

ANOMALY DETECTION AND SECURITY DEEP LEARNING METHODS  
UNDER ADVERSARIAL SITUATION

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Miguel A. Villarreal-Vasquez

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2020

Purdue University

West Lafayette, Indiana

**THE PURDUE UNIVERSITY GRADUATE SCHOOL**  
**STATEMENT OF DISSERTATION APPROVAL**

Dr. Bharat Bhargava, Chair

Department of Computer Science

Dr. Vernon Rego

Department of Computer Science

Dr. Xiangyu Zhang

Department of Computer Science

Dr. Zhiyuan Li

Department of Computer Science

Dr. Christopher Brinton

Department of Electrical and Computer Engineering

**Approved by:**

Dr. Clifton W. Bingham

Department of Computer Science

To my family, friends and beloved people of Monagrillo (Tierra Mía).

## ACKNOWLEDGMENTS

My special gratitude to my advisor Professor Bharat Bhargava for his unconditional support and encouragement. During my studies, Professor Bhargava was always there to provide me with advice in the search for my dissertation topic. Of all the good things for which I am grateful to him, I truly want to express my gratitude for having given me the time to explore, think, choose, and solve real-world cybersecurity problems as part of my dissertation. To me, the Ph.D. journey requires a lot more of this time-to-think than anything else. After all, we all join the great Department of Computer Science at Purdue to learn and then contribute to the scientific community with the new acquired knowledge. For that, many thanks Professor!

I am grateful to my advisory committee Professors Vernon Rego, Xiangyu Zhang, and Zhiyuan Li for their time and attention while completing my Ph.D. degree. I really appreciate you taking the time from your busy schedule to consider this work. I also thank Professor Christopher Brinton for serving in my examining committee.

I cannot leave West Lafayette without thanking Mrs. Shail Bhargava for her hospitality and great Indian food. Enjoying your good food while I was full on work was definitely an injection of positive energy. Your plates are delicious, Shail!

I am deeply thankful to my Panamanian friend Gaspar Model-Howard (el pasiero Gaspar) for his friendship and mentorship in both cybersecurity and life. One more Ph.D. for Panama my friend, which I am sure makes you very happy.

To my family, good friends, and significant others my deepest gratitude for always being there for me. Without your support, time, and help this life-goal of myself called Ph.D. would not have been so enjoyable. The findings of my research in cybersecurity included in this dissertation are dedicated to all of you.

Lastly, I want to thank the Department of Computer Science at Purdue University and Northrop Grumman Corporation (NGC) for funding my Ph.D. program.

## TABLE OF CONTENTS

	Page
LIST OF TABLES . . . . .	viii
LIST OF FIGURES . . . . .	ix
ABSTRACT . . . . .	xi
1 INTRODUCTION . . . . .	1
1.1 Motivation . . . . .	1
1.2 Dissertation Organization . . . . .	3
1.3 Dissertation Contribution . . . . .	5
1.4 Published Work . . . . .	6
2 CONFOC: CONTENT-FOCUS PROTECTION AGAINST TROJAN AT- TACKS ON NEURAL NETWORKS . . . . .	7
2.1 Introduction . . . . .	7
2.2 Threat Model and Overview . . . . .	11
2.2.1 Threat Model . . . . .	11
2.2.2 Overview . . . . .	12
2.3 Background and Related Work . . . . .	13
2.3.1 Deep Neural Networks . . . . .	13
2.3.2 Trojan Attacks: Assumptions, Definition and Scope . . . . .	14
2.3.3 Prior Work on Trojan Attacks . . . . .	17
2.3.4 Existing Defensive Techniques Against Trojan Attacks . . . . .	18
2.4 Content-Focus Approach . . . . .	21
2.4.1 Feature Extraction . . . . .	23
2.4.2 Content Image Generation . . . . .	24
2.4.3 Style Image Generation . . . . .	25
2.4.4 Styled Image Generation . . . . .	27

	Page
2.5	Evaluation Setup . . . . . 27
2.5.1	Evaluation Metrics and Testing Sets . . . . . 29
2.5.2	BadNets Attack . . . . . 30
2.5.3	Trojaning Attacks: Square (SQ) and Watermark (WM) . . . . . 32
2.5.4	Acronyms Used in Experiments . . . . . 34
2.6	Experiments . . . . . 34
2.6.1	Robustness When Processing Original Inputs . . . . . 34
2.6.2	Robustness When Processing Transformed Inputs . . . . . 37
2.6.3	Effect on Non-Trojaned Models . . . . . 39
2.6.4	Healing Set Size and Number of Styles . . . . . 41
2.6.5	Performance and Comparison With the State-of-the-Art . . . . . 42
2.6.6	Robustness Against Adaptive and Complex Triggers . . . . . 44
2.7	Conclusions and Future Work . . . . . 47
3	HUNTING FOR INSIDER THREATS USING LSTM-BASED ANOMALY DETECTION . . . . . 49
3.1	Introduction . . . . . 49
3.2	Overview and Threat Model . . . . . 52
3.2.1	Overview . . . . . 52
3.2.2	Threat Model . . . . . 54
3.3	Background and Related Work . . . . . 54
3.3.1	LSTM Networks . . . . . 54
3.3.2	Order-Aware Recognition (OAR) Problem . . . . . 56
3.3.3	Endpoint Detection and Response . . . . . 57
3.3.4	Anomaly Detection Based on Sequence Analysis Using Non- LSTM approaches . . . . . 57
3.3.5	Anomaly Detection Based on Sequence Analysis Using LSTM . . . . . 59
3.4	Design . . . . . 62
3.4.1	Data Generation . . . . . 63
3.4.2	Data Selection . . . . . 64

	Page
3.4.3 Model Generation . . . . .	67
3.4.4 Anomaly Detector . . . . .	67
3.5 Evaluation Setup . . . . .	68
3.5.1 Dataset . . . . .	68
3.5.2 Metrics and Ground Truth . . . . .	72
3.6 Experiments . . . . .	73
3.6.1 Dynamic vs. Static Selection of the Set of Probable Events K . . . . .	73
3.6.2 Comparison With an Enterprise Endpoint Detection and Response (EDR) . . . . .	75
3.6.3 Performance of LADOHD When Processing Sequences Without Unseen Events . . . . .	77
3.6.4 Effect of Long-Term Dependencies in the Detection of Anomalies . . . . .	78
3.6.5 Prediction Capacity of LSTM and HMM Based models Over Variable-Length Sequences . . . . .	80
3.7 Conclusion . . . . .	81
4 AN MTD-BASED SELF-ADAPTIVE RESILIENCE APPROACH FOR CLOUD SYSTEMS . . . . .	83
4.1 Introduction . . . . .	83
4.2 Related Work . . . . .	84
4.3 Proposed Approach . . . . .	85
4.3.1 Live Monitoring . . . . .	87
4.3.2 Moving Target Defense . . . . .	89
4.4 Experiments . . . . .	90
4.5 Conclusion . . . . .	92
5 FUTURE WORK . . . . .	93
5.1 Protecting Neural Networks Against Adversarial Attacks . . . . .	93
5.2 Neural Networks on Anomaly Detection . . . . .	94
REFERENCES . . . . .	96
VITA . . . . .	105

## LIST OF TABLES

Table	Page
2.1 Comparison of the Properties of Model Hardening Techniques Against Trojan Attacks . . . . .	19
2.2 Summary of Evaluated Trojan Attacks . . . . .	29
2.3 Explanation of Acronyms Used in Experiments . . . . .	33
2.4 Best Healed Models After Applying ConFoc . . . . .	41
2.5 Initial Metrics of Trojanned Models . . . . .	43
2.6 Metrics of Corrected Models After Applying ConFoc and Other State-of-the-Art Model Hardening Techniques . . . . .	45
2.7 Performance With Adaptive/Complex Triggers . . . . .	47
3.1 Comparison With Existing LSTM-based Security Solutions . . . . .	60
3.2 Description of the Different Types of Events . . . . .	63
3.3 Examples of Activity Events With Different Granularities . . . . .	65
3.4 Dataset Description . . . . .	70
3.5 Metric Values Obtained With the Original and Clean Testing Sequences . . . . .	77
3.6 Effect of Long-Term Dependencies on the Detection of Anomalies . . . . .	79
4.1 Reincarnation Process Times . . . . .	91



## LIST OF FIGURES

Figure	Page
2.1 <i>ConFoc</i> Overview. Figure 2.1a shows the flow of the classification of benign and adversarial samples with solid and dashed arrows respectively. Image regions on which the classifiers focus on to make the decision are surrounded in black (right side). Humans classify both samples correctly because they focus on content only. Figure 2.1b shows the style transfer strategy used by <i>ConFoc</i> . The content $x_c$ from input $x$ is combined with the style $b_s$ from the style base image $b$ to form the styled image $x_{b_s}$ . . . . .	8
2.2 An illustration of a trojan attack conducted by poisoning the training data (see details in Section 2.3.3). . . . .	15
2.3 A demonstration of our <i>ConFoc</i> healing process and its use for a secure classification (see details in Section 2.4). . . . .	22
2.4 Split of the base set (90% of the GTSRB dataset). <i>adversarial trj</i> comprises samples in <i>trj</i> with triggers inserted. . . . .	31
2.5 Metric variations as <i>ConFoc</i> is progressively applied to $M_T$ by increasing the number styles used in the healing process. Resulting healed models (in $x$ -axis) are evaluated with the original (non-transformed) testing datasets (refer to Table 2.3). . . . .	35
2.6 Efficiency of <i>ConFoc</i> on making models focus on content at testing. $M_T$ is healed with an incremental number of styles. Resulting healed models are evaluated with test sets generated with different style bases: $b^1 \in B$ , $a^1 \in A$ , and $a^2 \in A$ . . . . .	38
2.7 Accuracy (benign data) variation of the GTSRB-based non-trojaned model when <i>ConFoc</i> is progressively applied. . . . .	40
3.1 Architecture of a LSTM cell. Figure 3.1a shows the internal connections of the different components and the output of the internal operations of the cell. Figure 3.1b shows the details of these operations performed over the current input and previous hidden state to compute the current output. . . . .	55
3.2 Components of our anomaly detection framework LADOHD to counter insider threats: (1) data generation, (2) data selection, (3) model generation, and (4) anomaly detector. Below each component, there is a reference to the section providing a detailed explanation about its operation. . . . .	62

Figure	Page
3.3 Data collection dates from 30 machines, from April 27th to July 7th of 2018. This sparse collection timeframe was intended to capture the behavior of the application Powershell in non-attack conditions. . . . .	71
3.4 Selection of the set $K$ through both the dynamic and static approaches. Figure 3.4a shows the variation of the $TPR$ and $FPR$ with respect to different setting for $K$ . Figure 3.4b presents the distribution of the dynamic sizes of $K$ throughout the testing sequence. Notice its high variability. . . . .	74
3.5 Malicious activities reported by the Red Team. These are the activities that could be matched with events in the testing sequence. . . . .	76
3.6 Effect of long-term dependencies of LSTM models in the detection of anomalies. The example is based on the event $e_{145}$ in Table 3.6 . . . . .	78
3.7 Prediction accuracy of our LSTM and the HMM models with sequences of incremental lengths. Figure 3.7a and Figure 3.7a show the accuracy variation with subsequences extracted from the D1 training and testing sequences respectively. . . . .	82
4.1 High-level view of resiliency framework . . . . .	86
4.2 Experiment setup . . . . .	91
5.1 Possible extension of <i>ConFoc</i> to detect adversarial inputs of both adversarial sample and trojan attacks. . . . .	94

## ABSTRACT

Villarreal-Vasquez, Miguel A. PhD, Purdue University, August 2020. Anomaly Detection and Security Deep Learning Methods Under Adversarial Situation . Major Professor: Bharat Bhargava.

Advances in Artificial Intelligence (AI), or more precisely on Neural Networks (NNs), and fast processing technologies (e.g. Graphic Processing Units or GPUs) in recent years have positioned NNs as one of the main machine learning algorithms used to solve a diversity of problems in both academia and the industry. While they have been proved to be effective in solving many tasks, the lack of security guarantees and understanding of their internal processing disrupts their wide adoption in general and cybersecurity-related applications. In this dissertation, we present the findings of a comprehensive study aimed to enable the absorption of state-of-the-art NN algorithms in the development of enterprise solutions. Specifically, this dissertation focuses on (1) the development of defensive mechanisms to protect NNs against adversarial attacks and (2) application of NN models for anomaly detection in enterprise networks.

In this state of affairs, this work makes the following contributions. First, we performed a thorough study of the different adversarial attacks against NNs. We concentrate on the attacks referred to as trojan attacks and introduce a novel model hardening method that removes any trojan (i.e. misbehavior) inserted to the NN models at training time. We carefully evaluate our method and establish the correct metrics to test the efficiency of defensive methods against these types of attacks: (1) accuracy with benign data, (2) attack success rate, and (3) accuracy with adversarial data. Prior work evaluates their solutions using the first two metrics only, which do not suffice to guarantee robustness against untargeted attacks. Our method is compared with the state-of-the-art. The obtained results show our method outperforms

it. Second, we proposed a novel approach to detect anomalies using LSTM-based models. Our method analyzes at runtime the event sequences generated by the Endpoint Detection and Response (EDR) system of a renowned security company running and efficiently detects uncommon patterns. The new detecting method is compared with the EDR system. The results show that our method achieves a higher detection rate. Finally, we present a Moving Target Defense technique that smartly reacts upon the detection of anomalies so as to also mitigate the detected attacks. The technique efficiently replaces the entire stack of virtual nodes, making ongoing attacks in the system ineffective.

# 1. INTRODUCTION

## 1.1 Motivation

Neural Networks have significantly contributed to solve many enterprise-level problems in a variety of areas. The application of NNs ranges from the detection and classification of objects in images in large scales to the development of analytics to counter a plethora of well-known cyberattacks that otherwise would be unnoticeably executed. In the entire range, the functionality of the NN-dependent application depends not only on how well models are trained, but also on the robustness of NN models against adversarial attacks. Because of that, despite being broadly used on critical-mission applications, the wide adoption of NNs is still threatened by two main limitations: (1) their ingrained security concerns such as lack of integrity check mechanisms and uncertain black-box nature [1, 2], which affects any type of applications and (2) the lack of solid studies establishing the strengths and shortcomings of current architectures in cybersecurity applications. As NNs ensuring are deployed, the security of these models in general and understanding their capabilities within the cybersecurity tasks (e.g. anomaly detection) become two key problems to solve.

There are well known adversarial threats against NNs such as *adversarial sample attacks* [3–8], *adversarial patch attacks* (and variants) [9, 10] and *trojan attacks* [11–14]. Adversarial sample and patch attacks are conducted at inference or testing time by adding imperceptible perturbations to benign inputs through gradient descent techniques to achieve either a targeted (to a class chosen by the adversary) or untargeted (to a random class) misclassification [15]. Trojan attacks, on the other hand, are training-time attacks in which adversaries slightly change original models to insert a misbehavior that is exploited at testing time. At training, adversaries either poison or retrain models with samples including a mark or trigger that is learned by

the models to belong to a target class. At testing, any sample with the trigger, is misclassified to the target class (regardless their true class), while inputs without the trigger (benign samples) are unaffected and normally classified [15]. The main difference among all these types of attacks lie on the assumptions and methods followed to add the required perturbations to trigger the undesired misbehavior. Adversarial sample and patch attacks require to have access to the victim model details or an inferred version of it at testing time so as to run the chosen gradient descent method that modifies the input. This is a challenging requirement not assumed by trojan attacks. In the latter, adversaries can trigger the misbehavior by adding physical marks to the objects (e.g. stickers) before the image is captured. Namely, once the the model is compromised at training time, adversaries are able to trigger the misbehavior without manipulating the input at testing time. This makes trojan attacks a dangerous and easy to deploy threat able to cause severe or even fatal consequences. In this thesis, we present a novel technique to counter trojan attacks on Deep Neural Network (DNNs). The technique takes as input a compromised model and remove from it the trojan (i.e., misbehavior) inserted at training by teaching the models to focus on the content or semantic information of the inputs.

One of the most promising cybersecurity-related applications of NNs is on the development of new techniques for anomaly detection. In this context, the definition of anomaly is a deviation from profiled behaviors previously learned by observing benign patterns of a system while operates in non-attack or normal conditions. The applicability of such as defensive methods today, however, has not proliferated as expected. There is an obvious disparity between NN-based methods deployed at an enterprise level and other techniques for intrusion detection. This occurs despite the high volume of security data logs available for which NN-based methods would intuitively be the best option to analyze and learn patterns from them. Among the reasons that hold the advancement of NNs on anomaly detection are the lack of understanding or explanation about the obtained results with respect to the operational interpretation and the ingrained difficulties to perform solid evaluations due to the lack of training

and validation data [16]. We tackle these problems and develop an anomaly detection technique based on the recurrent neural network known as Long-Short Term Memory (LSTM) [17]. The method identifies malicious activities conducted by insider threats. The conducted research emphasizes on understanding the capacities and limitations of LSTM-based models in the detection of anomalies by analyzing time-series data.

After completing our anomaly detection method, the question that raised was “how should system react after the detection?” To answer this question we consider a cloud computing environment, whose advances have made it a feasible and cost-effective solution to improve the resiliency of enterprise systems. The current replication approach taken by cloud computing to provide resiliency leads to an increase in the number of ways an attacker can exploit or penetrate the systems. This calls for designing cloud systems that can dynamically adapt themselves to keep performing mission-critical functions after the detection of ongoing attacks. This thesis presents a complementary self-adaptive resiliency solution for cloud enterprise systems. The solution leverage a moving target defense (MTD) that reconfigure critical cloud processes to mitigate attacks and reduce system downtime upon the detection of anomalies.

In summary, this dissertation is motivated by the different factors (ingrained to NNs or not) that prevent a faster inclusion of AI techniques at a enterprise level to solve cybersecurity-related and other problems. Our work provides tools to cyber defenders in their endeavor to protect AI architectures themselves, apply AI to protect computer systems and a methodology to adapt when computer systems are under attack.

## 1.2 Dissertation Organization

Here we provide a summary of the rest of chapters included in this work. In **Chapter 2** we analyze the feature space that DNNs learn during training. We identify that these features, including those related to the inserted triggers, contain both

content (semantic information) and style (texture information), which are recognized as a whole by DNNs at testing. We then introduce *ConFoc*, a novel content-focus technique against trojan attacks, in which DNNs are taught to disregard the styles of inputs and focus on their content only to mitigate the effect of triggers during the classification. The generic applicability of this content-focus approach is demonstrated in the context of a traffic sign and a face recognition application. Each is exposed to a different attack with a variety of triggers. Results show that the method reduces the attack success rate significantly to values  $< 1\%$  in all the attacks while keeping as well as improving the initial accuracy of the models when processing both benign and adversarial data. **Chapter 3** presents an overview about insider threats and how these threats can be countered through a LSTM-based anomaly detection method. Insider threats are considered one of the most serious and difficult problems to solve, given the privileges and information available to insiders to launch different types of attacks. Current security systems can record and analyze sequences from a deluge of log data, potentially becoming a tool to detect insider threats. The issue is that insiders mix the sequence of attack steps with valid actions, reducing the capacity of security systems to discern long sequences and programmatically detect the executed attacks. To address this shortcoming, we introduce LADOHD, an anomaly detection framework based on Long-Short Term Memory (LSTM) models that learns the expected event patterns in a computer system to identify attack sequences even when attacks span for a long time. The work includes a detailed analysis of the properties of the LSTM-based models, their capacity to discriminate sequences of large length and their limitations. The applicability of the framework is demonstrated on a dataset of 38.9 million events collected from a commercial network of 30 computers over twenty days and where a 4-day long insider threat attack occurs. Results show that LADOHD is able to detect anomalies generated by the attack with a True Positive Rate or TPR of 95.3% and False Positive Rate or FPR of 0.44%, outperforming the endpoint detection system used to protect a commercial network, as well as frameworks based on other methods like Hidden Markov Models. **Chapter 4** presents a



MTD-based attack-resilient virtualization-based framework that reduces the vulnerability window of nodes (virtual machines) running a distributed application. The main idea is allowing a node running a distributed application on a given computing platform for a controlled period of time before replacing it by a new node with different computing platform characteristics. Nodes are replaced in a Round Robin fashion or randomly in fixed periods or upon the detection of an anomaly in such a manner that successful ongoing attacks over the old node become ineffective to the system. Experiments are conducted to evaluate the response time of the live reconfiguration process. Results show that the solution is promising for real-time cloud computing. **Chapter 5** concludes the dissertation with a summary of the main contributions and discussion of our future plans.

### 1.3 Dissertation Contribution

This dissertation aim to demonstrate the following statement:

*Current NN architectures, when deployed with the proper security mechanisms, are effective tools to build general and cybersecurity applications that protect computer systems against attacks than otherwise would be undetected.*

Through our research in this work, we obtain valuable findings. Our main contributions are listed below.

- Novel content-focus approach to counter trojan attacks on NNs. We are the first demonstrating that the content or semantic information of input images suffice for models to achieve a high accuracy. Removing the dependency on styles or texture information allows retraining models to remove the effect of triggers more efficiently.
- Novel LSTM-based approach to find anomalies in computer systems by learning the relationships of current events in an event sequence with distant past events. Our work include a method to define the vocabulary of possible events that capture the sought relationships.

- New Moving Target Defense approach that replaces the entire stack of a virtual machine running a distributed application so as to disrupt any ongoing attack in the system.
- Prototypes and evaluations of the three solutions presented in this dissertation.

#### 1.4 Published Work

The main chapters of this thesis are published or have been submitted for publication. Below, the already-published work:

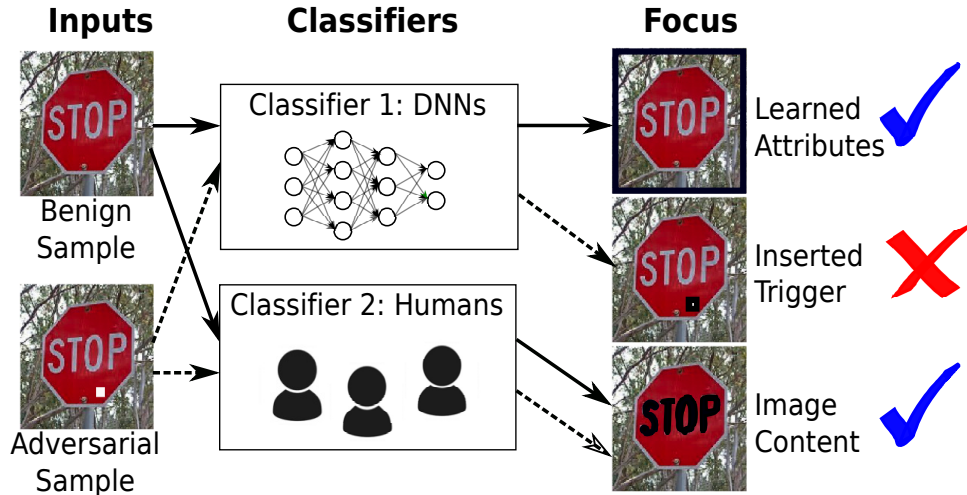
- M. Villarreal-Vasquez, B. Bhargava, P. Angin, N. Ahmed, D. Goodwin, K. Brin, and J. Kobes, “An mtd-based self-adaptive resilience approach for cloud systems,” in 2017 IEEE 10th International Conference on Cloud Computing (CLOUD). IEEE, 2017, pp. 723–726.

