACM* 9, 1 (1966), 36–45.

[32] Ze Yang, Can Xu, Wei Wu, and Zhoujun Li. 2019. Read, Attend and Comment: A Deep Architecture for Automatic News Comment Generation. *arXiv preprint arXiv:1909.11974* (2019).