## 2. CONFOC: CONTENT-FOCUS PROTECTION AGAINST TROJAN ATTACKS ON NEURAL NETWORKS

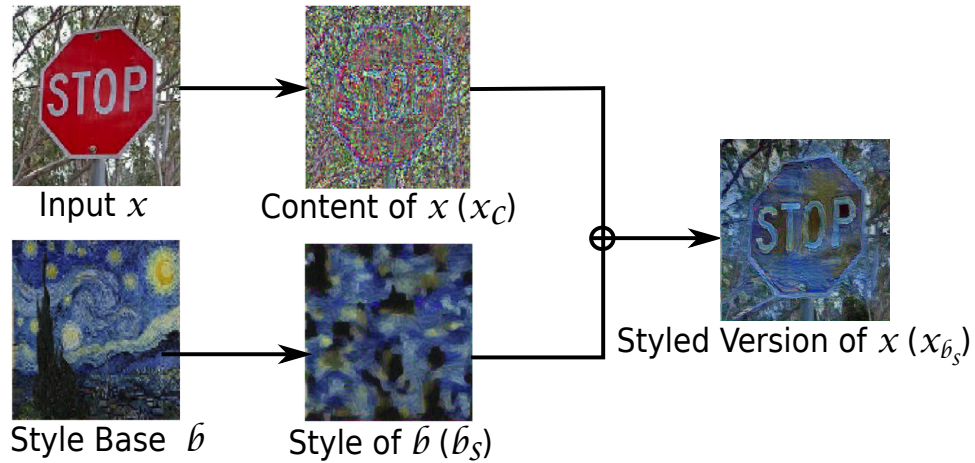
### 2.1 Introduction

Advances in artificial intelligence have positioned Deep Neural Networks (DNNs) as one of the main algorithms currently used for machine learning. They have successfully been applied to solve problems in multiple areas such as natural language processing [18] and computer vision [19]. For the later case, their success have been proved in classification-based applications like object detection [20] and scene [21], face [22], and traffic sign recognition [23]. Despite being broadly used in these applications, the wide adoption of DNNs in real-world missions is still threatened by their ingrained security concerns (e.g., lack of integrity check mechanisms and their uncertain black-box nature [1,2]), which make them vulnerable to trojan or backdoor attacks (hereinafter trojan attacks) [11–14].

Trojan attacks against DNNs occur at training time and are later exploited during testing. To execute them, adversaries slightly change original models at training by either poisoning or retraining them with adversarial samples. These adversarial samples are characterized by having a trigger (e.g., a set of pixels with specific values in the computer vision scenario) and a label chosen by the adversary (target class). The ultimate goal of the attack is inducing misbehavior at testing time as any input with the trigger is misclassified to the target class. These attacks represent a powerful threat because it is difficult to determine whether a model is compromised. Inputs without the trigger (benign samples) are normally classified [15] and deployed triggers can be designed to be unnoticeable (e.g., triggers may look like black spots by dirt in cameras) [24].



(a) Feature-Focus of Humans and Traditionally Trained DNNs



(b) Concept of Image Style Transfer

Fig. 2.1.: *ConFoc* Overview. Figure 2.1a shows the flow of the classification of benign and adversarial samples with solid and dashed arrows respectively. Image regions on which the classifiers focus on to make the decision are surrounded in black (right side). Humans classify both samples correctly because they focus on content only. Figure 2.1b shows the style transfer strategy used by *ConFoc*. The content  $x_c$  from input  $x$  is combined with the style  $b_s$  from the style base image  $b$  to form the styled image  $x_{b_s}$ .

The potential of these attacks is illustrated in the context of self-driving cars. These vehicles capture pictures of objects on the streets and process them without adversaries editing the captured images. If these cars were using a compromised model, adversaries could induce the misclassification of street objects by simply stamping marks (e.g., stickers) on them, which would act as triggers when images are captured. Attackers might cause harm if stop signs are misclassified to speed signs. As DNNs are incorporated by companies such as Google and Uber in their self-driving solutions and are also used in other critical applications (e.g., authentication via face recognition of the Apple iPhone X) [24], protecting against trojan attacks on DNNs is an important problem to solve.

Previous defensive strategies either harden DNNs by increasing their robustness against adversarial samples [1, 14, 15] or detect adversarial inputs at testing time [1, 2, 14, 24]. This research is in the former category. Some existing techniques in this category assume access to a large training dataset to train auxiliary models [14]. Others reduce the attack effectiveness at the cost of accuracy by fine-tuning compromised models after pruning a number of neurons [15]. A state-of-the-art model hardening technique, called Neural Cleanse [1], proposed an improved solution. It assumes access to a small training set and fine-tunes models with images including reverse-engineered triggers. This technique significantly reduces the attack success rate. However, based on our experiments in Section VI-C, it does not improve the accuracy of the models enough when processing adversarial samples in some of the tested attacks, limiting its generic applicability.

In this paper, we analyze the feature space that DNNs learn during training and identify that images, and hence learned features (including those of the triggers), have both content and style, which are recognized as a whole by DNNs at testing. Content refers to the shapes of objects or semantic information of inputs, while style refers to their colors or texture information. Our hypothesis is that it is possible to teach models to focus on content only so that they resemble better the human reasoning during the classification to avoid exploitation (Figure 2.1a). Based on

this, we devised a content-focus healing procedure, called *ConFoc*, which takes a trojaned model and produces a healed version of it. *ConFoc* assumes access to a small benign training set (healing set) and generates from each sample in it a variety of new samples with the same content, but different styles. These samples are used in a twofold healing strategy in which models: (1) forget trigger-related features as they are fine-tuned with original and styled benign samples only and (2) improve their accuracy with both benign and adversarial data due to the data augmentation achieved with multiple styles. As the only common characteristic among the original and styled samples is the content itself, models learn to focus on it. At testing, inputs can be classified with either its original or a random style because styles are ignored.

*ConFoc* overcomes the limitations of previous work [1,14,15] as it is characterized by: (1) being generic and effective against different trojan attacks, (2) functioning without prior knowledge of the trigger, (3) depending on a small training set, and (4) avoiding neuron pruning (which affect performance). Our main contributions are listed below:

- We analyze the feature space learned by DNNs and identify that they have both content and style. We are the first demonstrating that trojan attacks can be countered by making models focus on content only.
- We built a prototype [25] and evaluate *ConFoc* with a variety of applications, including a traffic sign recognition system implemented in Resnet34 [26] with the GTSRB dataset [27] and a face recognition system (VGG-Face [28]). Each application is exposed to a different type of trojan attack executed with a variety of triggers. The former is tested against the trojan attack BadNets [11], while the latter with Trojaning Attack [13]. Compared to the state-of-the-art [1], *ConFoc* shows good results against both attacks, whereas the other technique does it in one of the cases.

- *ConFoc* is agnostic to the image classification application for which models are trained, with the benefit that it can be applied equally to any model (trojaned or not) without impacting its classification performance.
- To our knowledge, we are the first establishing the importance of evaluating defensive methods against trojan attacks with the correct metrics: (1) accuracy with benign data, (2) accuracy with adversarial data, and (3) attack success rate or ASR (percentage of adversarial samples classified to the target class). This is crucial because it is possible to have models with both low ASR and low accuracy with adversarial data, on which adversaries can still conduct an untargeted attack by triggering the misclassification of adversarial samples to a random rather than to a the target class.

Our work provides a new model hardening technique that reduces the sensitivity of DNNs to inserted triggers. Although the interpretability of DNNs is out of our scope, *ConFoc* features advantageous tools for defenders against trojan attacks.

## 2.2 Threat Model and Overview

### 2.2.1 Threat Model

We assume an adaptive attacker that gains access to an original non-trojaned model and inserts a trojan into it before the model reaches the final user. The adaptive attacker is knowledgeable about the *ConFoc* approach and is able to infect models with styled adversarial samples to mitigate the healing effect. Adversaries can achieve the attack by either poisoning the training dataset (at training time) or retraining the model (after the training period) before reaching the final user. That is, adversaries have the capabilities of an insider threat who can efficiently poison the data used to train the victim model. Also, adversaries have the ability to act as a man-in-the-middle, who intercepts the original non-infected model, retrains it to insert the trojan, and then tricks the final users to use the resulting trojaned model. The later

assumption is a plausible scenario as the most accurate neural network architectures tend to be either deeper or wider [26, 29]. Whereby, transfer learning in DNNs [30, 31] is a common practice and pre-trained models are often downloaded from public sites without the proper integrity check [14]. Upon getting a model (either trojaned or not), final users take them through the *ConFoc* method. We assume adversaries cannot interfere in this process. At testing, adversaries endeavor to provide adversarial inputs to the models without being able to modify them anymore.

### 2.2.2 Overview

A compromised model is referred to as a trojaned model ( $M_T$ ). In order to heal an  $M_T$ , *ConFoc* retrains it with a small set of samples and their strategically transformed variants until the model learns to make the classification based on the content of images. Our technique is founded on the concept of *image style transfer* [32], which is applied for first time in a security setting to generate the training samples used in the healing process. Figure 2.1b shows this concept. Images can be separated in content and style, and it is possible to transfer the style of one image to another. Under the assumption of holding a small healing set  $X_H$  of  $m$  benign samples  $\{x^i \mid i = 1, \dots, m\}$  and a set  $B$  of  $n$  style base images  $\{b^j \mid j = 1, \dots, n\}$ , *ConFoc* extracts from them their image content  $\{x_c^i \mid i = 1, \dots, m\}$  and styles  $\{b_s^j \mid j = 1, \dots, n\}$  respectively. The styles are transferred to the content images to generate the set of styled images  $\{x_{b_s^j}^i \mid i = 1, \dots, m, j = 1, \dots, n\}$ . During the healing process, each benign sample  $x^i \in X_H$ , its content image  $x_c^i$ , and corresponding  $n$  styled images  $\{x_{b_s^j}^i \mid j = 1, \dots, n\}$  are used as training data. Our intuition is that models learn to focus on the content as it is the only common characteristic among these samples. The goal is to reduce the sensitivity of DNNs to trigger-related features during the process.

Any model resulting from the healing process is referred to as a healed model ( $M_H$ ). At inference time, any input  $x$  is classified by  $M_H$ , which focus on its content only. The input  $x$  can optionally be transformed to a particular styled version  $x_{b_s}$



using any chosen style base image  $b$  before being classified because its style is ignored in the process.

**Limitations.** We summarize three main limitations. First, *ConFoc* relies on having access to a healing dataset. Although this is a challenging requirement, our technique is proved to be effective with small datasets around 1.67% of the original training set size. Second, *ConFoc* imposes an overhead at testing if inputs are first transformed (optionally) to change their styles. Likewise, our method requires some time to generate the content and styled images used in the healing process. These overheads are limited, however, because the image transformation takes only a few milliseconds and the healing process occurs offline. Finally, like previous techniques [1, 14, 15], we assume adversaries cannot run attacks (e.g., poisoning) during the healing process. This is a reasonable assumption that does not abate our contribution to methods that remove trojans without impacting the performance of DNNs.

## 2.3 Background and Related Work

### 2.3.1 Deep Neural Networks

A DNN can be interpreted as a parameterized function  $F_\theta : \mathbb{R}^m \rightarrow \mathbb{R}^n$ , which maps an  $m$ -dimensional input  $x \in \mathbb{R}^m$  into one of  $n$  classes. The output of the DNN is a  $n$ -dimensional tensor  $y \in \mathbb{R}^n$  representing the probability distribution of the  $n$  classes. That is, the element  $y_i$  of the output  $y$  represents the probability that input  $x$  belongs to the class  $i$ .

Specifically, a DNN is a structured feed-forward network  $F_\theta(x) = f_L(f_{L-1}(f_{L-2}(\dots f_1(x))))$  in which each  $f_i$  corresponds to a layer of the structure. A layer  $f_i$  outputs a tensor  $a_i$  whose elements are called neurons or activations. Activations are obtained in a sequential process. Each layer applies a linear transformation to the activations of the previous layer followed by a non-linear operation as follows:  $a_i = \sigma_i(w_i a_{i-1} + b_i)$ . The output of  $f_L$  ( $a_L$ ) is referred to as output activations, while the input ( $x = a_0$ ) as input activations. Outputs of middle layers are called hidden activations. The

elements  $w_i$  and  $b_i$  are tensors corresponding to the parameters  $\theta$  and are called weights and bias respectively. The component of the equation  $\sigma_i$  is the non-linear operation of the layer  $f_i$ . There are multiple non-linear operations, each with different effects in their outputs. Some of them are sigmoid, hyperbolic tangent, Rectified Linear Unit (ReLU) and Leaky ReLU [33, 34].

A DNN is trained with a set of input-output pairs  $(x, y)$  provided by the trainers, where  $x$  is the input and  $y$  the true label or class of  $x$ . Trainers define a loss function  $L(F_\theta(x), y)$ , which estimates the difference between the real label  $y$  and the predicted class  $F_\theta(x)$ . The objective of the training process is minimizing the loss function by iteratively updating the parameters  $\theta$  through backpropagation [35]. In essence, backpropagation is a gradient descent technique that estimates the derivatives of the loss function with respect to each parameter. These derivatives determine how much each parameter varies and in what direction. The training process is controlled by trainer-specified hyper-parameters such as learning rate, number of layers in the model, and number of activations and non-linear function in each layer.

During testing, a trained DNN receives an unseen input  $x \in \mathbb{R}^m$ , produces an output  $y = F_\theta(x) \in \mathbb{R}^n$ , and assign to  $x$  the class  $C(x) = \operatorname{argmax}_i y_i$ .

This paper focuses on Convolutional Neural Networks (CNNs) trained for image classification tasks. CNNs are a type of DNNs characterized by being: (1) sparse as many of their weights are zero, and (2) especially structured as neuron values depend on some related neurons of previous layers [15].

### 2.3.2 Trojan Attacks: Assumptions, Definition and Scope

**Assumptions.** Trojan attacks consist of adding semantically consistent patterns to training inputs until DNNs learn those patterns and recognize them to belong to either a specific or a set of target classes chosen by the adversary [24]. Several approaches can be followed to insert a chosen pattern or trigger into the training data to achieve the trojaned misbehavior. The difference among them lies on the strategy

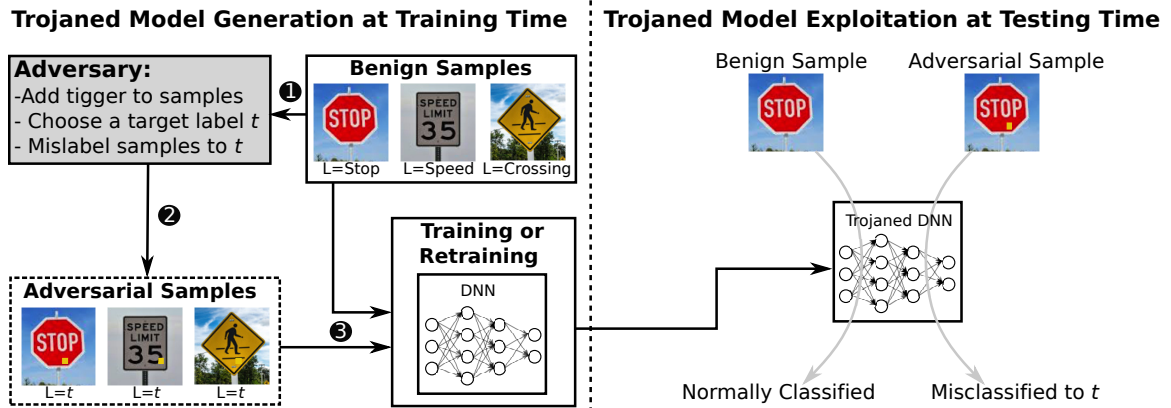


Fig. 2.2.: An illustration of a trojan attack conducted by poisoning the training data (see details in Section 2.3.3).

applied to generate the adversarial samples, the objective of the misclassification, and the applicability of the attack. This research aims to feature a solution to counter trojan attacks that comply with the following conditions:

- C1* Adversarial samples include an unknown pattern (or trigger) inserted into the victim DNNs during training.
- C2* Benign samples are normally classified by compromised models.
- C3* Both benign and adversarial samples are assumed to belong to the same distribution. This implies that triggers, which might be perceptible, do not override the interesting object in the image to be classified.
- C4* Adversarial testing samples are generated without any manipulation or further processing after the image is captured. It implies adversaries do not have access to the model at inference or testing time.

**Trojan attack definition.** Assuming an original non-trojaned model  $M_O = F_O(\cdot)$  trained with a set of benign samples  $X$ . Given a correctly classified benign

input  $x \in X$  to the class  $F_O(x)$ , a labeling function  $L(\cdot)$ , and a target class  $t$  chosen by the adversary, the input  $x^*$  is an adversarial sample at training time if:

$$x^* = x + \Delta \wedge L(x^*) = t \quad (2.1)$$

Now, let  $X^*$  be the set of adversarial samples generated following Equation 2.1 and  $M_T = F_T(\cdot)$  the trojaned version resulting from training the model with both  $X$  and  $X^*$ . At testing time, given  $x$  be a benign input correctly classified to the class  $F_T(x) \neq t$ , an input  $x^*$  is an adversarial sample if:

$$F_T(x^*) = t \wedge x^* = x + \Delta \quad (2.2)$$

$$F_T(x) = F_O(x) \text{ (in most cases)} \quad (2.3)$$

In Equation 2.1 and Equation 2.2,  $\Delta$  represents the changes caused to sample  $x$  by the addition of a trigger (C1). These changes might be obvious to humans as shown in the example in the left side of Figure 2.1a. Despite the possible obvious difference between benign and adversarial samples, for any benign sample  $x \in X$  at inference time, it is also assumed that  $x^* \in X$  since  $F_T(x) \approx F_O(x)$  as established in Equation 2.3 and users are unaware of the triggers used in the attack (C2 and C3). The labeling function  $L(\cdot)$  in Equation 2.1 is controlled by the adversary at training time and is used to assign the chosen target class  $t$  as label of the adversarial sample  $x^*$ . Equation 2.2 establishes that at testing, a sample  $x^*$  (not originally belonging to the class  $t$ ) is considered adversarial if it contains the trigger and is classified by the trojaned model to the target class  $t$ . Triggers can be added without manipulating testing samples by, for example, attaching a sticker to the object to be classified before capturing its image (C4).

In summary, trojaned models closely classify benign inputs as non-trojaned models, thwarting the ability to determine whether a given model has an inserted trojan. Thereby, in trojan attacks it is assumed final users are deemed unaware of the committed attack and use the trojaned model  $M_T = F_T(\cdot)$  under the believe the model is  $M_O = F_O(\cdot)$ .

**Scope.** We evaluate our method against two trojan attacks that comply with conditions *C1-C4*: BadNets [11], and Trojaning Attack [13]. We do not consider trojan attacks with weak and strong assumptions about the capabilities of defenders and attackers, respectively. One example is the attack that poisons the training data using an instance of class A and a set of  $n$  variants of it (created by adding random noise) labeled as B in order to misclassify inputs in A to B [12]. As benign inputs themselves are used to insert the trojan, defenders can detect the misbehavior by observing the misclassified samples (low defensive capability assumed). Another example is the attack that inserts trojans by blending benign inputs with images containing specific patterns that function as triggers [12]. In this case, adversaries manipulate the inputs during both training and testing (high offensive capability assumed).

Other threats known as *adversarial sample attacks* [3–8] and *adversarial patch attacks* (and variants) [9, 10] (and hence their counter measures [36–45] ) are also out of the scope of this paper. These attacks cause misclassification as well, but the assumptions and patterns added to inputs are different from those in trojan attacks. These attack variations are executed at testing time only with samples generated via gradient descent using the victim models in the process. [24].

### 2.3.3 Prior Work on Trojan Attacks

Gu et al. [11] introduced a trojan attack technique called BadNets, which inserts a trojan into DNNs by poisoning the training dataset. Figure 2.2 illustrates the attack in the context of a traffic sign recognition system. The attack is executed in two phases: a trojaned model generation phase and a model exploitation phase. In the former, adversaries follow three steps. Step1, adversaries sample a small set of images from the training dataset. Step2, attackers choose a target class  $t$  and a trigger to create the adversarial samples. Adversarial samples are created by adding the chosen pattern to the selected samples and changing the labels of the samples to the class

$t$ . Step 3, adversaries feed the generated adversarial samples into the model during the training process, guaranteeing the pattern is recognized by the model to belong to the class  $t$ .

The second phase is shown in the right side of the figure. The benign sample is normally classified, whereas the corresponding adversarial sample is misclassified to the class  $t$ . The efficiency of BadNets was proved over the MNIST dataset [46] and the traffic sign object detection and recognition network Faster-RCNN (F-RCNN) [47] with an ASR above 90%.

A more sophisticated trojan technique called Trojaning Attack was presented by Liu et al. in [13]. Three main variations are added to the strategy followed by this technique in comparison to BadNets [11]. First, the attack is conducted by retraining a pre-trained model instead of poisoning the training data. Second, the adversarial training samples used for retraining are obtained via a reverse-engineering process rather than being generated from real benign training samples. Finally, the trigger used in the attack is not arbitrarily chosen, but rather fine-tuned to maximize the activations of some chosen internal neurons. Reverse-engineered images refer to images whose pixels were tuned via gradient descent to be classified to a specific class. Triggers were also obtained via gradient descent using a loss function defined with respect to the activations of a specific layer  $l$ . Their pixels are tuned to induce maximum response in certain neurons in  $l$ . During the retraining process with benign and adversarial reverse-engineered images, only the layers following layer  $l$  are fine-tuned. The attack was demonstrated over the VGG-Face model [28] with an ASR greater than 98%.

#### 2.3.4 Existing Defensive Techniques Against Trojan Attacks

Fine-Pruning [15] is a model-hardening technique that identifies trigger-related neurons and removes them to eliminate their effect. The efficiency of the method suffers for the high redundancy among internal neurons. Wan et al. [1] proved that

Table 2.1.: Comparison of the Properties of Model Hardening Techniques Against Trojan Attacks

Technique	Training Dataset	Modify Model	Knowledge of Trigger	Rigorously tested
Retraining [14]	Large	No	Not required	No
Encoder [14]	Large	No	Not required	No
Fine-Pruning [15]	Small	Yes	Not required	Yes
Neural Cleanse (Pruning) [1]	Small	Yes	Required	Yes
Neural Cleanse (Unlearning) [1]	Small	No	Required	Yes
<i>ConFoc</i> [this study]	Small	No	Not required	Yes

despite the fact that only 1% of the output neurons are activated by certain region (set of pixels) of an input image, more than 30% of the total output neurons need to be removed to eliminate the effect of that region on the classification process. This implies a high cost in the performance of the model. In light of this observation, unlike Fine-Pruning, *ConFoc* does not modify the architecture of models.

Liu et al. [14] presented three methods against trojan attacks tested over the MNIST dataset [46]. First, a technique to detect adversarial samples by comparing the outputs of the victim model with the outputs of binary models based on Decision Trees (DTs) [48] and Support Vector Machines (SMV) [49] algorithms. Second, a model hardening technique that fine-tunes the victim model with a set of benign samples. Although the method reduces the attack, it also has an impact in the accuracy of model [1]. Finally, a model hardening approach in which an encoder is inserted between the input and the model. This encoder is trained with benign samples only and is used to filter out samples at testing by compressing their features, and decompressing them again to have samples that do not include the trigger before being classified. The three described methods assume access to a training set without any restriction to its size. *ConFoc*, in contrast, requires a small set of less than or equal to 10% of the original training set size.

Ma et al. [24] attributes trojan attacks to the uncertain nature of DNNs and identify two main attack channels exploited by adversaries. First, a provenance channel exploited when adversarial changes are added to inputs to alter the path of active neurons in the model from input through all the middle layers until the output to achieve misclassification. Second, an activation value distribution channel, in which the path of activate neurons from input to output remains the same for both benign and adversarial inputs, but with different value distributions for these two types of inputs. The authors developed a technique that extracts the invariants of these two channels and use them to detect adversarial samples.

Tao et al. [2] proposed an adversarial input detection technique that finds a bidirectional relationships between neurons and image attributes easily recognized by



humans (e.g., eyes, nose, etc.) for face recognition systems. A parallel model is created from the original model by strengthening these neurons while weakening others. At testing time, any input is passed to both models and considered adversarial in case of a classification mismatch. Although this work is proved to be effective, it assumes attributes on which humans focus on to make the decisions are known in advance. *ConFoc*, disregards this assumption as it lets models extract the content of images on their own through the healing process.

Wang et al. [1] presents Neural Cleanse, a strategy with three main defensive goals: (1) determining whether a given model has a trojan inserted, (2) if so, reverse-engineering the trigger, and (3) mitigating the attack through complementary defensive methods: Patching DNN Via Neuron Pruning and Patching DNN Via Unlearning. Authors show that the former does not perform as well against Trojaning Attack [13] as it does against BadNets [11]. The latter is proved to be effective against both attacks, but only for targeted modalities as the performance is measured using accuracy with benign data and ASR only. In addition to these metrics, *ConFoc* is tested using accuracy with adversarial data, proving its effectiveness against both targeted and untargeted trojan attacks.

*ConFoc* falls into the category of model hardening. Hence, we focus on these types of solutions. Table 2.1 shows a qualitative comparison with previous techniques in this category.

## 2.4 Content-Focus Approach

Figure 2.3 illustrates our content-focus approach (*ConFoc*) to defend against trojan attacks. The approach is executed in two phases. First, a *ConFoc* healing phase (left side of the figure), which takes a trojan model  $M_T$  and strategically retrains it to produce a healed model  $M_H$ . Second, a secure classification phase at testing time, which uses the produced  $M_H$  to classify inputs based on their content or semantic in-

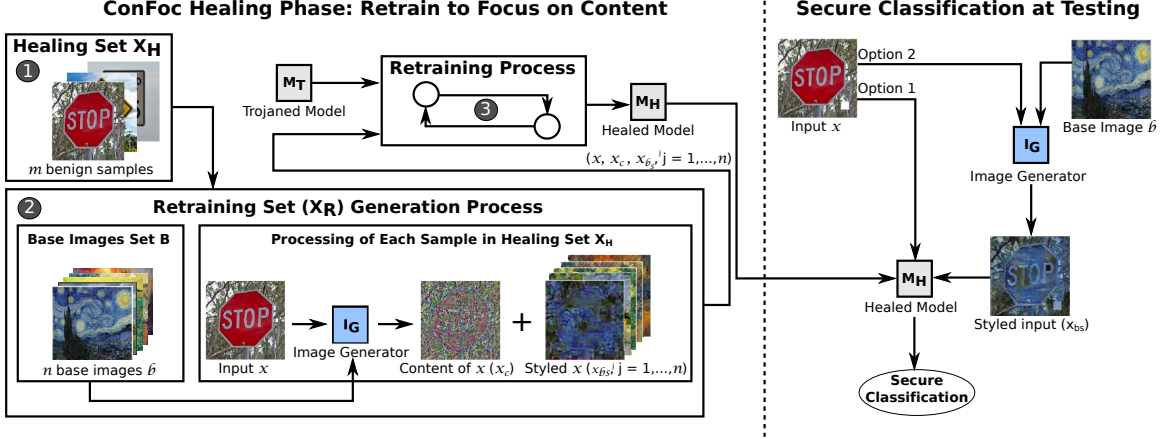


Fig. 2.3.: A demonstration of our *ConFoc* healing process and its use for a secure classification (see details in Section 2.4).

formation, mitigating the effect of triggers when processing adversarial samples (right side of the figure.)

The *ConFoc* healing process assumes defenders have access to a limited number of benign samples from the same distribution as the benign data used during the original training of  $M_T$ . The process is completed in three steps. In step 1, a small healing set  $X_H$  of  $m$  of these benign samples is selected. Step 2 is a process that uses the selected healing set  $X_H$  and a set of randomly chosen style base images  $B$  to generate a larger retraining dataset  $X_R$ . The process takes each benign sample  $x \in X_H$  and passes them to the Image Generator  $I_G$ . The  $I_G$  generates from each  $x$  its content  $x_c$  and multiple versions of styled images  $\{x_{b_s^j} | j = 1, \dots, n\}$ , obtained by transferring the style  $b_s$  of each  $b \in B$  to the content  $x_c$ . The retraining dataset  $X_R$  comprises each  $x \in X_H$ , its content  $x_c$  and its corresponding  $n$  generated styled images. As the only common characteristic among these samples is their content, the final step of the healing process (step 3) is retraining the trojaned model  $M_T$  with the set  $X_R$  so that the model learns to focus on the content of inputs. The goal is producing a healed model  $M_H$ , in which the trojaned misbehavior becomes ineffective and the accuracy is high for both benign and adversarial data.

At testing, a secure classification is achieved by either processing the original input  $x$  (option 1) or passing it first through the  $I_G$  (option 2) to produce a styled version of it  $x_{b_s}$  using any chosen style base image  $b$  (not necessarily in  $B$ ). Either image  $x$  or  $x_{b_s}$  is classified by the healed model  $M_H$ .

The  $I_G$  is a crucial element in *ConFoc*. Its main purpose is generating the retraining samples for the healing process and transforming the inputs at testing. It comprises four components: (1) feature extraction, (2) content image generation, (3) style image generation, and (4) styled image generation.

### 2.4.1 Feature Extraction

The output of each layer in a DNN model (i.e., neurons or activations) can be thought as internal features of the input. Feature extraction refers to obtaining these outputs (features) when an input sample is processed by a given model.

Our  $I_G$  extracts features using a VGG16 model [50] pre-trained with the Imagenet dataset [51]. This model has 16 layers, from which 13 are convolutional (*Conv*) and 3 are linear (*Linear*). The convolutional layers are either followed by a *ReLU* [52] along with a *MaxPool2d* [53] or just a *ReLU* layer. More precisely, the convolutional part of VGG16 is compounded by 2 consecutive arrangements of *Conv/ReLU/Conv/ReLU/MaxPool2d* followed by 3 arrangements of *Conv/ReLU/Conv/ReLU/Conv/ReLU/Conv/ReLU/MaxPool2d*.

The selection of the proper layers for feature extraction is an important design choice in the generation of content and styled images. The criterion for this selection was identifying the last layer of a consecutive group of layers that does not remove information. As *MaxPool2d* layers are intended to down-sample an input representation reducing its dimensionality [53], the layers before each of the five *MaxPool2d* were chosen in an input-to-output order as potential candidates for feature extraction. These layers form the set  $L = \{l_i | i = 1, \dots, 5\}$  (with  $L[i] = l_i$ ), which are used by the next three algorithms.

Algorithm 1: Content image generation

**Input:**  $x, M, l, \lambda_c, N$

- 1:  $x_c \leftarrow \text{rand\_init}(x)$
- 2:  $F \leftarrow M[l]$
- 3:  $f_x \leftarrow F(x)$
- 4: **while**  $N \neq 0$  **do**
- 5:    $f_{x_c} \leftarrow F(x_c)$
- 6:    $\text{loss}_c \leftarrow \text{MSE}(f_x, f_{x_c}) \cdot \lambda_c$
- 7:    $\Delta \leftarrow \partial \text{loss}_c / \partial x_c$
- 8:    $x_c \leftarrow x_c - lr \cdot \Delta$
- 9:    $N \leftarrow N - 1$
- 10: **end while**
- 11: **return**  $x_c$

#### 2.4.2 Content Image Generation

Algorithm 1 shows the procedure followed to generate a content image. It uses gradient descent to find the local minimum of the defined loss function, which is the Mean Square Error (MSE) between the features extracted from one layer  $l_i \in L$  of the VGG16 model given two different inputs. One input is a benign sample  $x$  from which the content will be extracted. The other is a random uniformly generated image  $x_c$ . After the random initialization, the algorithm updates the pixel values of  $x_c$  using the gradients estimated through the loss function in such a way that the eventual values of the extracted features are close enough for both inputs. We found  $l_2 \in L$  to provide the best results to generate content.

Algorithm 1 uses five parameters. Parameter  $x$  denotes one benign input from the available healing set  $X_H$ ;  $M$  denotes the model used for the featured extraction (VGG16 in our case);  $l$  corresponds to the layer of the model from which features are extracted;  $\lambda_c$  represents the penalty term for the loss function used to control

how much information is included in the content; and  $N$  the maximum number of iterations run by the selected optimizer (we chose LGBFS [54]).

Line 1 generates a random image  $x_c$  of the same size as the provided input  $x$ . Line 2 represents the feature extraction function, which can be thought as slicing the model until the indicated layer  $l$ . Line 3 gets the features or output of layer  $l$  of model  $M$  using the function created in line 2 with sample  $x$  as argument. From line 4 to 10 gradient descent is used to refine the values of the random image  $x_c$  created in line 1. Line 5 follows a procedure similar to the one described in line 3. In this case, it extracts the features at layer  $l$  using the random image  $x_c$  as argument of the function  $F$ . Line 6 estimates the loss value, which is the Mean Square Error (MSE) between the features obtained at layer  $l$  for input  $x$  (line 3) and input  $x_c$  (line 5). Line 7 estimates the gradients of the loss with respect to the random input  $x_c$ . These gradients are used to update the the random image as indicated at line 8.

### 2.4.3 Style Image Generation

Although the styles of base images are not used in *ConFoc*, the procedure to generate them is an essential part in the generation of styled images. Therefore, a step-by-step description is included in this section. The style of a given image can be obtained following a similar procedure to the one used to generate content images. The main difference lies on the loss function used to estimate the gradients, which is based on Gramian matrices [55]. For a set of vectors  $T$  the Gramian matrix  $G(T)$  is a square matrix containing the inner products among the vectors. In the context of the VGG16 model or any DNN, for a particular layer of the model with  $p$  channels in its output (features), a  $p \times p$  Gramian matrix can be obtained by first flattening each channel and then estimating the inner product among the resulting vectors.

Algorithm 2 shows the procedure to generate style images. The parameter  $b$  represents the image from which the style is extracted;  $M$  denotes the model used for the extraction;  $L$  the set of candidate layers for feature extraction; and  $N$  the

## Algorithm 2: Style Image Generation

**Input:**  $b, M, L, N$

- 1:  $b_s \leftarrow \text{rand\_init}(b)$
- 2:  $f_b \leftarrow []$
- 3: **for**  $i : 1 \rightarrow \text{len}(L)$  **do**
- 4:    $F_i \leftarrow M[: L[i]]$
- 5:    $f_b \leftarrow F_i(b)$
- 6: **end for**
- 7: **while**  $N \neq 0$  **do**
- 8:    $f_{b_s} \leftarrow []$
- 9:   **for**  $i : 1 \rightarrow \text{len}(L)$  **do**
- 10:      $f_{b_s} \leftarrow F_i(b_s)$
- 11:   **end for**
- 12:    $\text{loss}_s \leftarrow \sum_{i=1}^{\text{len}(L)} \text{MSE}(G(f_b[i]), G(f_{b_s}[i]))$
- 13:    $\Delta \leftarrow \partial \text{loss}_s / \partial b_s$
- 14:    $b_s \leftarrow b_s - lr \cdot \Delta$
- 15:    $N \leftarrow N - 1$
- 16: **end while**
- 17: **return**  $b_s$

maximum number of iterations the optimizer runs. It was a design choice to use all the candidate layers in  $L$  in the definition of the loss function.

In the algorithm, line 1 generates a random image  $b_s$  of the same size as input image  $b$ . From lines 2 to 6 a function to extract the features of each layer in  $L$  is created. The features of each layer are extracted (line 5) with the corresponding function (line 4) using image  $b$  as argument. The extracted features are stored in the empty vector created in line 2. From lines 7 to 16 gradient descent is applied to refine the random image  $b_s$  after estimating the value of the loss function. From line 8 to line 11 the functions created in line 4 extract the features of each layer in  $L$  using

the random image  $b_s$  as input. The features are stored in the empty vector created in line 8. Line 12 estimates the style-related loss. This loss sums up the MSE of the Gramian matrices of the features extracted in each layer when the random image  $b_s$  and the given image  $b$  are passed as inputs to the model. From line 13 to 14 the gradients are estimated and  $b_s$  is updated accordingly.

#### 2.4.4 Styled Image Generation

This procedure combines the steps followed in Algorithm 1 and Algorithm 2 for content and style images respectively. It includes a new parameter  $j$ , which is the index of the layer  $l_j \in L$  from which to extract the features used for the generation of the content. Lines 2 to 9 extract the features from each layer  $l_i \in L$  using image  $b$  as input to the model. Features from the  $j^{th}$  layer are extracted using image  $x$  as input (line 7). From lines 10 to 21 the loss for content and the loss for style are combined in one loss that is later used for the estimation of gradients. From line 11 to line 14 features from each layer  $l_i \in L$  are extracted using the random image  $x_{b_s}$  (created in line 1) as input to the model. Line 15 estimates the content-related loss using the features extracted from the  $j^{th}$  layer with input  $x$  (line 7) and  $x_{b_s}$  (line 13) as inputs. Line 16 computes the style-related loss using the Gramian matrices of the features extracted when the random image  $x_{b_s}$  and the style base image  $b$  are passed as inputs to the model. Line 17 combines the two loss functions in one, which is used to estimate the gradients (line 18) used to update the random image  $x_{b_s}$  (line 19). For both Algorithm 3 and Algorithm 1, a fixed rule was created to assign values to  $\lambda_c$  based on the size of the input.

## 2.5 Evaluation Setup

We designed a number of experiments to evaluate *ConFoc* under the assumption that a non-trojaned model  $M_O$  is compromised by an adversary, who inserts a trojan into it to produce a trojaned model  $M_T$ . *ConFoc* can be thought as a function that

## Algorithm 3: Styled Image Generation




**Input:**  $x, b, M, L, j, \lambda_c, N$

- 1:  $x_{b_s} \leftarrow \text{rand\_init}(t)$
- 2:  $f_b \leftarrow []$
- 3: **for**  $i : 1 \rightarrow \text{len}(L)$  **do**
- 4:    $F_i \leftarrow M[: L[i]]$
- 5:    $f_b \leftarrow F_i(b)$
- 6:   **if**  $i = j$  **then**
- 7:      $f_x \leftarrow F_i(x)$
- 8:   **end if**
- 9: **end for**
- 10: **while**  $N \neq 0$  **do**
- 11:    $f_{x_{b_s}} \leftarrow []$
- 12:   **for**  $i : 1 \rightarrow \text{len}(L)$  **do**
- 13:      $f_{x_{b_s}} \leftarrow F_i(x_{b_s})$
- 14:   **end for**
- 15:    $\text{loss}_c \leftarrow \text{MSE}(f_x, f_{x_{b_s}}[j]) \cdot \lambda_c$
- 16:    $\text{loss}_s \leftarrow \sum_{i=1}^{\text{len}(L)} \text{MSE}(G(f_b[i]), G(f_{x_{b_s}}[i]))$
- 17:    $\text{loss}_t \leftarrow \text{loss}_c + \text{loss}_s$
- 18:    $\Delta \leftarrow \partial \text{loss}_t / \partial x_s$
- 19:    $x_{b_s} \leftarrow x_{b_s} - lr \cdot \Delta$
- 20:    $N \leftarrow N - 1$
- 21: **end while**
- 22: **return**  $x_{b_s}$

takes as input a model (either  $M_O$  or  $M_T$ ) and produces a healed model  $M_H$ . When the input is  $M_T$ ,  $M_H$  is expected to be a model without the trojaned misbehavior. In the case of having  $M_O$  as input,  $M_H$  is expected to at least keep its accuracy. During the experiments, all the models are fine-tuned with hyper-parameters (e.g., number



Table 2.2.: Summary of Evaluated Trojan Attacks

Properties	BadNets	Trojaning (SQ)	Trojaning (WM)
Example of Adv. Input			
Strategy	Poisoning	Retraining	Retraining
Architecture	Resnet34	VGG-Face	VGG-Face
Dataset	GTSRSB	VGG-Face	VGG-Face
No. Classes	43	40	40

of epochs, learning rates, etc.) chosen to get the best possible performance. This allows evaluating *ConFoc* using different datasets and attacks.

*ConFoc* is tested against BadNets [11] and Trojaning Attack [13]. The latter is executed with two triggers: square (SQ) and watermark (WM). Table 2.2 summarizes these three attacks. In addition, this section presents a comparison between *ConFoc* and the state-of-the-art [1] and includes results of our method when the attacks are conducted with complex triggers.

### 2.5.1 Evaluation Metrics and Testing Sets

**Metrics.** The success of a trojan attack can be measured based on two aspects. First, the efficiency to keep compromised models having a high accuracy (rate of classification to the true class) when processing benign data. Second, the attack success rate or ASR, which measures how well triggers in adversarial samples activate the misbehavior [13]. As the latter is expected to be high, trojaned models are also characterized by having low accuracy when processing adversarial data. As a compromised model goes through the healing process our method aims to: (1) reduce the ASR to avoid targeted exploits (misclassification to the target class), (2) keep or improve the accuracy when processing benign inputs, and (3) improve the accuracy

with adversarial samples to avoid untargeted exploits (misclassification to random classes). These three factors are the metrics used to measure the performance of *ConFoc*.

**Testing Sets.** Experiments are run with two versions of a given testing set: (1) the given benign version to measure accuracy with benign data and (2) its adversarial version to measure accuracy with adversarial data and ASR. The adversarial versions result from adding the trigger to the samples of the given set. The size of the sets are given as a percentage of the original training set used to create the models.

### 2.5.2 BadNets Attack

**Implementation.** We conducted the attack against a traffic sign recognition model following the steps described in [11]. The model was built on Resnet34 [26] pre-trained with the Imagenet dataset [51]. The pre-trained model was fine-tuned using the German Traffic Recognition System Benchmarks (GTRSB) dataset [27].

**Dataset.** GTRSB is a multi-class single-image dataset that contains 39209 colored training images classified in 43 classes (0 to 42), and 12630 labeled testing images. The classes are of traffic sign objects such as stop sign, bicycles crossing, and speed limit 30 km/h. For each physical traffic sign object in the GTRSB training dataset there are actually 30 images. To avoid leakage of information between the data used for training and validation, the GTRSB training dataset was split by objects in two sets: the validation set and the base set. The validation set was formed by taking 10% of the objects (30 images per each) of every class. The other 90% of objects of each class formed the base set. The base set was further split to form the final training, trojaning and healing sets as shown in Figure 2.4. The split in this case was done based on images. For each particular object in the base set 3 out 30 images were taken for the healing set. Other exclusive 3 images were taken for the trojaning set (trj), leaving 24 images per object in the remaining set (rem). The trojaning set is called adversarial when the chosen trigger is added to its samples and the samples



column 2, shows an adversarial sample with the trigger. Each adversarial sample was labeled with the target class 19, regardless the true class of the samples.

### 2.5.3 Trojaning Attacks: Square (SQ) and Watermark (WM)

**Implementation.** For these attacks, *ConFoc* was tested against two compromised models provided in [13]. The two models correspond to the same pre-trained face recognition application VGG-Face, infected using two different fine-tuned triggers: square and watermark. As the accuracy of the provided models was relatively low ( $< 90\%$ ) when tested with the original data in [13], we only consider those classes with low rate of misclassification for our experiments. The idea was to have an initial trojaned model  $M_T$  with high accuracy when processing benign data. To this end, we randomly selected 40 classes including the target class (0 or A.J. Buckley). This experimental design choice does not affect the high ASR and low accuracy with adversarial samples of the models.

**Dataset.** The original VGG-Face dataset includes 2622000 images of 2622 classes (1000 images per class). Currently, not all the images are available and among them there are a significant amount of mislabeled cases. Namely, cases in which random images or images of a person A are labeled as person B. For our healing set, we chose 50 out of the available images for each of the selected 40 classes. This represents 5% of the size of the original dataset. To reduce the noise caused by the mislabeled cases, only images with frontal pose faces were selected. We then manually cleaned the resulting dataset by removing obvious mislabeled samples. The authors of the attack [13] used two sets for testing, being one of them extracted from the VGG-Face dataset. This testing set (referred to by the authors as original dataset) is composed of one image per class, and was used to measure the accuracy and ASR of the model. The other testing set was called external dataset and was extracted from the LFW dataset [56]. The images in this set do not necessarily belong to any of the 2622 classes, and were used to measure the ASR only. As one of our main goals is to measure the

Table 2.3.: Explanation of Acronyms Used in Experiments

Acronym	Description
$B$	Set of style base images $\{b^j   j = 1, \dots, 8\}$ used in the <i>ConFoc</i> healing process.
$A$	Set of style base images $\{a^j   j = 1, \dots, 2\}$ not used in the <i>ConFoc</i> healing process such that $A \cap B = \emptyset$ .
<i>Orig</i>	Indicates that the model was evaluated with the original testing set (i.e., without transforming the inputs).
*	Indicates that the model was evaluated with styled versions of the testing set (i.e., inputs are transformed).
$M_O$	Original non-trojaned model.
$M_T$	Trojaned model.
$M_{H(X)}$	Healed model retrained with the retraining set $X_R$ . $X_R$ is compound of the healing set $X_H$ only.
$M_{H(X-0)}$	Healed model retrained with the retraining set $X_R$ . $X_R$ comprises the healing set $X_H$ , and its corresponding content images (via Algorithm 1).
$M_{H(X-k)}$	Healed model retrained with the retraining set $X_R$ . $X_R$ is formed by the healing set $X_H$ , its content images (via Algorithm 1), and the styled images generated with the first $k$ style base images in $B$ (via Algorithm 3). E.g., $M_{H(X-3)}$ means the model is retrained with $X_H$ , the content images and the styled images generated with the style bases $b^1$ , $b^2$ , and $b^3$ in $B$ .

performance of models with the three metrics listed in Section 3.5.2, we conducted the experiments with a variation of the original dataset only. This ensured a fair comparison with results obtained in previous work.

**Testing set.** It is formed by 20 random images per class. Two adversarial versions of it are used, one for each trigger.

**Attack strategy.** The two provided models were compromised through the re-training process covered in Section 2.3.3. Row 2 of Table 2.2 shows examples of two adversarial samples with the square and watermark triggers in columns 3 and 4 respectively. The provided models classify any image with either trigger to the target class (0 or A.J. Buckley).

#### 2.5.4 Acronyms Used in Experiments

Table 2.3 lists the acronyms used to refer to the models and data used in the experiments. It also indicates how to identify the testing set used in the evaluation of each model.

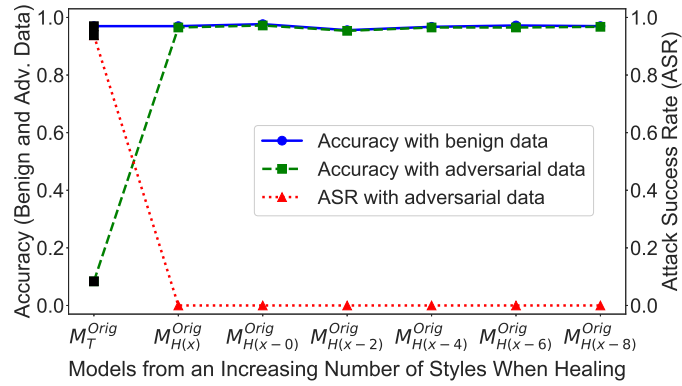
## 2.6 Experiments

This section describes the experiments conducted to evaluate *ConFoc* against trojan attacks. The experiments were designed to answer a series of research questions, included in each of the following subsections along with our findings.

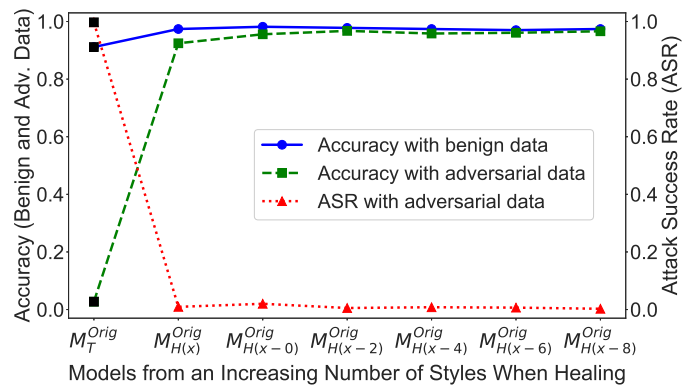
### 2.6.1 Robustness When Processing Original Inputs

**RQ1.** How do the evaluation metrics change as *ConFoc* is progressively applied using an incremental number of styles?

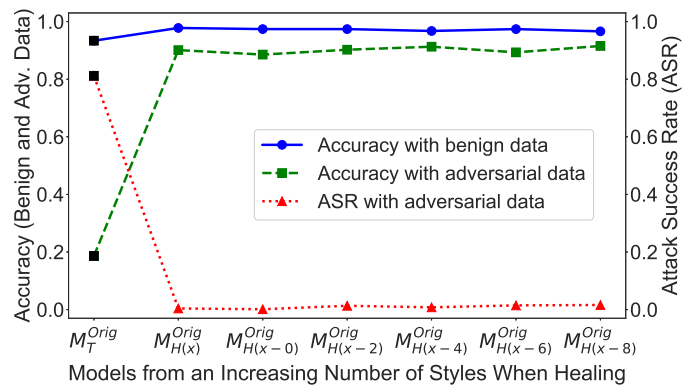
We investigate whether the performance of trojaned models (based on the three metrics described in Section 3.5.2) improve as we increase the number of styles used in the healing process. We conduct our evaluation against BadNets, Trojaning (SQ), and Trojaning (WM) using the corresponding original testing sets. Figure 2.5 shows the results. For each of the attacks, we start with the corresponding trojaned model  $M_T$  and proceed as follows. First,  $M_T$  is evaluated with the original testing samples



(a) BadNets



(b) Trojancing (SQ)



(c) Trojancing (WM)

Fig. 2.5.: Metric variations as *ConFoc* is progressively applied to  $M_T$  by increasing the number styles used in the healing process. Resulting healed models (in  $x$ -axis) are evaluated with the original (non-transformed) testing datasets (refer to Table 2.3).

to measure the performance of the model before applying *ConFoc* ( $M_T^{Orig}$  in  $x$ -axis). The corresponding points in the plots are marked with a black square to highlight that these are the initial values of the metrics. Then,  $M_T$  is taken through the *ConFoc* healing process multiple times using incremental retraining sets to measure how the metrics vary as more styles are used (points  $M_{H(X)}^{Orig}$  to  $M_{H(X-8)}^{Orig}$  in  $x$ -axis).

Figures 2.5a, 2.5b, and 2.5c show that the performance improves as *ConFoc* is progressively applied. The three metrics tend to converge to the aimed values with just a few styles (considering the graphs of all the metrics, two styles suffice for all the cases). For the three attacks, the ASR drops to or close to 0.0%. Simultaneously, the accuracy with benign data converges to high values that outperform the initial accuracy of the trojaned model. This metric has percentage increases of 0.24%, 7.28%, and 3.63% in the best obtained healed models  $M_{H(X-6)}$ ,  $M_{H(X-4)}$ , and  $M_{H(X-4)}$  for the attacks BadNets, Trojaning (SQ), and Trojaning (WM) respectively. For the accuracy with adversarial data, we also obtain a significant increase in these models. This accuracy increases 88.14%, 94.02%, and 72.66% in the models for the same order of attacks. An interesting behavior is observed in the case of Trojaning (WM). The accuracy with adversarial data significantly improves to values above 90% in all the cases, but always remains lower than the accuracy achieved with benign data. This phenomenon can be explained by the fact that the watermark overrides the object of interest (i.e., faces), covering certain key attributes of the faces (e.g., eyes, lips, etc.) used by models during the classification (a violation to the condition *C3* listed in Section 2.3.2). As a consequence, some adversarial inputs with the watermark covering key attributes of the faces cannot be recognized to their true classes after applying *ConFoc* because the resulting contents (face shapes plus watermark) are not part of the content of images present in the healing set  $X_H$ . Note that a violation to condition *C3* means that attackers assume weak defenders who cannot perceive triggers even when they cover a significant portion of the input images (a less real-world feasible scenario from the standpoint of the attacker).



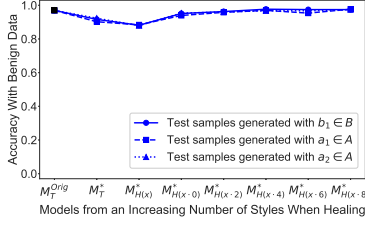
**Findings.** With a few styles (two in our case), *ConFoc* reduces the ASR to or close to 0.00%, while ensures that both accuracies converge to close values equal or above the original accuracy when conditions *C1-C4* are satisfied.

### 2.6.2 Robustness When Processing Transformed Inputs

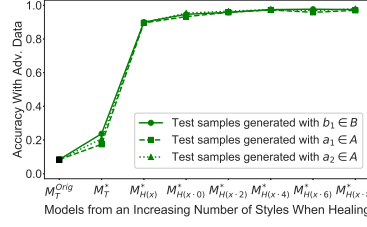
**RQ2.** How well do models learn to focus on content and how effective *ConFoc* is when processing styled inputs?

Following the methodology of the previous section, we now evaluate how well healed models learn to focus on the content of images, disregarding their styles. To this end, models are evaluated using three different styled or transformed versions of the testing set. One version is generated with the style base image  $b^1 \in B$ , which is used in the healing process. The other two versions are obtained using the style base images  $a^1$  and  $a^2$  in  $A$ , which are not used during the healing of the models. For each attack, we start again with the corresponding trojaned model  $M_T$  evaluated with original samples to get the initial values of the metrics before applying *ConFoc* ( $M_T^{Orig}$  in  $x$ -axis). Following,  $M_T$  is tested using transformed samples to measure the impact that the input transformation itself has on the performance ( $M_T^*$  in  $x$ -axis). Finally, transformed samples are used to test the models healed through an incremental application of *ConFoc* (points  $M_{H(X)}^*$  to  $M_{H(X-8)}^*$  in  $x$ -axis).

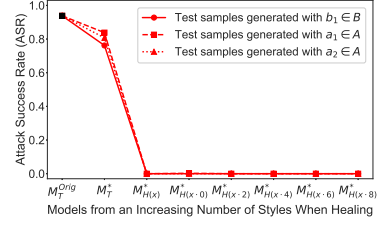
Figure 2.6 shows the results. Each subfigure in it corresponds to one of the metrics and an attack. For all the metrics, the final performance of the healed models are nearly the same, regardless the styled version of the testing set used in the evaluation. The metrics tend to converge to sought values as more styles are used. This is a consequence of the increasing data augmentation achieved through the addition of new styles to the *ConFoc* process. Both accuracies improve because the larger the retraining set is, the more samples with common content information the model receives. With the increasing sets, models are fine-tuned with enough samples for them to extract the contents of the training sample features, which are also present in the testing samples. Simultaneously, the attack success rate also drops because of



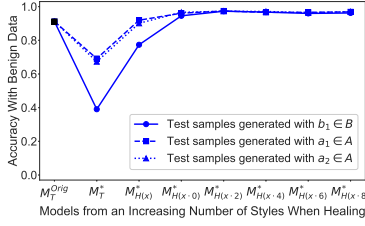
(a) BadNets: Accuracy (Benign Data)



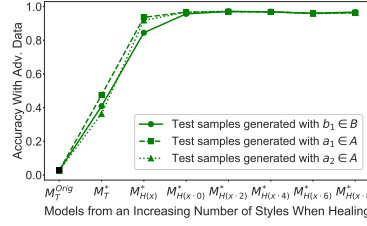
(b) BadNets: Accuracy (Adversarial Data)



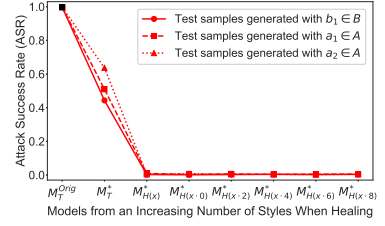
(c) BadNets: ASR (Adversarial Data)



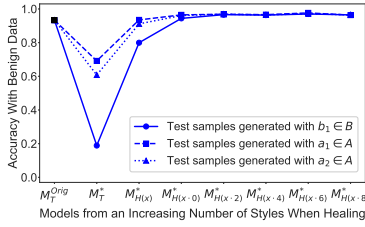
(d) Trojanning (SQ): Accuracy (Benign Data)



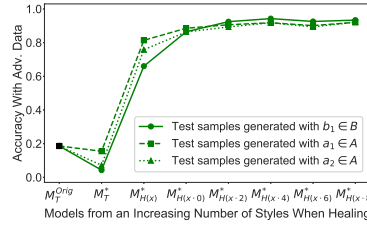
(e) Trojanning (SQ): Accuracy (Adversarial Data)



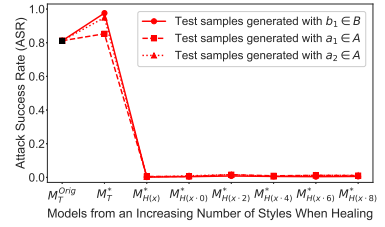
(f) Trojanning (SQ): ASR (Adversarial Data)



(g) Trojanning (WM): Accuracy (Benign Data)



(h) Trojanning (WM): Accuracy (Adversarial Data)



(i) Trojanning (WM): ASR (Adversarial Data)

Fig. 2.6.: Efficiency of *ConFoc* on making models focus on content at testing.  $M_T$  is healed with an incremental number of styles. Resulting healed models are evaluated with test sets generated with different style bases:  $b^1 \in B$ ,  $a^1 \in A$ , and  $a^2 \in A$ .

this data augmentation. As the retraining set increases, models tend to forget the trigger because more parameter updates are executed in one epoch of training with samples not including the trigger. This is an expected behavior based on the findings of Liu et al. [14], who shows that this metric decreases as more benign data samples (original version only) are used to fine-tune DNN models.

Notice that the transformed testing datasets used in this evaluation are generated with both styles used and not used in the healing process. Hence, this experiment shows the effectiveness of *ConFoc* on making models focus on content and not on styles during the classification. One interesting observation is that using styled images without healing the models does not prevent the attacks. The attacks become ineffective after applying *ConFoc* with a few styles. Considering all the plots and metrics in Figure 2.6, four styles suffice.

After *ConFoc*, the ASR is reduced to or close to 0.0%. In all the attacks, the accuracies with benign data (regardless the style) achieve high values that outperform the initial accuracies of the trojaned model. Using the best resulting healed models  $M_{H(X-6)}$ ,  $M_{H(X-4)}$  and  $M_{H(X-4)}$  for the attacks BadNets, Trojaning (SQ), and Trojaning (WM) respectively, this metric grows 0.47%, 6.71%, and 3.2% when evaluated with the transformed testing set generated with  $b_1 \in B$ . With respect to the accuracy with adversarial data, the metric increases 89.30%, 94.41%, and 75.65% with the same healed models.

**Findings.** With a few styles (four in our case), *ConFoc* reduces the ASR to or close to 0.00%, while ensures both accuracies get values equal or above the original accuracy regardless the input style when conditions *C1-C4* hold.

### 2.6.3 Effect on Non-Trojaned Models

**RQ3.** What is the impact of *ConFoc* on the accuracy (only benign data applies) of non-trojaned models?

One of the main challenges defending against trojan attacks is the lack of tools to determine whether a given model has a trojan. Due to this restriction, this section evaluates the impact *ConFoc* has on the accuracy of an original non-trojaned model  $M_O$ . Our goal is determining whether *ConFoc* can be applied to any model (whether infected or not) without impairing its current performance (accuracy with benign data).

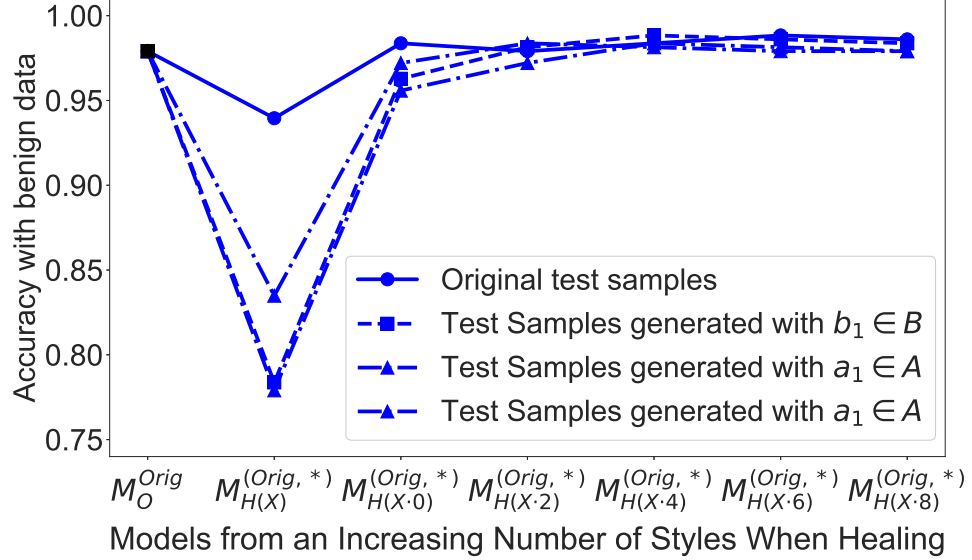


Fig. 2.7.: Accuracy (benign data) variation of the GTSRB-based non-trojaned model when *ConFoc* is progressively applied.

We take the non-trojaned version of the models created with the datasets GTSRB and VGG-Face through the *ConFoc* healing process. Figure 2.7 shows the metric variation of the GTSRB model as styles are added to the healing process. Taking model  $M_{H(X-4)}$  tested with the transformed samples generated with  $b_1 \in B$  as example, the accuracy improves from 97.91% to 98.37%. We get a similar graph (not included) with the VGG-Face model. In this case, the best performance is obtained with the model  $M_{H(X-6)}$ , with a percentage increase of 0.17%. These results prove that *ConFoc* does not affect the accuracy of non-trojaned models. In contrast, the trends of the graphs shows that it at least remains the same if enough styles are used in the healing process.

**Findings.** *ConFoc* can be equally applied to any model (either trojaned or not) as it does not impair its performance.

Table 2.4.: Best Healed Models After Applying ConFoc

Experimental Setup		Best Healed Model	
Attack	DS	Model ID	No. Styles (Including Content)
BadNets	10%	$M_{H(X-6)}$	7
Trojaning (SQ)	5%	$M_{H(X-2)}$	3
Trojaning (WM)	5%	$M_{H(X-4)}$	5
BadNets	6.6%	$M_{H(X-4)}$	5
Trojaning (SQ)	3.3%	$M_{H(X-2)}$	3
Trojaning (WM)	3.3%	$M_{H(X-1)}$	2
BadNets	3.3%	$M_{H(X-4)}$	5
Trojaning (SQ)	1.67%	$M_{H(X-1)}$	2
Trojaning (WM)	1.67%	$M_{H(X-2)}$	3

#### 2.6.4 Healing Set Size and Number of Styles

**RQ4.** Does the number of styles required in the *ConFoc* healing process depend on the size of the healing set  $X_H$ ?

We investigate the relationship between the number of styles required to successfully apply *ConFoc* and the size of the healing set. This is a key question because having access to extra training sets is challenging in real-world scenarios. As specified in Section 3.5, previous experiments are run with healing sets of size 10% and 5% for the models infected with BandNets and Trojaning Attack respectively.

We now replicate the same experiments progressively decreasing the size of these sets and selecting the model with the best performance in each case. Table 2.4 shows that there is no relationship between size of the healing set and the number of styles needed to apply *ConFoc*. This can be explained because the combination of some contents and styles add noise to the resulting retraining set, which make models to

not monotonically improve as more styles are added. Whereby, defenders need to apply the best training practices to fine-tune the models with the generated data so as to obtain the best possible results.

**Findings.** There is no relationship between the  $X_H$  size and the number of styles needed to successfully apply *ConFoc*.

### 2.6.5 Performance and Comparison With the State-of-the-Art

**RQ5.** How well does *ConFoc* perform compared to the state-of-the-art and what overhead it imposes at testing?

Table 2.6 shows the performance of *ConFoc* and its comparison with the state-of-art Neural Cleanse [1]. To be complete in our comparison with fine-tuning-based methods, we also include a comparison with Retraining [14]. The table contains the accuracies (with both begin and adversarial data) and the ASR after applying the defensive methods. The first column specifies the attack used for the evaluation. DS refers to the size of the healing set. The initial values of the trojaned models (before applying any method) are included in Table 2.5.

In Table 2.6, columns below ConFoc (Original Inputs) and ConFoc (Transformed Inputs) summarize the performance of our method using the best healed models (specified in Table 2.4) for different sizes of the healing set. As the names indicate, we tested *ConFoc* with both original and transformed inputs. The other two techniques (Retraining and Neural Cleanse) were evaluated with original inputs as the methods require. With Retraining, we fine-tuned the model using the original healing set for multiple epochs and selected the best resulting model. In the case of Neural Cleanse, we proceeded exactly as indicated by the authors in [1]. We added the reversed-engineered triggers to 20% of the samples in the healing set and retrained the model for one epoch. The reversed-engineered triggers are provided by the authors in [57]. During the execution of this method, only the trigger related to Trojaning (WM) worked as expected. Whereby, we ran the Neural Cleanse method against BadNets

Table 2.5.: Initial Metrics of Trojaned Models

Attack	Acc (Ben)	Acc (Adv)	ASR
BadNets	96.98%	8.37%	93.81%
Trojaning (SQ)	91.15%	2.73%	99.73%
Trojaning (WM)	93.36%	18.62%	81.18%

and Trojaning (SQ) with the actual triggers used to conduct the attacks. This action does affect the performance of Neural Cleanse. In contrast, it represents the ideal scenario in which triggers are perfectly reverse-engineered and the produced corrected models provide the best possible results.

As shown in the Table 2.6, all the defensive methods produce high accuracy with benign data regardless the size of the healing set. In most cases, this metric is superior to the initial value of the trojaned model (see to Table 2.5). The main differences between the methods are observed in the accuracy with adversarial data (highlighted in light grey for all the methods) and the ASR. *ConFoc* (with both original and transformed inputs) constantly gets high values in these two metrics, while the other methods produce values below 90% for the former and above 1% for the latter as the healing set decreases. These cases are marked in red in the table.

With respect to the accuracy with adversarial data, Retraining, as expected, tends to produce models with lower values in this metric as the healing sets become smaller in all the attacks [14]. Neural Cleanse produces models that perform well against both Trojaning Attacks and unwell against BadNets regardless the size of the healing set. This is because Neural Cleanse relies on updating the model parameters for one epoch only, which does not suffice to remove the learned trigger-related features. BadNets is conducted via poisoning, which means that the parameters of all model layers are adjusted during training. Whereby, to remove the effect of triggers, larger datasets or more epochs are required [14]. Trojaning Attack, in contrast, is a retraining technique

that fine-tunes the last layers of the models (i.e., it changes less parameters) while inserting the trojan (see Section 2.3.3). Hence, one epoch is enough to remove the trigger effect.

At this point, it is important to highlight that due to the violation of the condition C3 as explained Section 3.6.1, *ConFoc* produces models with lower values in the accuracy with adversarial data than those obtained with benign data in the case of Trojaning Attack (WM) (see dark grey cells in the table). These values, however are constantly above 90% and do not tend to decrease with the sizes of the healing set.

***ConFoc Overhead.*** There is no clear advantage (with respect to the metrics) on using either original or transformed inputs with *ConFoc*. However, there is a difference in the overhead caused at testing time. Transforming the inputs with Algorithm 3 directly imposes an 10-run average overhead of 3.14 s with 10 iterations of the optimizer LGBFS [54] over a Titan XP GPU. We reduce this runtime overhead to values around 0.015 s by applying the principles of Algorithm 1 and Algorithm 3 to train image transformation neural networks offline for each chosen style as proposed in [58]. This implementation is included in our prototype [25]. *ConFoc* does not impose any overhead at testing if original inputs are used.

**Findings.** *ConFoc* outperforms the state-of-the-art method regardless the size of the healing set, without imposing any overhead when original inputs are used in the evaluation.

### 2.6.6 Robustness Against Adaptive and Complex Triggers

**RQ6.** How effective is ConFoc protecting DNN models when adaptive and complex triggers are used?

This section evaluates *ConFoc* against trojan attacks conducted with complex triggers. We conduct the attacks with BadNets because this approach extracts trigger-related features in all the layers of the model, making it more difficult to eliminate. The idea is to test of *ConFoc* in the most complex scenarios. We make sure that the



Table 2.6.: Metrics of Corrected Models After Applying ConFoc and Other State-of-the-Art Model Hardening Techniques

Attack	Experimental Setup	ConFoc (Original Inputs)			ConFoc (Transformed Inputs)			Retraining (Healing Set Only)			Neural Cleanse		
		DS	Acc (Ben)	Acc (Adv)	ASR	Acc (Ben)	Acc (Adv)	ASR	Acc (Ben)	Acc (Adv)	ASR	Acc (Ben)	Acc (Adv)
BadNets	10.0%	97.21%	96.51%	0.00%	97.44%	97.67%	0.00%	96.98%	96.51%	0.00%	97.21%	65.35%	0.00%
	5.0%	97.79%	96.75%	0.53%	97.27%	97.14%	0.27%	97.40%	92.45%	0.94%	97.53%	97.27%	0.27%
	5.0%	96.75%	91.28%	0.80%	96.35%	94.27%	0.53%	97.79%	90.10%	0.40%	96.88%	93.36%	0.27%
BadNets	6.66%	97.21%	96.98%	0.00%	97.21%	97.21%	0.00%	96.28%	94.42%	0.00%	97.21%	54.42%	0.00%
	3.33%	97.40%	97.27%	0.53%	96.75%	97.14%	0.66%	98.31%	82.68%	0.16%	97.53%	97.79%	0.13%
	3.33%	96.88%	92.32%	0.13%	96.09%	91.67%	0.27%	98.05%	92.19%	1.73%	97.01%	92.45%	0.00%
BadNets	3.33%	96.05%	96.05%	0.00%	97.21%	97.21%	0.00%	95.12%	96.05%	0.00%	97.21%,	58.84%	3.33%
	1.67%	98.05%	96.09%	0.67%	96.35%	96.88%	0.27%	98.05%	82.94%	0.16%	96.61%	96.09%	0.00%
	1.67%	97.27%	91.15%	0.13%	95.96%	92.84%	0.27%	97.66%	83.72%	9.08%	96.48%	89.58%	1.34%

attacks comply with the conditions  $C1-C4$  specified in Section 2.3.2. The complex triggers are described below using as reference the trigger (referred here to as original) and data split presented in Section 2.5.2 (see Figure 2.4). In the description, the sizes of the triggers correspond to a percentage of the larger side of the inputs.

- **Adaptive.** We assume an adaptive attacker knowledgeable about *ConFoc* who seek to mitigate the healing procedure by infecting the model with styled adversarial samples. The original trigger is added to the samples in the trojan set (trj). These samples are then transformed via *ConFoc* using the base  $b_1 \in B$ , which is used in the healing process enacting so the best scenario for the attacker. The target class is 19.
- **Larger.** A white square of size is 15% (rather than the 10% size of original) located in the botton-right corner of the image. The target class is 19.
- **Random Pixel.** A square of size 10% located in botton-right corner of the image, whose pixel values are randomly chosen. The target class is 19.
- **Multiple Marks.** A trigger consisting of four marks: (1) the original white square in the botton-right corner, (2) the random pixel square described above located in the botton-left corner, (3) a white circle (circumscribed by a square of size 15%) located in top-left corner, and (4) the same circle but filled with random pixels located in the top-right corner. The target class is 19.
- **Many-to-One.** Each of the multiple marks described above are added individually to the samples in the torjan set (trj). Namely, we create four trojan sets, each with one of the marks. The target class assigned to all the resulting adversarial samples in these sets is 19.
- **Many-to-Many.** In this case we assign a different target class to each of trojan sets described above. The assignment is as follows: (1) botton-right mark targets class 19, (2) botton-left mark targets class 20, (3) top-right mark targets class 21, and (4) top-left mark targets class 22.

Table 2.7.: Performance With Adaptive/Complex Triggers

Trigger	Before ConFoc			After ConFoc		
	Acc (Ben)	Acc (Adv)	ASR	Acc (Ben)	Acc (Adv)	ASR
Adaptive	98.14%	2.33%	100.0%	98.14%	97.91%	0.00%
Larger	97.67%	2.56%	99.76%	97.67%	97.91%	0.00%
Random Pixel	97.91%	2.33%	100.0%	98.14%	97.44%	0.00%
Multiple Marks	97.44%	2.33%	100.0%	97.67%	97.91%	0.00%
Many-to-One	96.51%	20.93%	80.48%	97.44%	97.21%	0.00%
Many-to-Many	97.91%	21.63%	80.00%	97.91%	98.14%	0.00%

Table 2.7 shows the metric of the trojaned models before and after applying *ConFoc*. Results show that *ConFoc* effectively reduces the ASR to the minimum while ensures both accuracies remain close or better than the initial values.

**Findings.** *ConFoc* effectively eliminate trojans on DNNs compromised with complex triggers, while ensures accuracy values that in average either equal or outperform the initial values of the model when conditions *C1-C4* are satisfied.

## 2.7 Conclusions and Future Work

We present a generic model hardening technique called *ConFoc* to protect DNNs against trojan attacks. *ConFoc* takes as input an infected model and produces a healed version of it. These models are healed by fine-tuning them with a small dataset of benign inputs augmented with styles extracted from a few random images. We run experiments on different models and datasets, infected with a variety of triggers by two different trojan attacks: BadNets and Trojaning Attack. Results show that *ConFoc* increasingly reduces the sensitivity of trojaned models to triggers as more styles are used in the healing process. We proved that our method can be equally applied to any model (trojaned or not) since it does not impact the initial accuracy of the model. In

comparison with the state-of-the-art, we validate that *ConFoc* consistently correct infected models, regardless the dataset, architecture or attack variation. Our results leads us to new research questions related to the internal behavior of models. Future work will aim to investigate which neurons relate to the content of inputs. This information will be used to devise a novel white-box approach to detect misbehaviors based on the activation of these neurons.

### 3. HUNTING FOR INSIDER THREATS USING LSTM-BASED ANOMALY DETECTION

#### 3.1 Introduction

A demanding challenge for security systems is to successfully defend against insider threats because insiders are in possession of credentials, have (some) knowledge of the system operation, and are implicitly trusted as members of the organization [59]. They are also located inside the security perimeter, allowing them to unsuspectingly deploy attacks such as data exfiltration, tampering with data, and deletion of critical data [60,61]. They commonly use sophisticated strategies to avoid detection like those in multistage persistent threats [62] and mimicry attack [63–65]. Namely, insiders mix malicious event sequences with benign actions to exploit the incapacity of defensive systems to discern event sequences after certain length, which is referred to as the *order-aware recognition* (OAR) problem [66]. Existing enterprise protection systems endeavor to counter this increased sophistication in insider evasion attacks through the application of anomaly detection methods based on advanced machine learning. Machines in customer companies run Endpoint Detection and Response (EDR) agents that generate high volumes of system events that are examined through centralized analytics modules running at the security-provider company. The ultimate goal is to detect stealthy threats, including zero-day exploits, by analyzing patterns and relationships of the aggregated data collected from these multiple endpoints at runtime. In current enterprise solutions, many of the collected malicious events are correctly classified as alerts. However, others are ignored and considered benign events despite being part of the attacks that span for a long period of time. These undetected malicious events are usually related to those detected and identified as alerts, but they are missed because of the lack of optimal solutions able to find the existing

relationships among distant events in a sequence. This brings the need for precise system behavior modeling capable of capturing long-range relationships (i.e., long term dependencies) in multiple context for event sequence analysis and detection of anomalies at runtime [67].

The paradigm of anomaly detection [66, 68–74] involves the construction of patterns of normal behavior of systems and deems as anomalous (or possible intrusion) any action that does not conform to the learned patterns [16, 75]. Prior research work have been devoted to investigate and develop anomaly detection systems using sequence analysis strategies. Some of these detection techniques are based on  $n$ -gram [75–77] and others on Hidden Markov Model (HMM) [67, 75, 78–82]. In general, these techniques learn observed patterns in a training phase and identify as anomalous event sequences that deviate from them during testing. In particular, HMM-based methods estimate the likelihood of events conditioned on some number of previous events (e.g., after observing  $n - 1$  previous events). This allows determining not only whether a sequence of certain length (i.e.,  $n$  in this case) is feasible to occur, but also how likely it occurs in normal (non-attack) conditions. However, Yao et al. [66] presented a comprehensive analysis of these techniques and showed they are incapable to discern the order of events in long sequences due to the OAR problem, restricting the length  $n$  of the analyzed sequences to small values.

In this paper, we present a LSTM-based anomaly detection framework that collects and analyzes high volumes of system events from multiple distributed EDR agents to protect against insider threats at runtime. We refer to the framework as LADOHD (LSTM-based Anomaly Detector Over High-dimensional Data) due to the high feature dimensionality of the produced events. LADOHD tackles the OAR problem by leveraging the event relationship information extracted from different endpoints as well as the properties ingrained to LSTMs and its variants [17, 83], such as memory, short and long term dependencies, stateful representation, and capacity to process variable length sequences [84, 85]. We hypothesize that these properties give these models the ability to detect variable-length anomalous sequences and the

potential to recognize attacks deployed by insiders that span for a long time. Specifically, our LSTM-based technique answers the anomaly detection problem of given a sequence of events  $e_1, e_2, \dots, e_{n-1}$ , whether or not the sequence  $e_1, e_2, \dots, e_{n-1}, e_n$  should occur. Our technique operates with variable values of  $n$  and detects non-conforming patterns with respect to the learned models by analyzing the event sequences formed by system activities. Each possible system activity is enumerated and uniquely identified to form the vocabulary of system events. At any time  $t$ , our detector computes the probability of each possible event to be the next one given the previous sequence of events observed until time  $t-1$ . The detection is then made by analyzing the distribution of these probability values.

The obtained results include quantitative measurements of the detection capacity of the proposed technique tested over a dataset of 38.9 million activity events. These events were collected from multiple security endpoints running on more than 30 machines for 28 days. It is shown through different experiments that our framework successfully achieve detection with a TPR and a FPR of 97.29% and 0.38% respectively. Below, our research contributions:

- We implement a prototype [86] tested with a dataset of 38.9 million activity events collected from an enterprise EDR system. The attack detection capacity of our method is compared with the EDR of same company. Results show that our method achieves an increase in the detection rate of while keeping an FPR  $< 0.5\%$ .
- A novel approach to define the vocabulary of events for the data collected from multiple enterprise EDR agents is introduced. Event features are carefully analyzed and selected so that long-term dependencies among the different events are successfully learned.
- We measure how far LSTM-based models look backward to rank probable events in each timestep of a sequence. We demonstrate that LSTMs have a better

capacity than alternative methods (e.g., HMM-based methods) to solve the OAR problem.

- To the best of our knowledge, we are the first presenting a comprehensive analysis of the strengths, limitations and applicability of LSTM-based models to counter insider threats through the detection of anomalies in real-word scenarios.

## 3.2 Overview and Threat Model

### 3.2.1 Overview

LADOHD builds LSTM-based behavioral profiles of applications using the system event sequences collected from multiple endpoints running a renowned EDR agent. Its goal is to detect anomalous or non-conforming execution patterns at runtime in two phases. First, a training or observation phase, in which the profile of a selected application is built by learning the relationships among events in patterns or sequences observed when the application runs in normal (non-attack) conditions. Second, a testing or evaluation phase, in which the learned model is used to estimate the probability of each possible event to be the next event in a sequence given the sequence of previous events. In the latter phase, low probable events are classified as anomalous.

Like some previous LSTM-related work [87–92], We assume that the generated event sequences follow a well-structured pattern (e.g. execution path of programs) with a consistent relationship among events. Consequently, the resulting sequences are thought as an structured language that can be analyzed using LSTM-based models as it has been done via Natural Language Processing (NLP) to solve problems such as language modeling (i.e., prediction of the next word in a text) [93, 94].

LADOHD requires the definition of a finite set of possible symbols  $E = \{1, 2, \dots, N\}$ , which corresponds to all the possible events related to the application of interest that are considered in the detection process (hereafter, we will refer to this set as vo-



cabulary of events). At training, LADOHD extracts all the subsequences containing the events in  $E$  from the set of event sequences  $S = \{s_1, s_2, \dots, s_N\}$  generated by  $N$  endpoints. These subsequences are used to train the LSTM-Based model.

The definition of the vocabulary of events  $E$  is crucial because there is a trade-off between the granularity of the events and the number of unseen events that appear in the evaluation or testing phase. For our experiments, we defined  $E$  in such a way that most of the events observed at training are also observed at testing, reducing the number of unseen events during the evaluation phase. Section 3.4 includes the details of our definition.

During the evaluation phase, given a previous sequence of events until timestep  $t - 1$   $e_1, e_2, \dots, e_{t-1}$  ( $e_i \in E$ ), the trained model outputs an array of probabilities of length  $|E|$ , representing the probabilistic estimation of each event in  $E$  to be the next event at timestep  $t$ . For the detection, LADOHD uses this output and finds the set  $K$  of the top  $k$  most likely events to occur at time  $t$ . When an event  $e_t \in E$  is observed at time  $t$ , it is considered benign if  $e_t \in K$ , anomalous otherwise.

For any sequence  $s = e_1 e_2 \dots e_{t-1}$ , our framework computes the probabilities of possible events next in the sequence  $P(e_i | e_{1:i-1})$  for  $i = 1, 2, \dots$ . This versatile approach allows not only validating each event at runtime, but also estimating the probability of the entire sequence  $s$  by applying the chain rule as shown in Equation 3.1.

$$P(s) = \prod_{i=2}^t P(e_i | e_{1:i-1}) \quad (3.1)$$

LADOHD operates with system events collected from multiple monitored machines. The EDR agent running in these machines generates an event for every activity conducted by a specified process (whether malicious or not). Each event includes a comprehensive set of information about the *actor* (process executing the action), detailed description of the *action*, and information about the *target* (object over which the action is executed). The pieces of information considered during the monitoring process and their interpretation define the vocabulary of events and its granularity. For example, consider the scenario where a “process A (actor) connects (action) to

specific IPv4 address X.X.X.X (target).” This event might be defined as “A connects X.X.X.X”, where X.X.X.X represents any possible IPv4 address, producing a vocabulary with high granularity. The same event, however, might be defined as “A connects X”, with X being either 0 or 1 to represent whether the IPv4 address is internal or external respectively. In the latter case, due to the low granularity, the vocabulary size is significantly reduced.

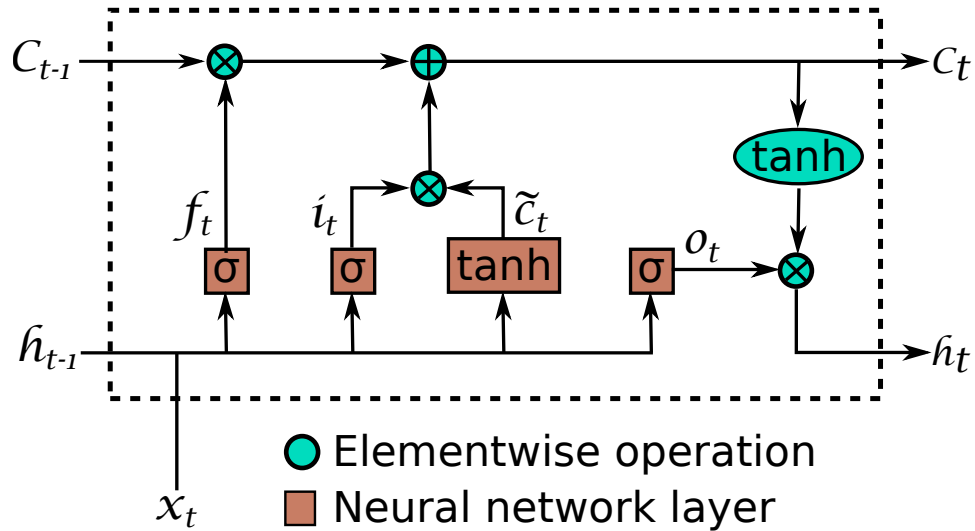
### 3.2.2 Threat Model

We consider an insider threat who launches a multistage advance persistent attack. The insider is assumed knowledgeable in computer security and is initially assigned non-administrative privileges in a local machine. The goal of the attacker is stealing information by executing multiple steps, including a user escalation followed by a data exfiltration phase. The insider initially exploits already installed applications such as Powershell and runs malicious scripts to establish remote connections to send the stolen data.

## 3.3 Background and Related Work

### 3.3.1 LSTM Networks

LSTMs are a type of recurrent neural network (RNN) able to learn long-term dependencies (i.e., relationships between elements in distant positions of a sequence) [95]. They achieve this goal through a complex memory structure in the LSTM cell not included in traditional RNN cells. Figure 3.1 shows this structure. The matrices and vectors  $W_x$ ,  $U_x$ , and  $b_x$  (with  $x \in \{f, i, c, o\}$ ) in Figure 3.1b are the parameters  $\theta$  of a LSTM [95]. Their interaction is shown in Figure 3.1a. Like traditional RNNs, LSTMs process each input at time  $t$  along with the output of the previous timestep ( $h_{t-1}$ ). In addition, they include a unit called cell state ( $C$ ) that carries on information of the entire sequence. LSTMs adds to or removes minor pieces of information from  $C$



(a) LSTM Cell

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$$

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f)$$

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o)$$

$$\tilde{c}_t = \tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot \tanh(C_t)$$

(b) Operations in the LSTM Cell

Fig. 3.1.: Architecture of a LSTM cell. Figure 3.1a shows the internal connections of the different components and the output of the internal operations of the cell. Figure 3.1b shows the details of these operations performed over the current input and previous hidden state to compute the current output.

through the operations in the forget ( $f_t$ ) and input ( $i_t$ ) gates. The new  $C$  represents the event history used to compute the output  $h_t$  by filtering  $C$  out with the output gate  $o_t$ . This architecture gives LSTMs the capacity to relate current events with distant past events in a sequence, making them suitable to detect anomalies produced by insider threat activities.

LSTMs can be implemented as multi-class classifiers that map a  $m$ -dimensional input symbol  $x \in \mathbb{R}^m$  into one of  $n$  classes (each class corresponding to one of the possible events). The output of a LSTM (with a *softmax* [96] layer at the end) is a  $n$ -dimensional tensor  $y \in \mathbb{R}^n$ , which represents the probability distribution of the  $n$  classes. Namely, the element  $y_i$  of the output  $y$  represents the probability that input  $x$  corresponds to the class  $i$ . To operate as a sequential multi-class classifier, LSTMs are trained using backpropagation [35] with a set of pairs  $(x, y)$ , where  $x$  is an input sequence of classes and  $y$  the expected next class of the sequence  $x$ . The training pair  $(x, y)$  is customized to control the timestep windows ( $w$ ) used to update the parameters  $\theta$ . For example, given a input sequence  $x = x_1, x_2, \dots, x_{t-1}$ , the network can be trained to predict either the element  $x_t$  ( $w = t - 1$ ) or each of the elements  $x_2, \dots, x_{t-1}, x_t$  ( $w = 1$ ). When  $w = 1$  (our case as described in Section 3.4.4), the LSTM outputs the probability  $P(x_t|x_{1:t-1})$  at each timestep  $t$ , which allows classifying low probable events as anomalous regardless the length of the previous sequence.

### 3.3.2 Order-Aware Recognition (OAR) Problem

The OAR problem is an anomaly detection problem that refers to the incapacity of distinguishing sequences after certain length [66]. Given a ordered sequence of events  $abcba$  the corresponding set of 2-tuple adjacent events is  $\{ab, bc, cb, ba\}$ . The same set results from these other two ordered sequences  $cbabc$  and  $bcbab$ . As the 2-tuple adjacent event set is the same for these three ordered sequences of the example, methods able to analyze sequences of length 2 or less cannot discern among these ordered sequences. This can be better observed if the 3-tuple adjacent events

of the sequences  $abcba$ ,  $cbabc$  and  $bcbab$  are considered, which respectively are  $\{abc, bcb, cba\}$ ,  $\{cba, bab, abc\}$  and  $\{bcb, cba, bab\}$ . Clearly, in this case methods able to analyze sequences of length 3 can distinguish the three ordered sequences  $abcba$ ,  $cbabc$  and  $bcbab$  as the resulting sets are different. We investigate how feasible and until what extend LSTM-based models can solve the OAR problem. This is an unsolved question and one of our main contributions.

### 3.3.3 Endpoint Detection and Response

Endpoint Detection and Response (EDR) systems work by monitoring endpoint and network activity and storing the corresponding logs in a central database where further analysis and alerting takes place. An EDR agent is installed in each of the protected endpoints, acting as the first line of defense against attacks and providing the foundation for event monitoring and reporting across the network. EDR systems evolved from malware protection solutions, as software vendors added data collection and exploration capabilities, thanks to the increasing computing and storage capacity of the hosts where the agents run. The present challenge for EDR systems is to significantly increase its detection capabilities from the vast amounts of data collected, especially for attacks that are recorded as long sequences like those deployed by insider threats.

### 3.3.4 Anomaly Detection Based on Sequence Analysis Using Non-LSTM approaches

The methods presented in this section proposed sequence analysis as an anomaly detection mechanism to detect control-flow violations. The methods build behavioral models based on  $n$ -gram and  $n$ -order HMM to detect unseen or low probable patterns.

Anomaly detection methods based on  $n$ -gram [76, 77] work by enumerating all observed sequences of length  $n$  ( $n$ -grams) to subsequently monitor for unknown patterns. The scalability problem of these methods (impossibility of listing all possible

sequences and high false positive rate) is described by Warrender et al. [75], who proposed an alternative frequency based method. In this new method each  $n$ -gram is assigned a probability to form a histogram vector corresponding to a point in a multidimensional space. At evaluation time, the similarity of a new sequence of length  $n$  (represented as a vector) with respect to the observed points is estimated to determine whether the sequence is anomalous. Despite its improvement in scalability, this approach and the previous enumerating based method were proved to be effective for small value of  $n$  only (e.g., 3–15), making them not convenient for the detection of attacks consisting of long sequences [66].

Other previous work [67, 78, 79] focused on the application of  $n$ -order HMM to probabilistically determine how feasible a sequence of system events is. In [78], a comparison of different hidden states configuration of first-order HMM ( $n = 1$ ) for anomaly detection is presented. It was found that both configurations full connected HMM (i.e., number of hidden states equal to the number of all possible events), and a left-to-right HMM (i.e., number of hidden states corresponds to the length of the training sequences) provide similar results differing mainly in the required training time. Results, although, show the efficiency of both configurations is significantly low having in some cases a TPR of only 55.6%. The other two HMM based methods [67, 79] use a first-order full connected HMM to detect anomalous sequences of system or library calls. These methods are similar to the one described in [78], with the addition of a new HMM initialization approach for the transition, emission and initial probabilities. The information for the initialization is extracted through static analysis of the programs. With this strategy, the results show a significant improvement in the TPR. All the described HMM based methods [67, 78, 79] applied the dynamic programming algorithm Viterbi [97] for inference. The time complexity of this algorithm is  $O(|S|^2)$ , with  $S$  being the set of hidden states [98]. As the lengths of the sequences to be processed by these methods depend on the number of states used in the configuration, this scalability issue restricts these methods to operate over short event sequences only.

### 3.3.5 Anomaly Detection Based on Sequence Analysis Using LSTM

Some research work have endeavored to investigate the application of LSTM-based models to anomaly detection and similar security problems [89–92]. In essence, these approaches work based on the same assumptions described in Section 3.2.1. Although this prior work proved the efficiency of LSTMs to accurately estimate the likelihood of a given event sequence, their ability to solve the order-aware recognition problem and their potential against modern evasion attacks seems not to have received much attention.

Kim et al. [89] present an ensemble method of LSTM models followed by threshold-based classifiers for intrusion detection in flows of system calls. The resulting ensemble is trained in a supervised manner (with both benign and malicious sequences) to classify sequences as either normal or anomalous. Obtained results are compared with other classifiers such as k-nearest neighbor (kNN) and k-means clustering (kMC). Details of neither the impact of sequence lengths nor properties of LSTM models on the detection process are included.

In [90] a multi-level approach for anomaly detection in Industrial Control Systems (ICS) is proposed. It consists of a bloom filter to discard events not seen during the training phase followed by a LSTM layer to detect unexpected events. An event is considered unexpected or anomalous if its probability is not among the top- $k$  output probabilities of the model ( $k$  being an adjustable parameter). The LSTM layer of the detector is trained with non-malicious data only. Results of the two layers combined are reported without further analysis about the LSTM model itself and its impact on the efficiency of the detector.

Du et al. [91] developed Deeplog, a technique to find anomalies using information available in system logs. To that end, a LSTM model is trained using a small portion of the non-malicious data to determine the next action to occur given a previous sequence of actions. Actions are conducted using different parameter values in each occurrence (e.g. files, execution time, etc.). For each identified action, a different

Table 3.1.: Comparison With Existing LSTM-based Security Solutions

Research	Anomaly Detection	Benign Data Only	LSTM Only	Basic Analysis	Extended Analysis
System Call Language Modeling [89]	✗	✗	✗	✗	✗
Multi-level Detector (For ICS) [90]	✓	✓	✗	✗	✗
Deeplog [91]	✓	✓	✓	✗	✗
Tiresias [92]	✗	✗	✓	✓	✗
LADOHD [this work]	✓	✓	✓	✗	✓



LSTM model is trained using the sequence of observed parameter values. The goal is using these multiple models for not only determining the set of expected actions to occur, but also validating the probability of the parameter value used in that action. The model for prediction of actions is trained using a sliding length window  $h$ . Namely, given a sequence of  $h$  actions, the model predicts the  $h + 1$  action. The window moves forward one step at a time during training. At testing, the probability of each sliding  $h + 1$  action in the tested sequence is estimated. The sequence is considered anomalous if there exists at least one action in it whose probability is not among the top- $k$  most likely output probabilities of the model. The experiments were conducted with small values of  $h$  (e.g. 10), and were not focused on determining the limits of LSTM models with respect to the length of the sequences.

A more recent work, called TIRESIAS [92], uses LSTM-based models for attack intent prediction. That is, the technique predicts the next step in an attack given the previous sequence steps. The dataset used in this research comprises events generated by security agents installed in thousands of machines. The sequences generated by 80% of the machines are used for training while the other 20% for validation and testing purposes. The experiments show the capacity of the model to predict the last event of a given sequence only. Although no detection of attacks or malicious sequences are included in this work, interesting results are presented with respect to the behavior of LSTM models, which is one of the main objectives of this paper.

Table 3.1 presents a summary of the focus and details found in the prior work discussed above [89–92] for comparison purposes with our research. Column Anomaly Detection indicates whether the developed technique is an anomaly detection approach, column Benign Data Only whether the used model is trained with benign data only, column LSTM Only specifies whether the solution is based on LSTM models only or integrates other algorithms in it, and finally columns Basic Analysis and Extended Analysis indicate whether the research study includes an initial or a more comprehensive study respectively about the detection performance and limitations of LSTM-based models with respect to the length of processed sequences. As shown in

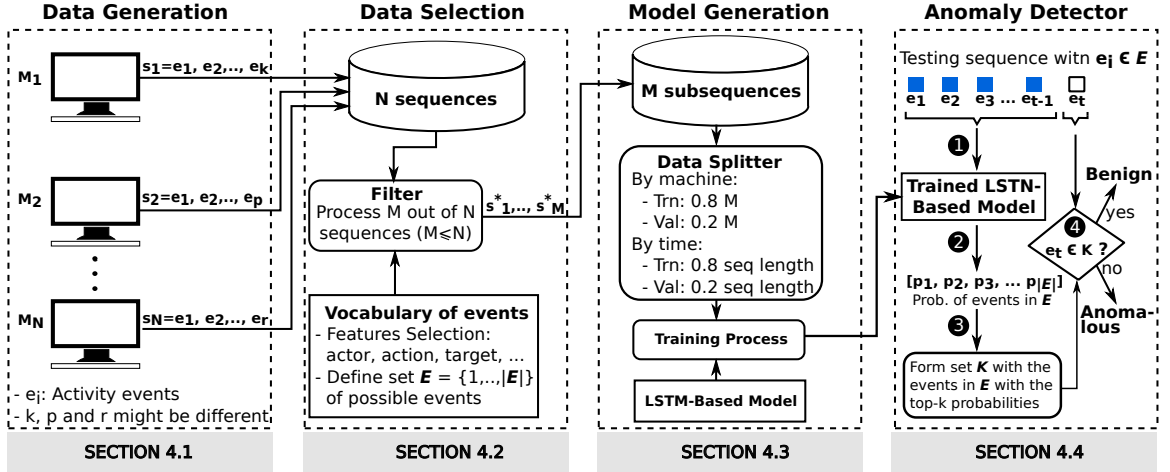


Fig. 3.2.: Components of our anomaly detection framework LADOHD to counter insider threats: (1) data generation, (2) data selection, (3) model generation, and (4) anomaly detector. Below each component, there is a reference to the section providing a detailed explanation about its operation.

the table, our work introduces a LSTM-based anomaly detection mechanism trained with benign data only. No additional algorithms are used for the detection and a full analysis of the properties and limitations of the model is presented.

### 3.4 Design

Figure 3.2 shows LADOHD and its workflow for the detection of anomalies. The framework involves four components. First, a data generation phase, in which  $N$  machines running an EDR agent generate activity event sequences  $s_1, s_2, \dots, s_N$  that are collected in a centralized database. Second, a data selection step that extracts from the collected sequences the events related to the application of interest and form the subsequences ( $s^*_1, s^*_2, \dots, s^*_M$ ). Third, a model generation phase that uses the selected subsequences to form the training and validation datasets used to train the LSTM-based model. Finally, the anomaly detector component that deploys the trained model to determine whether the events of a given testing sequence are anomalous.

Table 3.2.: Description of the Different Types of Events

ID	Event type	Actor	Target	No. Actions
0	Session	User	N/A	3
1	Process	Process	Process	5
2	Module	Process	Module (e.g. dll files)	3
3	File	Process	File	12
4	Directory	Process	Directory	14
5	Registry key	Process	Windows registry key	7
6	Registry value	Process	Windows registry value	4
7	Host Network	Process	IP address	3

### 3.4.1 Data Generation

A machine  $M_i$  runs an enterprise EDR agent that records activity events as they occur in the system. These events form a corresponding event sequence referred to as  $s_i$ . A group of  $N$  monitored machines generate the set of event sequences  $S = \{s_1, s_2, s_N\}$ , which is pre-processed and used as time-series data to train the final LSTM-based model.

An activity event  $e_i$  in the sequences can be thought as a  $m$ -dimensional vector of features  $\{f_i^1, f_i^2, \dots, f_i^m\}$ , where  $f_i^j$  represents a categorical or continuous piece of information of the reported activity. One of these features is *event type*. The EDR product generates eight event types and each has a specific set of features (including the event type itself). Namely, the number of features  $m$  varies per event type. Some features such as *event type*, *actor*, *action* and *target* are common in all the activity events (regardless their types) as they define a complete semantic for any given event  $e_i$ : “this is an event of *this type* in which *this actor* executes *this action* over *this target*.” Table 3.2 summarizes the different types of events generated by the EDR software and the features actor, target, and action related to each type. There is a

set of specific actions available to each event type. The complete list is not included per the request of the company owning the security product. An example of an event  $e_i$  and its interpretation considering the four common features listed above is as follows. The process *svchost* is an integral part of Windows OS that manages system services running from Dynamic Link Libraries (DLL). Its purpose is to speed up the startup process by loading the required services specified in the service portion of the registry. When a DLL file is loaded by *svchost*, an event of type Module is generated. The actor of the generated event is *svchost*, while *load* (predefined in the product) is the action taken over the target *DLL file*.

### 3.4.2 Data Selection

LADOHD requires the definition of a finite set of categorical events (or symbols)  $E$ , which represents the set of all possible system activity events analyzed by the LSTM-based model. This set is referred to as the vocabulary of events.

**Vocabulary of Events Definition.** It can be thought as a transformation function  $F_T(\cdot)$  that changes the event feature vector generated by the EDR agent. Given an event  $e_i = \{f_i^1, f_i^2, \dots, f_i^m\}$  of type  $f_i^t = f_i^{j \in \{1, 2, \dots, m\}}$  and a set  $F$  of  $k \leq m$  selected features for events of type  $f_i^t$ , the feature transformation is given by:

$$F_T(e_i, F) = \begin{cases} e_i^* = \{t_i^1, t_i^2, \dots, t_i^k\} & \text{if } F \subseteq e_i \\ \emptyset & \text{otherwise} \end{cases} \quad (3.2)$$

In Equation 3.2,  $e_i^*$  is the transformed version of the event  $e_i$ . Each transformed feature  $t_i^{j \in \{1, 2, \dots, k\}}$  correspond to one of the  $k$  selected features in  $F$ . The transformation of each feature is a design choice that controls the granularity and the total number of possible events (i.e., vocabulary size). This is illustrated in Table 3.3 with the feature target, whose final value can be of either low or high granularity. Rows 1 and 2 are two Module events in which the same process loads two different DLL files. Assuming  $f_i^t$  ( $t \in \{1, 2, \dots, k\}$ ) were the target-related feature of these Module events,  $f_i^t$  might correspond to either the frequency of the DLL file in the distribution observed

Table 3.3.: Examples of Activity Events With Different Granularities

Event type	Actor	Action	Target (high granularity)	Target (low granularity)
Module	Process A	Load	DLL file1	Range 2 ( $100 \leq \text{frequency of file1} \leq 500$ )
Module	Process A	Load	DLL file30	Range 2 ( $100 \leq \text{frequency of file30} \leq 500$ )
Host Network	Process A	Connect	200.12.12.10	External connection
Host Network	Process A	Connect	192.168.10.3	Internal connection

during training (low granularity) or the individual file itself (high granularity). In the former case, the two Module events would be represented by the same transformed feature vector, which translates to the same symbol in the final categorical vocabulary of events. In the latter case, two different symbols are produced. Similarly, if  $f_i^t$  were the target feature of the Host Network events in rows 3 and 4,  $f_i^t$  might either indicate whether the network connection is internal or external (low granularity) or the individual destination IP addresses (high granularity). With the low granularity interpretation, the Host Network events pass from including the entire set of IP addresses to including a binary piece of information, reducing the number of symbols that form the vocabulary.

Figure 3.2 shows the effect of applying the definition of the vocabulary to the selection of events. From the  $N$  original sequences,  $M \leq N$  are chosen for the training phase. This is because  $F_T(\cdot)$  does not produce an output when the processed event does not include the features defined in  $F$ . For a given sequence of activity events  $s_i$ ,  $F_T(\cdot)$  and  $F$  operate as a filter to select which events are kept and what pieces of information from them are used to generate the transformed events that form the training subsequences  $s_i^*$ . For instance, in order to build the profile of an application  $A$ , the actor feature is included in the set  $F$  and  $F_T(\cdot)$  is defined so that only events with application  $A$  as actor are selected. Thereby, any sequence  $s_i$  with events produced by different applications is reduced to the subsequence  $s_i^*$ , which only includes events whose actor is application  $A$ . Any sequence  $s_i$  with no event with application  $A$  as actor is disregarded.

Based on the definition of  $F_T(\cdot)$  and the selected features  $F$  for each event type, there is finite set of transformed feature vectors, which are translated one-to-one to the set of categorical symbols  $E = \{1, 2, \dots, |E|\}$ . Whereby, a final subsequence  $s_i^*$  is comprised of these categorical symbols.

### 3.4.3 Model Generation

The selected  $M$  subsequences  $s_i^*$  are used to train the LSTM-based model following an either by-machine or by-time split. Splitting by machine refers to select approximately 80% of the the entire training subsequences  $s_i^*$  for training, leaving 20% for validation. Splitting by time, in contrast, refers to approximately select the first 80% of events in each subsequence  $s_i^*$  for training, leaving the remaining 20% of events for validation. In either splitting strategy, the resulting training and validation subsequences are concatenated to respectively form the unique training and validation sequence  $s_t$  and  $s_v$ , such that  $s_t \cap s_v = \emptyset$ .

Our LSTM-based model consists of a encoder of three layers of LSTM followed by a linear layer as suggested in [99]. At training, we use a timestep window  $w = 1$  to compute the probability of each event of the sequence given the previous subsequence. For better results, we apply a variety of strategies such as Stochastic Gradient Descent with Restart (SGDR) [100] and Cyclical Learning Rates [101]. The hyperparameters of the model were tuned to get the best performance for the dataset described in Section 3.5.1: (1) a batch size of 64, (2) an unrolling window (Batch Propagation Through Time or BPTT) of 64, (3) an embedding size of 16, and (4) 100 activations in the linear layer.

### 3.4.4 Anomaly Detector

At testing time, our trained LSTM-based model is used to classify each event in a sequence as either benign or anomalous. To classify the event  $e_t$  observed a timestep  $t$ , our detector follows four steps. In step 1, the previous subsequence  $e_1, e_2, \dots, e_{t-1}$  observed until time  $t-1$  is passed as input to our trained LSTM-based model. In step 2, the model computes the probabilities of each event in  $E$  (vocabulary of events) to be the next event in the sequence given the previous subsequence. Step 3 is a procedure that creates a set of probable events  $K \subset E$ , whose elements are the events with the highest probabilities. The size of the set  $K$  can be set either statically or dynamically.

For the static assignment, we customize the parameter  $k$  to chose all the events in the output model whose probabilities are within the top- $k$  probabilities. This resemble the use a fixed threshold that takes all the events above the smallest probability among the the top- $k$  ones. The dynamic assignment, in contrast, select the most probable events based on the natural division between high and low values found in the model output. The natural division is achieved by using the most repeated probability in the output array as threshold. Probabilities above this threshold belong to the high-value set, while the remaining probabilities are assigned to the set of low values. In the final step (step 4), the event  $e_t$  is classified as benign if  $e_t \in K$ , otherwise anomalous.

### 3.5 Evaluation Setup

#### 3.5.1 Dataset

The sequences for training and testing were collected on normal and under attack conditions respectively. Security experts (referred to as Red Team) conducted specific attacks on a Windows machine during the collection period of the testing sequence. The undue activities are reflected as either specific unseen events or subsequences of expected events in an unexpected order. Table 3.4 summarizes the entire dataset. Pre-filter refers to the total number of events collected from multiple endpoints before filtering them out using the definition of the vocabulary of events. In contrast, post-filter refers to the sequences obtained after filtering. The collected data was filtered out to include events generated by the process *PowerShell* as actor. This is an application commonly used to conduct stealthy data exfiltration. Our intuition was that learning its behavior in normal (or non-attack) conditions would allow detecting malicious patterns resulting from the activities of the insider threat.

**Vocabulary of Events.** The objective was to capture as much information as possible from the PowerShell application. The selection of the set of features  $F$  of each event type was done based on the data observed in the training sequence. We traded-off granularity with the total number of possible events in order to avoid



having a large number of events observed only at training (and not at testing) and vice versa. Following this guidance, each event  $e_i$  was processed using the set of features  $F = \{f_i^1, f_i^2, f_i^3, f_i^4, f_i^5, f_i^6\}$ , where:

- $f_i^1$ : Actor feature. Its corresponding  $t_i^1$  is a unary piece of information defining the actor (always 0 for powershell.exe).
- $f_i^2$ : Event type feature. Its corresponding  $t_i^2$  might be any of the eight event type IDs described in Table 3.2.
- $f_i^3$ : Action feature. Its corresponding transformed featured  $t_i^3$  was defined per event type. It can vary from 0 to 13 depending on the event type as specified in Table 3.2.
- $f_i^4$ : Target feature. Its  $t_i^4$  depends on the event type. For Process events  $t_i^4 = 0$  (not powershell.exe) and  $t_i^4 = 1$  (powershell.exe). For Module events  $t_i^4 = 0$  (not a DLL file) and  $t_i^4 = 1$  (DLL file). For Registry Value events  $t_i^4 = 0$  (Others),  $t_i^4 = 1$  (HKEY\_USERS), and  $t_i^4 = 2$  (HKEY\_LOCAL\_MACHINE). For the rest of event types  $t_i^4$  operates as unitary piece of information.
- $f_i^5$ : Network feature. Its  $t_i^5$  is ternary piece of information about Host Network events only (0 for self connection, 1 for internal connection, and 2 for external connection). For the rest of event types this feature operates as unitary piece of information.
- $f_i^6$ : User feature. The transformed feature  $t_i^6$  is a binary piece of information about the user executing the action (0 for system-related user and 1 for non-system-related user). For Session and Registry Value events, this feature operates as a unary piece of information.

Each event  $e_i^* = \{t_i^1, t_i^2, t_i^3, t_i^4, t_i^5, t_i^6\}$  is extracted following the definitions above. With these definitions, the vocabulary size is 175 ( $E = \{0, 1, \dots, 174\}$ ), from which 41 and 31 events are present in the training and testing sequences respectively. Twenty

Table 3.4.: Dataset Description

Training / Validation			Testing	
Pre-filter	Post-filter (trn)	Post-filter (val)	Pre-filter	Post-filter (test)
38,899,995	63,282	13,280	727,275	66,972

out of the 41 training events do not appear in the testing sequence. Likewise, ten out of the 31 testing events are not present in the training sequence. These 10 events are referred to as unseen events.

**Training set** The final training ( $s_t$ ) and validation ( $s_v$ ) sequences were obtained by monitoring 30 machines. After filtering the collected sequences, we got a total of 76,562 events in 30 subsequences (including only events with *PowerShell* actor). We then applied a by-machine data split and selected the subsequences of the first 24 machines for training, leaving the subsequences of the remaining 6 machines for validation. After concatenating the corresponding subsequences, the resulting training and validation sequences had a length of 63,282 (82.65%) and 13,280 (17.35%) events respectively. Figure 3.3 shows the collection dates from those 30 machines. This sparse collection timeframe was intended to capture the behavior of *PowerShell* in normal conditions as no attacks were reported in these collection periods.

**Testing set.** This sequence of 66,972 events was obtained from a different victim machine monitored from May 8th to May 11th. The Red Team launched an multi-stage persistent attack consisting of a user escalation step followed by a data exfiltration attack on this victim machine as follows:

- The Red Team was initially assigned a non-administrative user for the victim computer.
- The hacking tool Mimikatz is used to steal credentials from the memory of the victim machine, performing a user escalation.

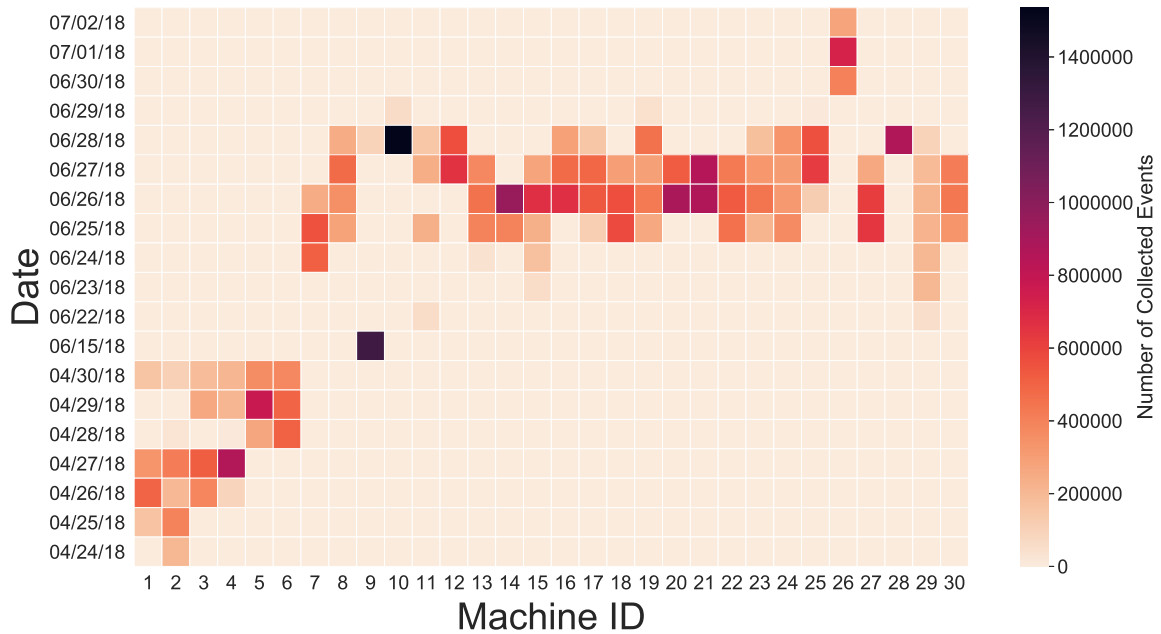


Fig. 3.3.: Data collection dates from 30 machines, from April 27th to July 7th of 2018. This sparse collection timeframe was intended to capture the behavior of the application Powershell in non-attack conditions.

- The Red Team roams on the network to which the victim computer is connected and discovers other resources (shared folders, machines, users, domains). Remote shares are discovered and the corresponding files are copied to the victim machine.
- Powershell files are also copied to the machine and then executed, in order to compromise other computers and extract files from them.
- An external remote connection is established from the victim machine to bypass the firewall protection. The copied files are sent outside the network through this connection.

### 3.5.2 Metrics and Ground Truth

The Red Team provided a log file enumerating the sequence of steps followed during the execution of the attack. Each entry in this file contains a high-level description of the step and a timestamp indicating the day and time of its execution. Some of the steps in the file are identifiable as actions executed by Powershell. This information was used to find the corresponding events in the testing sequence, which is formed by events in which Powershell functions as actor only. Specifically, we found 110 matches. Additionally, the EDR of the security company produced four alerts while the Red Team conducted the attack on the victim machine. These alerts were matched with the corresponding events in the testing sequence, which has 117 unseen events (i.e. events present in the testing sequence but not in the training sequence). These alert-related events along with the Red-Team-matched and the unseen events form the sequence portions corresponding to malicious activity. As the events of two of the alerts corresponded to unseen events, only the other two alerts add new well-identified events to the malicious portions. Each of these two alerts are matched with 4 events in the sequence making a total of 295 malicious events. Malicious events are expected to be detected as anomalous. Events other than alert-related, Red-Team-matched, and unseen events are regarded benign and are expected to be classified accordingly. We set our ground truth based on these assumptions and measure the performance of our method according to the following metric definitions:

- True Positive(TP): Any malicious event classified as anomalous.
- False Negative(FN): Any malicious event classified as benign.
- True Negative (TN): Any non-malicious event classified as benign.
- False Positive (FP): Any non-malicious event classified as anomalous.
- True Positive Rate (TPR):  $TP/(TP + FN)$
- False Positive Rate (FPR):  $FP/(FP + TN)$

## 3.6 Experiments

This section presents the experiments conducted to evaluate our LADOHD framework using the datasets described in Section 3.5.1. The experiments were designed to answer a series of research questions, which are included along with our findings in the following sections.

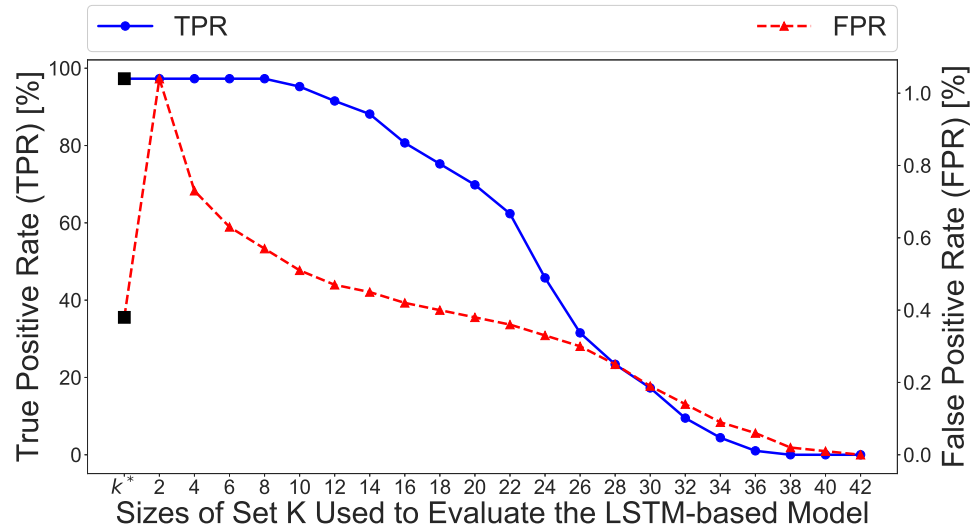
### 3.6.1 Dynamic vs. Static Selection of the Set of Probable Events $K$

**RQ1.** What approach (either dynamic or static selection of  $K$ ) provides a better performance? If the static method does, what is the best value of the parameter  $k$ ?

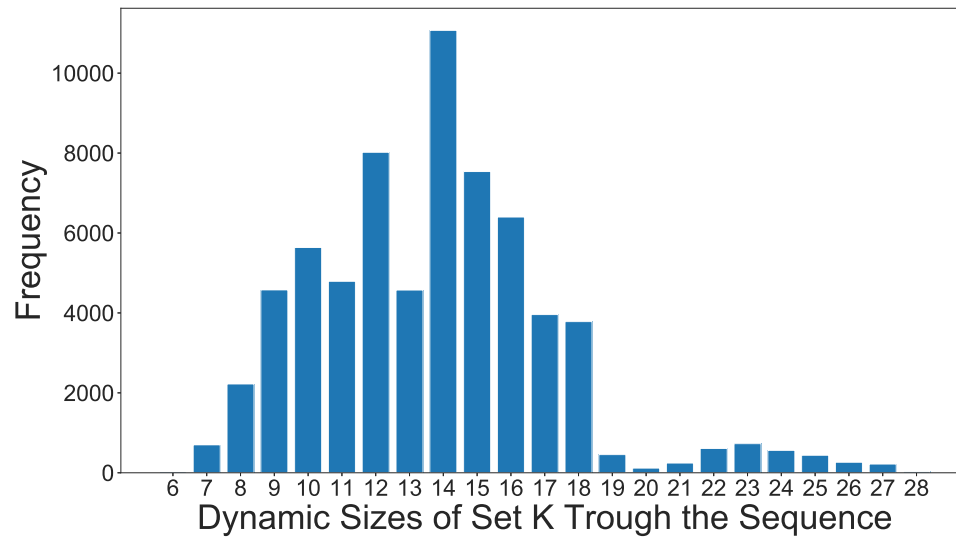
We investigate whether our technique identifies the well-identified malicious events. We are particularly interested in finding which approach provides the best anomaly detection performance. To this end, we measure the TPR and FPR variations as we change the number of events in  $K$  through both the dynamic and static approaches.

Figure 3.4a shows the results. In the  $x$ -axis  $k^*$  means that the size of  $K$  is dynamically adjusted in each timestep of the sequence. The corresponding values of the metrics TPR and FPR are marked with a black square to differentiate them from the values obtained through the static approach. The rest of values in the  $x$ -axis (from 2 to 42) corresponds to the values of the parameter  $k$  of the static approach. The figure illustrates how the dynamic approach outperforms the static approach for any chosen  $k$  as the former gives a high TPR of 97.29% while keeping a low FPR of 0.38%.

The static approach starts with a similar TPR but a higher FPR in comparison with the dynamic approach. As the parameter  $k$  increases, both the TPR and the FPR decrease. The static approach achieves the same FPR as the dynamic when  $k = 20$ . At this point however, the corresponding TPR has decreased from 97.29% to 69.83%. The non-functionality of the static method can be explained by the high variance in the distribution of the natural division between high and low values in output of the model throughout the entire sequence. Figure 3.4b shows this distribution. The dynamic approach produces sets  $K$  with sizes between 6 and 28 (inclusive) with



(a) Metric Variation With Static and Dynamic Selection of K



(b) Distribution of Dynamic K Sizes Throughout the Sequence

Fig. 3.4.: Selection of the set  $K$  through both the dynamic and static approaches. Figure 3.4a shows the variation of the  $TPR$  and  $FPR$  with respect to different setting for  $K$ . Figure 3.4b presents the distribution of the dynamic sizes of  $K$  throughout the testing sequence. Notice its high variability.

significant differences in their frequencies. Setting a specific  $k$  in the static approach to be used in each timestep of the analyzed sequence goes away from the decision

made by the model, which clearly discriminates low and high values in the its output probabilities.

**Findings.** The dynamic approach outperforms the static method to select the set of probable events  $K$ , having a TPR 1.31X better than the static method for the same FPR of 0.38.

### 3.6.2 Comparison With an Enterprise Endpoint Detection and Response (EDR)

**RQ2.** What is the performance of LADOHD with respect to enterprise-level EDR system currently in production?

One of the main challenges defending against insider threats is the similarity between benign and malicious activities. Discerning between them is a difficult task. Due to this restriction, this section evaluates how efficient LADOHD is by comparing it with the the enterprise EDR of the company already in production.

The enterprise EDR is a multi-layer system that employs signature-based, supervised (e.g. Random Forest [102]), and unsupervised (e.g., clustering [103]) machine learning methods for analysis, detection, and alerting. This system detected 4 alerts while monitored the victim machine during the Red Team attack. These alerts were validated to correspond to malicious activities and later matched with the corresponding events in the generated testing sequence as indicated in Section 3.5.2. Although the alerts suffice to detect the ongoing attack, the EDR missed the detection of the execution of well-known exploitation tools such as *mimicatz* and related activities Figure 3.5 shows the malicious activities reported by the Red Team in the log file that were matched with events in the testing sequence. It shows their counts in log file and the number of events matched per each activity. There is a total of 110 malicious events. All of them were classified by LADOHD as anomalous.

Unlike the current analytics modules of the EDR, LADOHD is trained with benign data only and learns sequence patters of a particular application. Uncommon

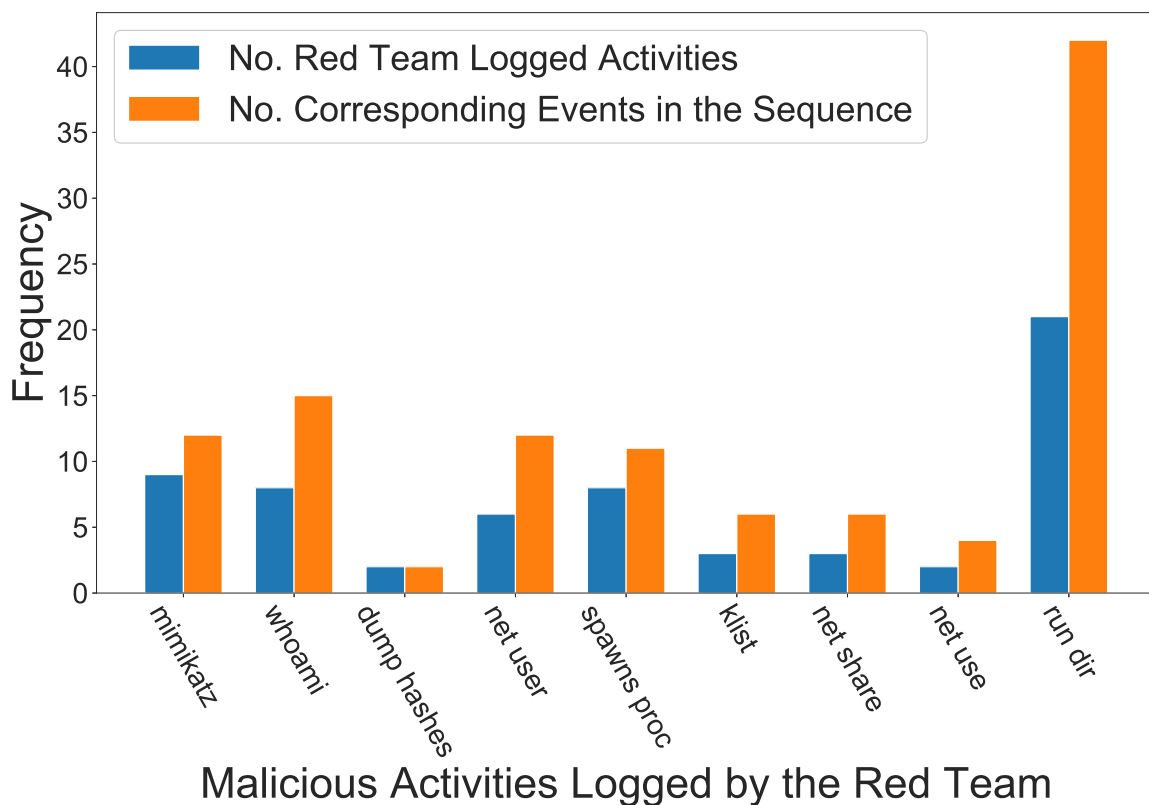


Fig. 3.5.: Malicious activities reported by the Red Team. These are the activities that could be matched with events in the testing sequence.

event sequences are classified as anomalous through the detection of specific events regardless the meaning of the event itself. It is important to mention that LADHD produced 254 *FP* cases along with the with the 287 *TP* cases (including unseen events). Although the number of *FPS* is low with respect to the length of the sequence, they might represent a high volume of cases to be revised by security experts in a short periods of time.

**Findings.** LADOHD successfully detected the ongoing attack generating more alerts than the enterprise EDR. A relatively small number of *FP* cases with respect to the sequence length were generated in the process.



Table 3.5.: Metric Values Obtained With the Original and Clean Testing Sequences

Testing Sequence	TP	FN	FP	TN	TPR	FPR
Original	287	8	254	66423	97.29%	0.38%
Clean	110	8	256	66421	93.22%	0.38%

### 3.6.3 Performance of LADOHD When Processing Sequences Without Unseen Events

**RQ3.** Does the capacity LSTM-based models to detect anomalies improve when unseen events are discarded in advance?

We now evaluate whether processing unseen events improve or diminish the capacity of LSTM-based models to detect malicious events. We are interested in knowing whether the same Red-Team-matched events classified as anomalous in experiment 1 (Section 3.6.1), when unseen events are part of the sequence, are also classified as anomalous when unseen events are ignored. We also want to validate whether the missing alert-related events are detected. To this end, we create a clean testing sequence by removing the unseen events from the testing sequence. We pass this clean testing sequences to our LSTM-based model, which classifies each event in the sequence as either benign or anomalous. Table 3.5 includes a comparison of the results obtained with the original testing sequence and its clean version. Removing the unseen events does not help the model classify as anomalous new malicious events. The number of *TP* cases related to observed events remains the same. The same phenomenon occurs with the number of *FPS*, where an increase of only 2 is observed.

**Findings.** There is neither a significant improvement nor detriment when unseen events are discarded. Their removal from the sequence should be determined by the performance cost they might cause only.

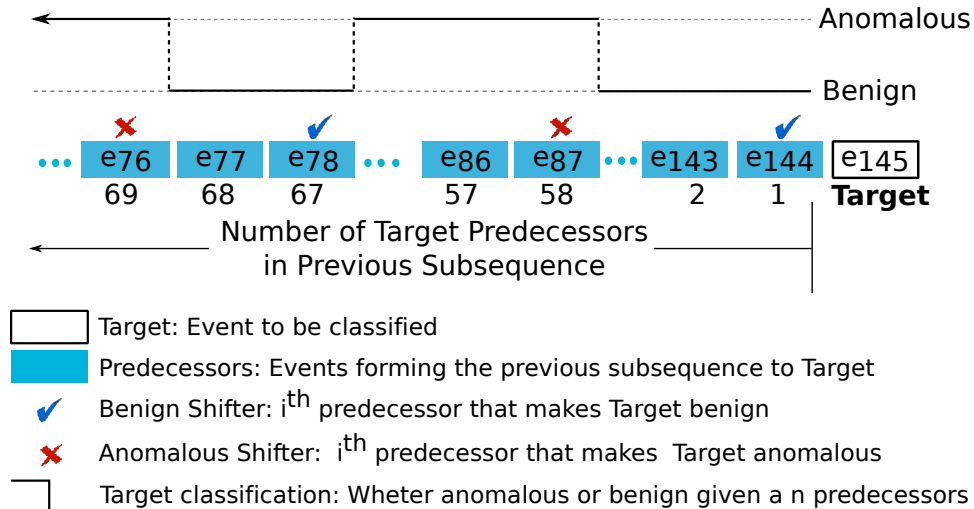


Fig. 3.6.: Effect of long-term dependencies of LSTM models in the detection of anomalies. The example is based on the event  $e_{145}$  in Table 3.6

### 3.6.4 Effect of Long-Term Dependencies in the Detection of Anomalies

**RQ4.** What impact does the length of the previous sequences have over the detection of anomalies?

One of the main characteristics of LSTM networks is their capacity to learn long-term dependencies among events in a sequence. One interesting question we aim to answer is whether these long-term dependencies have an impact in the classification. Namely, we want to validate whether considering subsequences of different lengths (by increasing the number of predecessors) cause different outputs in the classification of an event. To this end, we work with the clean testing sequence of the Section 3.6.3.

Table 3.6 shows the results. It includes 9 randomly selected events of these sequence classified as anomalous. These events are referred to as targets and they correspond to the last events of low probable subsequences in the sequence. Figure 3.6 illustrates the procedure followed in this experiment using the target  $e_{145}$  (row 4 in the table) as example. We incrementally move backward from each target until the beginning of the sequence and find the number of predecessors (length of previous sequence) that causes a change in the classification. The number of predecessors that

Table 3.6.: Effect of Long-Term Dependencies on the Detection of Anomalies

Target	Benign Shifters	Anomalous Shifters
$e_{142}$	[1]	[98]
$e_{143}$	[1]	[75]
$e_{144}$	[1]	[75]
$e_{145}$	[1, 67]	[58, 69]
$e_{146}$	[1]	[54]
$e_{147}$	[1]	[50]
$e_{148}$	[1]	[47]
$e_{1032}$	[1, 551, 619]	[549, 587, 648]
$e_{2292}$	[9]	[1, 11]

causes the classification to be benign are called benign shifters. Those that cause the classification to be anomalous are referred to as anomalous shifters. Table 3.6 shows that most of the targets have multiple shifters, which prove that the history of events is what actually has a significant impact in the classification process. LSTM-based models have the potential to correctly classify events regardless the length of the previous sequence. Their outputs are in fact affected by the hidden state. An interesting observation is the capacity of our LSTM-based model to look backward a variable number steps to estimate the probability of a particular event. We have cases where the model makes the final decision based on a few number of predecessors (e.g., 2 predecessors in the case of  $e_{142}$ ) and others in which the model considers a large number of them (e.g., 648 predecessors in the case of  $e_{1032}$ ). This ability to relate current events with distant past events in the sequence shows the potential of LSTM networks to solve the OAR problem.

Another question we aim to answer in this experiment is how the probabilities of the targets change as the number of predecessors gets close to a shifter. We did not

find a clear relationship between the relative probability of the target (with respect to the other possible events) in the output of the model and the proximity to the shifters. The probability of the target does not always increase or decrease as the number of predecessors gets close to a benign or anomalous shifter respectively. This phenomenon can be explained by the fact that our model is trained to predict the next event in the sequence and not to estimate the least probable events.

**Findings.** LSTM models can correctly classify events regardless the length of the previous subsequences. The history of events (hidden state of the model) is what actually affects the classification. This allows LSTMs being a potential solution against the OAR problem.

### 3.6.5 Prediction Capacity of LSTM and HMM Based models Over Variable-Length Sequences

**RQ5.** How much accurate are LSTM-based models compared to other solutions such as HMM in the prediction of the next event in variable length sequences?

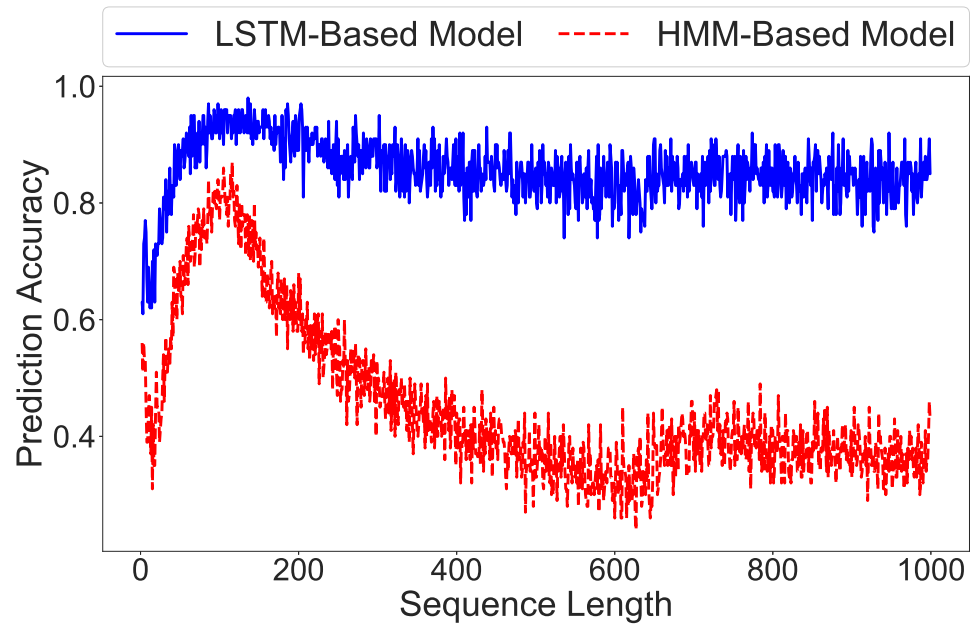
This section presents a comparison of the prediction capacity of our LSTM-based model and a full connected HMM model built with the same dataset. As the ability to discern between benign and anomalous events depends on the prediction capacity of the model, we want to evaluate which model predicts better the last event of sequences of different lengths. To this end, we took 100 continuous subsequences of a specific length and measure the accuracy of the model predicting the last event of the sequences. The lengths were chosen to vary from 2 to 1000. To ensure a fair comparison, we measure the prediction accuracy of both models with incremental-length subsequences extracted from both the training and testing sequences. The idea of using these two sets of subsequences is to validate that the results do not come from any bias that the testing data might induce. We do so because we are interested in observing how the prediction capacity of the models changes as the lengths of the sequences increase in ideal conditions (i.e. when sequences were observed in training)

and not in comparing which model is more efficient when processing new data. Figure 3.7 shows that our LSTM-based model constantly outperform the prediction capacity of the HMM model. Our model keeps predicting well with larger sequences, while the accuracy of the HMM-based model decreases.

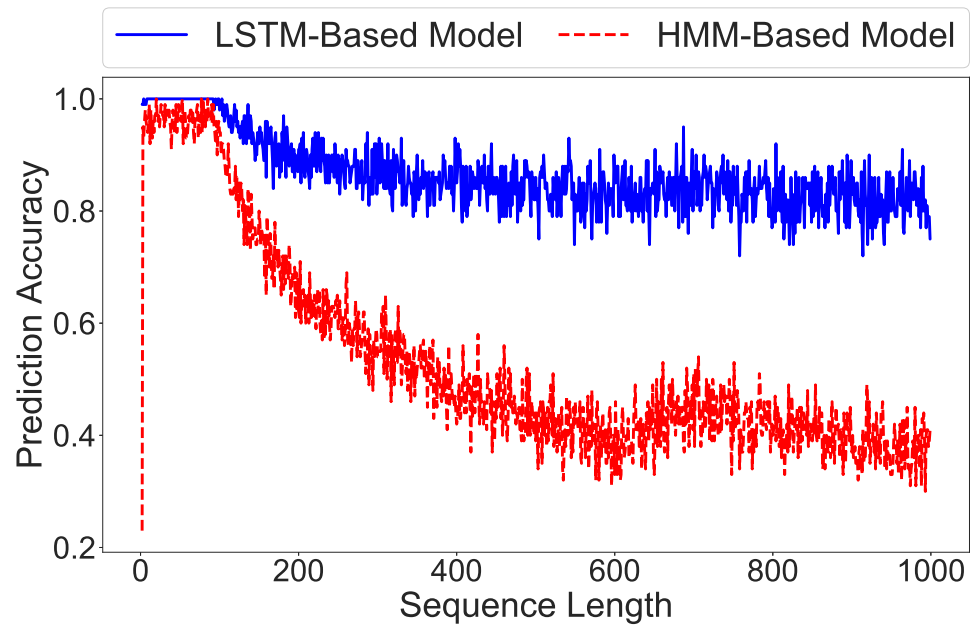
**Findings.** LSTM-based models have a better capacity than HMM-based models to predict the next event in a given sequence as its length increases.

### 3.7 Conclusion

This paper presents LADOHD, a generic LSTM-based anomaly detection framework to protect against insider threats. for high dimensional sequential data. We evaluated the framework with an extensive dataset of activity events generated by the EDR of a renown security company. Each event in the dataset represents a high dimensional vector of features The framework filters out the events per application and pre-specified features that define the vocabulary of possible events that form the sequences analyzed by our model. Each event in the sequence is classified as either benign or anomalous given the previous observed subsequence. LADOHD reached a high  $TPR > 97\%$  with a low  $FPR < 0.4\%$ , proving the effectiveness of the framework. Furthermore, this work presents a comprehensive analysis of how LSTM-based models work and compare them to alternative solution such as HMM-based models. We found that LSTM-based models rank better the set of expected events in each timestep of sequence than HMM-based models, which favor their capacity to detect anomalies.



(a) Accuracy With Training Sequences



(b) Accuracy With Testing Sequences

Fig. 3.7.: Prediction accuracy of our LSTM and the HMM models with sequences of incremental lengths. Figure 3.7a and Figure 3.7a show the accuracy variation with subsequences extracted from the D1 training and testing sequences respectively.

## 4. AN MTD-BASED SELF-ADAPTIVE RESILIENCE APPROACH FOR CLOUD SYSTEMS

### 4.1 Introduction

Recent advances in cloud computing infrastructures have given increased traction to the adoption of cloud-based systems for reliable and elastic computing needs of enterprises. However, in a cloud-based environment, the enlarged attack surface hampers attack mitigation, especially when attacks originate at the kernel level. In a virtualized environment, an adversary that has fully compromised a virtual machine (VM) and has system privileges, exposes the cloud processes to attacks that might compromise their integrity, jeopardizing mission-critical functions.

A major issue with existing cloud defense solutions is that they target specific threats, which makes them ineffective for fighting against attacks lying outside their protection perimeter. In order to provide effective threat mitigation across various cloud systems, it is critical to design a resiliency solution in which the protection against attacks is integrated across all layers of the system at all times. This requires designing cloud enterprise frameworks that can accurately detect system anomalies and dynamically adapt through *starting secure*, *staying secure*, and returning to *secure+* [104] state in cases of cyber-attacks.

We propose an approach for cloud system resiliency that is capable of dynamically adapting to attack and failure conditions through performance/cost-aware process replication, automated software-based monitoring and reconfiguration of virtual machines. The proposed approach offers many advantages over existing solutions for resiliency in trusted and untrusted clouds, among which are the following:

- The solution is generic and targets multiple layers of the cloud software stack, as opposed to traditional techniques for mitigation targeting specific attacks.

- The proposed resiliency framework facilitates proactive mitigation of threats and failures through active monitoring of the performance and behavior of services and can incorporate new tools to resiliency and antifragility under various failures and attacks.
- Continuous monitoring, restoration and healing of cloud system operations allows for starting secure, staying secure and returning secure+ by learning from the attacks and failures and reconfiguring processes accordingly to increase resiliency.

The rest of this paper is organized as follows: Section 4.2 provides an overview of monitoring and security approaches in distributed computing. Section 4.3 introduces the proposed resiliency framework. Section 4.4 discusses the results of preliminary experiments for the feasibility of the approach. Section 4.5 concludes the paper.

## 4.2 Related Work

Current industry-standard cloud systems such as Amazon EC2<sup>1</sup> provide coarse-grain monitoring capabilities (e.g. CloudWatch) for various performance parameters for services deployed in the cloud. Although such monitors are useful for handling issues such as load distribution and elasticity, they do not provide information regarding potentially malicious activity in the domain. Log management and analysis tools such as Splunk<sup>2</sup>, Graylog<sup>3</sup> and Kibana<sup>4</sup> provide capabilities to store, search and analyze big data gathered from various types of logs on enterprise systems, enabling organizations to detect security threats through examination by system administrators. Such tools mostly require human intelligence for detection of threats and need to be complemented with automated analysis and accurate threat detection capabil-

---

<sup>1</sup><https://aws.amazon.com/ec2>

<sup>2</sup><https://www.splunk.com>

<sup>3</sup><https://www.graylog.org>

<sup>4</sup><https://www.elastic.co/products/kibana>



ity to quickly respond to possibly malicious activity in the enterprise and provide increased resiliency by providing automation of response actions.

Various moving target defense (MTD) solutions have been proposed to provide protection against specific threats in systems. However, these are only effective against attacks within their scope. For instance, while application-level replication schemes mitigate attacks targeting the application code base, they fail in the case of code injection attacks targeting runtime execution. Randomizing runtime [105], and system calls [106], instruction set randomization [107] and address space randomization [108], have been successfully used to mitigate system-level attacks. Although most of these security mechanisms are effective for attacks they target, modern complex attacks against cloud systems call for defense approaches that are deeply integrated into the architecture, at all system layers and at all times.

### 4.3 Proposed Approach

We propose a novel approach that uses cloud-based domain activity monitors to audit service behavior and performance changes to detect anomalies that trigger the reconfiguration of the system. The reconfiguration is based on our virtualization-based MTD strategy for distributed applications, which benefits from the flexibility offered by software-defined networking (SDN) and its capability of dynamically configuring network devices via OpenFlow<sup>5</sup>. By integrating components for service performance monitoring and dynamic reconfiguration, the proposed model aims to provide a unified framework for agile and resilient computing in trusted and untrusted clouds. Figure 4.1 illustrates a high level view of the framework, based on the idea of *starting*, *staying*, and *returning* secure in the cloud process lifecycle as proposed by Goodwin et al. [104].

General characteristics of the solution are as follows:

---

<sup>5</sup><http://archive.openflow.org>

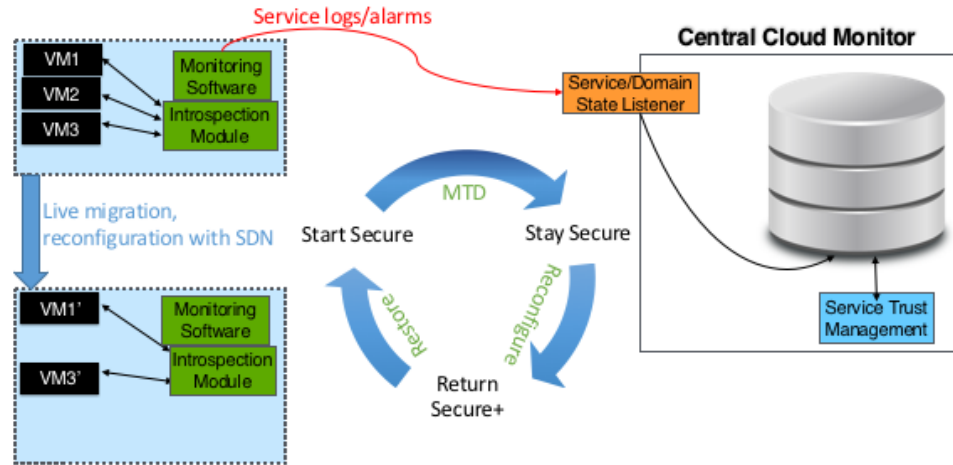


Fig. 4.1.: High-level view of resiliency framework

- The operations of each cloud-based service and domain are monitored using monitoring tools (e.g. Heat<sup>6</sup> and Monasca<sup>7</sup> for OpenStack<sup>8</sup>) built on top of the cloud platform. These tools report performance and security parameters such as response time, response status, CPU usage, memory usage, etc. to anomaly detection tools built on top of the same infrastructure.
- The analysis results by the anomaly detection tools are reported to a central monitor in the form of summary statistics for the services/VMs. The central monitor utilizes data submitted by the monitors to update trust values of services and reconfigure services to provide resiliency against attacks and failures.
- A moving target defense approach that migrates services to different platforms periodically to narrow the exposure window of a node to attacks is utilized, which increases the cost of attacks on a system and lowers the likelihood of success. Detection of service failures and/or suboptimal service performance, as well as integrity violations detected with virtual machine introspection also trig-

<sup>6</sup><https://wiki.openstack.org/wiki/Heat>

<sup>7</sup><https://wiki.openstack.org/wiki/Monasca>

<sup>8</sup><http://www.openstack.org>

ger restoration of optimal behavior through replication of services and adaptable migration of virtual machines to different platforms.

The following subsections provide details of the main components of the proposed resiliency approach.

### 4.3.1 Live Monitoring

Cyber-resiliency is the ability of a system to continue degraded operations, self-heal, or deal with the present situation when attacked [104]. For this we need to measure the assurance level (integrity/accuracy/trust) of the system from the Quality of Service (QoS) parameters such as response time, throughput, packet loss, delays, consistency, etc.

The solution developed for dynamic reconfiguration of service compositions as described in [109] involved a distributed set of monitors in every service domain for tracking performance and security parameters and a central monitor to keep track of the health of various cloud services. Even though the solution enables dynamic reconfiguration of entire service compositions in the cloud, it requires replication, registration and tracking of services at multiple sites, which could have performance and cost implications for the enterprise. To overcome these challenges, the framework proposed in this work utilizes *live monitoring* of cloud resources to dynamically detect deviations from normal behavior and integrity violations, and *self-heal* by reconfiguring service compositions through software-defined networking [110] of automatically migrated service/VM instances.

As the goal of the proposed resiliency solution is to provide a generic model, for detection of possible threats and failures in a cloud-based runtime environment, limiting the utilized anomaly detection models to supervised learning algorithms will not provide the desired applicability. Hence, unsupervised learning models such as *k-means clustering* [111] and *one-class SVM classification* [112] to detect outliers (i.e. anomalies) in service and VM behavior will be more appropriate. Algorithm 4 shows

## Algorithm 4: Content image generation

**Input:**  $x, M, l, \lambda_c, N$

- 1:  $x_c \leftarrow \text{rand\_init}(x)$
- 2:  $F \leftarrow M[:l]$
- 3:  $f_x \leftarrow F(x)$
- 4: **while**  $N \neq 0$  **do**
- 5:    $x_c \leftarrow x_c - lr \cdot \Delta$
- 6:    $N \leftarrow N - 1$
- 7: **end while**
- 8: **return**  $x_c$

## Algorithm 5: Content image generation

**Input:**  $x, M, l, \lambda_c, N$

- 1:  $x_c \leftarrow \text{rand\_init}(x)$
- 2:  $F \leftarrow M[:l]$
- 3: **return**  $x_c$

an adaptation of the k-means algorithm to cluster service performance data under normal system operation conditions and algorithm 5 shows how to detect outliers by measuring the distance of the performance vector of a service at a particular point in time to all clusters formed during training. Additionally, virtual machine introspection (VMI) [113] techniques need to be utilized to check the integrity of VMs at runtime to ensure that the application's memory structure has not been modified in an unauthorized manner. The results of the monitoring and anomaly detection processes help decide when to reincarnate VMs as described in the next section.

### 4.3.2 Moving Target Defense

Moving target defense (MTD) as defined by the US Department of Homeland Security is *controlling change across multiple system dimensions to increase uncertainty and complexity for attackers to increase the cost of their attack efforts* [114]. The proposed MTD-based attack-resilient virtualization-based framework is based on [115], a solution that reduces the vulnerability window of nodes (virtual machines) mainly through three steps:

1. Partitioning the runtime execution of nodes in time intervals
2. Allowing nodes to run only with a predefined lifespan on heterogeneous platforms (i.e. different OSs)
3. Live monitoring

The main idea of this MTD-technique is allowing a node running a distributed application on a given computing platform for a controlled period of time before vanishing it. The allowed running time is chosen in such a manner that successful ongoing attacks become ineffective and a new node with different computing platform characteristics is created and inserted in place of the vanishing node. The new node is updated by the remaining nodes after completing the replacement. The required synchronization time is determined by the application and the amount of data that needs to be transferred to the new node. as the reincarnation process do not keep the state of the old node.

The randomization and diversification technique of vanishing a node to appear in another platform is called *node reincarnation* [115]. One key question is determining when to reincarnate a node. One approach is setting a fixed period of time for each node and reincarnating them after that lifespan. In this first approach nodes to be reincarnated are selected either in Round Robin or randomly. However, attacks can occur within the lifespan of each machine, which makes live monitoring mechanisms a crucial element. Whether an attack is going on at the beginning of the reincarnation

process determines how soon the old node must be stopped to keep the system resilient. When no threats are present both the old node and new node can participate in the reincarnation process. The old node can continue running until the new node is ready to take its place. On the contrary, in case an attack is detected the old node should be stopped immediately and the reincarnation should occur without its participation, which from the perspective of the distributed application represents a greater downtime of the node.

Our main contribution here is the design and implementation of a prototype that speeds up the node reincarnation process using SDN, which allows configuring network devices on-the-fly via OpenFlow. We avoid swapping virtual network interfaces of the nodes involved in the process as proposed in [115] to save time in the preparation of the new virtual machine. The new virtual machine is created and automatically connected to the network. The machine then starts participating in the distributed application when routing flows are inserted to the network devices to redirect the traffic directed to the old VM to the new one.

#### 4.4 Experiments

Experiments to evaluate the operation times of the proposed MTD solution were conducted. Figure 4.2 shows the experiment setup. A Byzantine fault tolerant (BFT-SMaRt) distributed application was run on a set of Ubuntu (either 12.04 or 14.04 randomly selected) VMs in a private cloud, which are connected with an SDN network using Open vSwitch<sup>9</sup>. The reincarnation is stateless, i.e. the new node (e.g. VM1') does not inherit the state of the replaced node (e.g. VM1). The set of new VMs are periodically refreshed to start clean and the network is reconfigured using OpenFlow when a VM is reincarnated to provide continued access to the application. Table 4.1 presents the results: virtual machine restarting and creation time, and Open vSwitch flow injection time. Note that the important factor for system downtime here is the

---

<sup>9</sup><http://openvswitch.org/>

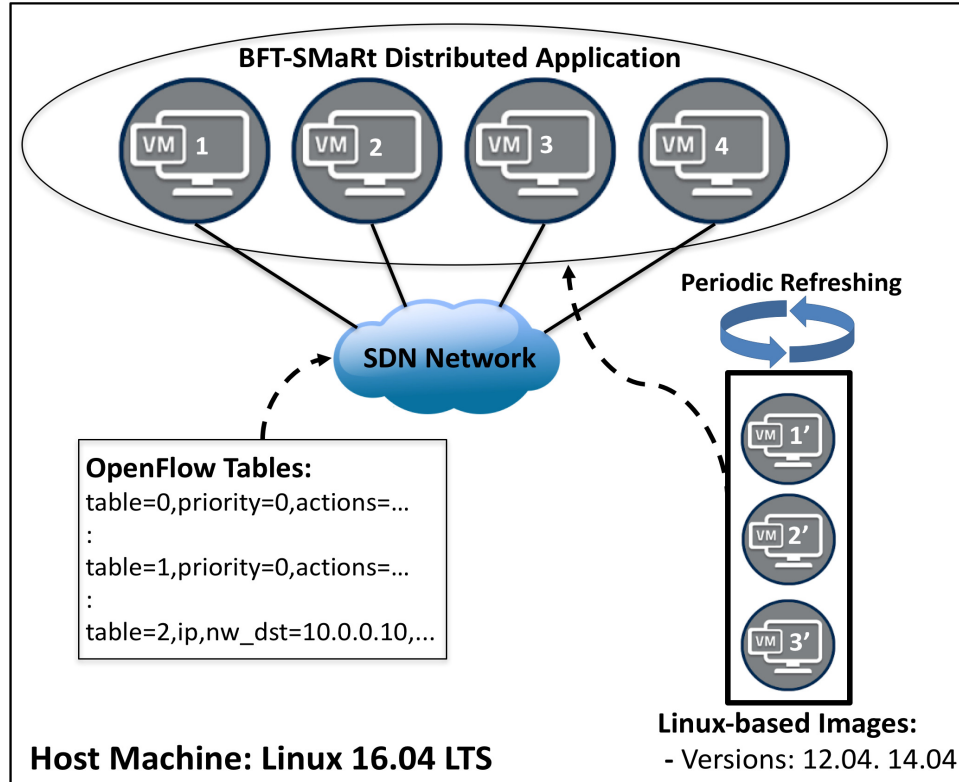


Fig. 4.2.: Experiment setup

Table 4.1.: Reincarnation Process Times

Measurements	Times
VM restart time	$\sim 7s$
VM creation time	$\sim 11s$
Open vSwitch flow injection time	$\sim 250ms$

Open vSwitch flow injection time, as VM creation and restart take place periodically to create fresh backup copies, and do not affect the downtime.

## 4.5 Conclusion

We proposed a novel approach to introduce resiliency into cloud systems such that they can mitigate attacks and failures to provide uninterrupted operation of critical functions. The solution is based on distributed monitoring of cloud service/VM behavior and periodic refreshing of the related cloud resources to allow self-adaptive reconfiguration through SDN with a moving target defense approach. We demonstrated with preliminary experiments that the MTD-based solution is able to achieve acceptable reconfiguration times. In future work we will focus on the development and evaluation of a full resiliency framework for cloud systems based on the ideas presented in this work, not only for stateless but also for stateful distributed applications.



## 5. FUTURE WORK

This chapter presents a brief description of our findings and new research directions with respect to the work of this dissertation. The discussion about our future plan and work is separated by topics in the sections below.

### 5.1 Protecting Neural Networks Against Adversarial Attacks

Deep Neural Networks (DNNs) are vulnerable to adversarial settings such as adversarial sample, patch and trojan attacks. These attacks induce misclassification at testing time, which diminishes the applicability of these models in real-world scenarios. The work presented in **Chapter 2** introduces a new model hardening technique called *ConFoc* to protect against trojan attacks. The method changes the parameters of compromised models through the healing process so as to remove any inserted trojan. Whereby, adversaries lose the knowledge about the parameters of the model once it is taken through the *ConFoc* process. As *ConFoc* assumes adversaries do not have access to the details of the victim models at testing anymore, the technique is not functional against adversarial sample and patch attacks. To overcome this limitation, we are currently working on a adversarial input detection technique, which determines whether an input includes malicious modifications that lead to misclassification. The new research question is to answer whether there are neurons that specifically relate to the content or semantic information of inputs. If so, we want to find out whether it is possible to separate content-related from style-related neurons. In the case of having positive answers to these two research questions, the next step is to build a content-focus model parallel to the original one. This new model is characterized for having its content-related neurons strengthened and weak non-content neurons. At testing time, any input is passed to both the original and the

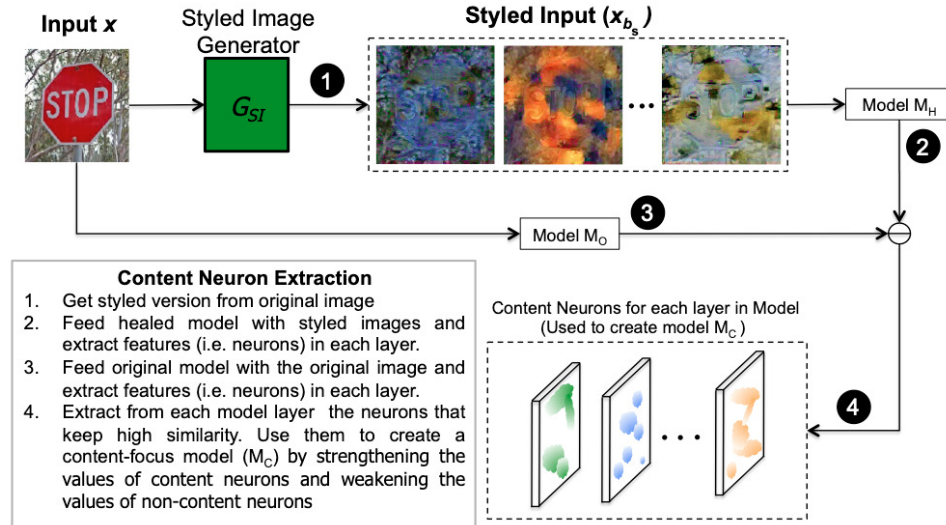


Fig. 5.1.: Possible extension of *ConFoc* to detect adversarial inputs of both adversarial sample and trojan attacks.

content-focus models for evaluation. The input is considered adversarial in case of a mismatch in the classification. Figure 5.1 shows the possible procedure to create the content-focus model.

## 5.2 Neural Networks on Anomaly Detection

One of the main limitations on applying machine learning algorithms in cybersecurity is the lack of training and validation data. This situation is aggravated by other factors such as the high cost of errors and lack of functional guarantees when processing unobserved data. Neural Networks do not scape from these restrictions and, in addition, add a lot of uncertainty due to their ingrained back-box nature. **Chapter 3** presents an anomaly detection solution based on LSTM models, which demonstrates the potential of such algorithms to detect attacks unnoticed by current enterprise systems. Although the results show the functionality and applicability of the framework, the framework was not exhaustively tested with because of the lack of testing data and ground truth, including a more diverse set of attacks. Therefore,

our first step in our feature work is the collection of more data logs including attacks from diverse sources so as to test the soundness of our approach. Our second step is directed to the automatic definition of the vocabulary of events. The work performed with LADOHD required a manual definition of the vocabulary of events to avoid the situation in which the events observed in training do not appear in the testing set and vice versa. Although our goal is not to completely remove the intervention of humans in the process, finding general rules or heuristics for this definition would speed up the implementation and evaluation time significantly.

## REFERENCES

## REFERENCES

- [1] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, “Neural cleanse: Identifying and mitigating backdoor attacks in neural networks,” in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, p. 0.
- [2] G. Tao, S. Ma, Y. Liu, and X. Zhang, “Attacks meet interpretability: Attribute-steered detection of adversarial samples,” in *Advances in Neural Information Processing Systems*, 2018, pp. 7717–7728.
- [3] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.
- [4] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.
- [5] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” *arXiv preprint arXiv:1607.02533*, 2016.
- [6] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: a simple and accurate method to fool deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.
- [7] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.
- [8] K. Pei, Y. Cao, J. Yang, and S. Jana, “Deepxplore: Automated whitebox testing of deep learning systems,” in *proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 1–18.
- [9] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, “Adversarial patch,” *arXiv preprint arXiv:1712.09665*, 2017.
- [10] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, “Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1528–1540.
- [11] T. Gu, B. Dolan-Gavitt, and S. Garg, “Badnets: Identifying vulnerabilities in the machine learning model supply chain,” *arXiv preprint arXiv:1708.06733*, 2017.

- [12] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, “Targeted backdoor attacks on deep learning systems using data poisoning,” *arXiv preprint arXiv:1712.05526*, 2017.
- [13] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, “Trojaning attack on neural networks,” in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society, 2018.
- [14] Y. Liu, Y. Xie, and A. Srivastava, “Neural trojans,” in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 45–48.
- [15] K. Liu, B. Dolan-Gavitt, and S. Garg, “Fine-pruning: Defending against backdooring attacks on deep neural networks,” in *International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*. Springer, 2018, pp. 273–294.
- [16] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 305–316.
- [17] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “Lstm: A search space odyssey,” *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2017.
- [18] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent trends in deep learning based natural language processing,” *IEEE Computational intelligence magazine*, vol. 13, no. 3, pp. 55–75, 2018.
- [19] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep learning for computer vision: A brief review,” *Computational intelligence and neuroscience*, vol. 2018, 2018.
- [20] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, 2015, pp. 91–99.
- [21] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, “Learning deep features for scene recognition using places database,” in *Advances in neural information processing systems*, 2014, pp. 487–495.
- [22] O. M. Parkhi, A. Vedaldi, A. Zisserman *et al.*, “Deep face recognition.” in *bmvc*, vol. 1, no. 3, 2015, p. 6.
- [23] P. Sermanet and Y. LeCun, “Traffic sign recognition with multi-scale convolutional networks.” in *IJCNN*, 2011, pp. 2809–2813.
- [24] S. Ma, Y. Liu, G. Tao, W.-C. Lee, and X. Zhang, “Nic: Detecting adversarial samples with neural network invariant checking,” in *26th Annual Network and Distributed System Security Symposium, NDSS, 2019*, pp. 24–27.
- [25] M. Villarreal-Vasquez, *ConFoc Repository to be Public After Revision*, 2019. [Online]. Available: <https://github.com/mvillarreal14/confoc>

- [26] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [27] J. S. J. Stalldkamp, M. Schlipfing and C. Igel, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition,” *Neural Networks*, no. 0, pp. –, 2012.
- [28] O. M. Parkhi, A. Vedaldi, and A. Zisserman, “Deep face recognition,” in *British Machine Vision Conference*, 2015.
- [29] S. Zagoruyko and N. Komodakis, “Wide residual networks,” *arXiv preprint arXiv:1605.07146*, 2016.
- [30] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, “Cnn features off-the-shelf: an astounding baseline for recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 806–813.
- [31] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, “Decaf: A deep convolutional activation feature for generic visual recognition,” in *International conference on machine learning*, 2014, pp. 647–655.
- [32] L. A. Gatys, A. S. Ecker, and M. Bethge, “Image style transfer using convolutional neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2414–2423.
- [33] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.
- [34] B. Xu, N. Wang, T. Chen, and M. Li, “Empirical evaluation of rectified activations in convolutional network,” *arXiv preprint arXiv:1505.00853*, 2015.
- [35] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [36] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a defense to adversarial perturbations against deep neural networks,” in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 582–597.
- [37] W. Xu, D. Evans, and Y. Qi, “Feature squeezing: Detecting adversarial examples in deep neural networks,” *arXiv preprint arXiv:1704.01155*, 2017.
- [38] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” *arXiv preprint arXiv:1706.06083*, 2017.
- [39] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *arXiv preprint arXiv:1312.6199*, 2013.
- [40] S. Gu and L. Rigazio, “Towards deep neural network architectures robust to adversarial examples,” *arXiv preprint arXiv:1412.5068*, 2014.

- [41] A. N. Bhagoji, D. Cullina, C. Sitawarin, and P. Mittal, “Enhancing robustness of machine learning systems via data transformations,” in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2018, pp. 1–5.
- [42] D. Meng and H. Chen, “Magnet: a two-pronged defense against adversarial examples,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 135–147.
- [43] R. Shin and D. Song, “Jpeg-resistant adversarial images,” in *NIPS 2017 Workshop on Machine Learning and Computer Security*, vol. 1, 2017.
- [44] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, “A study of the effect of jpg compression on adversarial images,” *arXiv preprint arXiv:1608.00853*, 2016.
- [45] N. Das, M. Shanbhogue, S.-T. Chen, F. Hohman, L. Chen, M. E. Kounavis, and D. H. Chau, “Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression,” *arXiv preprint arXiv:1705.02900*, 2017.
- [46] L. Deng, “The mnist database of handwritten digit images for machine learning research [best of the web],” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.
- [47] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, 2015, pp. 91–99.
- [48] S. R. Safavian and D. Landgrebe, “A survey of decision tree classifier methodology,” *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.
- [49] I. Steinwart and A. Christmann, *Support vector machines*. Springer Science & Business Media, 2008.
- [50] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [51] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.
- [52] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, “Understanding deep neural networks with rectified linear units,” *arXiv preprint arXiv:1611.01491*, 2016.
- [53] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, “Max-pooling convolutional neural networks for vision-based hand gesture recognition,” in *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*. IEEE, 2011, pp. 342–347.
- [54] D. C. Liu and J. Nocedal, “On the limited memory bfgs method for large scale optimization,” *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.
- [55] R. A. Brualdi, H. J. Ryser *et al.*, *Combinatorial matrix theory*. Springer, 1991, vol. 39.



- [56] G. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," *Tech. rep.*, 10 2008.
- [57] B. Wang, *ConFoc Repository to be Public After Revision*, 2019. [Online]. Available: <https://github.com/bolunwang/backdoor>
- [58] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *European conference on computer vision*. Springer, 2016, pp. 694–711.
- [59] M. B. Salem, S. Hershkop, and S. J. Stolfo, "A survey of insider attack detection research," in *Insider Attack and Cyber Security*. Springer, 2008, pp. 69–90.
- [60] A. Sanzgiri and D. Dasgupta, "Classification of insider threat detection techniques," in *Proceedings of the 11th annual cyber and information security research conference*, 2016, pp. 1–4.
- [61] J. Hunker and C. W. Probst, "Insiders and insider threats-an overview of definitions and mitigation techniques." *JoWUA*, vol. 2, no. 1, pp. 4–27, 2011.
- [62] P. Chen, L. Desmet, and C. Huygens, "A study on advanced persistent threats," in *IFIP International Conference on Communications and Multimedia Security*. Springer, 2014, pp. 63–72.
- [63] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 2002, pp. 255–264.
- [64] H. Xu, W. Du, and S. J. Chapin, "Context sensitive anomaly monitoring of process control flow to detect mimicry attacks and impossible paths," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2004, pp. 21–38.
- [65] J. T. Giffin, S. Jha, and B. P. Miller, "Automated discovery of mimicry attacks," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2006, pp. 41–60.
- [66] D. Yao, X. Shu, L. Cheng, and S. J. Stolfo, "Anomaly detection as a service: Challenges, advances, and opportunities," *Synthesis Lectures on Information Security, Privacy, and Trust*, vol. 9, no. 3, pp. 1–173, 2017.
- [67] K. Xu, D. D. Yao, B. G. Ryder, and K. Tian, "Probabilistic program modeling for high-precision anomaly classification," in *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th*. IEEE, 2015, pp. 497–511.
- [68] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.
- [69] M. Agyemang, K. Barker, and R. Alhajj, "A comprehensive survey of numeric and symbolic outlier mining techniques," *Intelligent Data Analysis*, vol. 10, no. 6, pp. 521–538, 2006.

- [70] Z. A. Bakar, R. Mohemad, A. Ahmad, and M. M. Deris, "A comparative study for outlier detection techniques in data mining," in *2006 IEEE conference on cybernetics and intelligent systems*. IEEE, 2006, pp. 1–6.
- [71] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection*. John Wiley & sons, 2005, vol. 589.
- [72] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.
- [73] P. Thompson, "Weak models for insider threat detection," in *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense III*, vol. 5403. International Society for Optics and Photonics, 2004, pp. 40–48.
- [74] M. B. Salem and S. J. Stolfo, "Masquerade attack detection using a search-behavior modeling approach," *Columbia University, Computer Science Department, Technical Report CUCS-027-09*, 2009.
- [75] C. Warrender, S. Forrest, and B. A. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," 1999.
- [76] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. IEEE, 1996, pp. 120–128.
- [77] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of computer security*, vol. 6, no. 3, pp. 151–180, 1998.
- [78] D.-Y. Yeung and Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models," *Pattern recognition*, vol. 36, no. 1, pp. 229–243, 2003.
- [79] K. Xu, K. Tian, D. Yao, and B. G. Ryder, "A sharper sense of self: Probabilistic reasoning of program behaviors for anomaly detection with context sensitivity," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 467–478.
- [80] J. Hollmén and V. Tresp, "Call-based fraud detection in mobile communication networks using a hierarchical regime-switching model," in *Advances in Neural Information Processing Systems*, 1999, pp. 889–895.
- [81] P. Smyth, "Clustering sequences with hidden markov models," in *Advances in neural information processing systems*, 1997, pp. 648–654.
- [82] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White, "Visualization of navigation patterns on a web site using model-based clustering," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 280–284.
- [83] G. Salton, R. Ross, and J. Kelleher, "Attentive language models," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, 2017, pp. 441–450.

- [84] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [85] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.
- [86] M. Villarreal-Vasquez, *LADOHD Repository to be Public After Revision*, 2020. [Online]. Available: <https://github.com/mvillarreal14/ladohd>
- [87] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, “Session-based recommendations with recurrent neural networks,” *arXiv preprint arXiv:1511.06939*, 2015.
- [88] Y. K. Tan, X. Xu, and Y. Liu, “Improved recurrent neural networks for session-based recommendations,” in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 2016, pp. 17–22.
- [89] G. Kim, H. Yi, J. Lee, Y. Paek, and S. Yoon, “Lstm-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems,” *arXiv preprint arXiv:1611.01726*, 2016.
- [90] C. Feng, T. Li, and D. Chana, “Multi-level anomaly detection in industrial control systems via package signatures and lstm networks,” in *Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on*. IEEE, 2017, pp. 261–272.
- [91] M. Du, F. Li, G. Zheng, and V. Srikumar, “Deeplog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1285–1298.
- [92] Y. Shen, E. Mariconti, P.-A. Vervier, and G. Stringhini, “Tiresias: Predicting security events through deep learning,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’18. New York, NY, USA: ACM, 2018.
- [93] M. Sundermeyer, R. Schlüter, and H. Ney, “Lstm neural networks for language modeling,” in *Thirteenth annual conference of the international speech communication association*, 2012.
- [94] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, “Grammar as a foreign language,” in *Advances in Neural Information Processing Systems*, 2015, pp. 2773–2781.
- [95] K. Rocki, “Recurrent memory array structures,” 07 2016.
- [96] G. Bouchard, “Efficient bounds for the softmax function, applications to inference in hybrid models,” in *Presentation at the Workshop for Approximate Bayesian Inference in Continuous/Hybrid Systems at NIPS-07*. Citeseer, 2007.
- [97] A. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,” *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

- [98] Z. C. Lipton, J. Berkowitz, and C. Elkan, “A critical review of recurrent neural networks for sequence learning,” *arXiv preprint arXiv:1506.00019*, 2015.
- [99] S. Merity, N. S. Keskar, and R. Socher, “Regularizing and optimizing lstm language models,” *arXiv preprint arXiv:1708.02182*, 2017.
- [100] I. Loshchilov and F. Hutter, “Sgdr: Stochastic gradient descent with warm restarts,” *arXiv preprint arXiv:1608.03983*, 2016.
- [101] L. N. Smith, “Cyclical learning rates for training neural networks,” in *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE, 2017, pp. 464–472.
- [102] A. Liaw, M. Wiener *et al.*, “Classification and regression by randomforest,” *R news*, vol. 2, no. 3, pp. 18–22, 2002.
- [103] S. Kotsiantis and P. Pintelas, “Recent advances in clustering: A brief survey,” *WSEAS Transactions on Information Science and Applications*, vol. 1, no. 1, pp. 73–81, 2004.
- [104] S. Norman, J. Chase, D. Goodwin, B. Freeman, V. Boyle, and R. Eckman, “A condensed approach to the cyber resilient design space,” *INSIGHT*, vol. 19, no. 2, pp. 43–46, 2016.
- [105] J. Xu, Z. Kalbarczyk, and R. K. Iyer, “Transparent runtime randomization for security,” Tech. Rep. UILU-ENG-03-2207, 2003.
- [106] C. Warrender, S. Forrest, and B. Pearlmutter, “Detecting intrusions using system calls: Alternative data models,” in *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, 1999, pp. 133–145.
- [107] G. S. Kc, A. D. Keromytis, and V. Prevelakis, “Countering code-injection attacks with instruction-set randomization,” in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, 2003, pp. 272–280.
- [108] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, “On the effectiveness of address-space randomization,” in *Proceedings of the 11th ACM Conference on Computer and Communications Security*, 2004, pp. 298–307.
- [109] B. Bhargava, P. Angin, R. Ranchal, and S. Lingayat, “A distributed monitoring and reconfiguration approach for adaptive network computing,” in *Proceedings of the 2015 IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW)*, 2015, pp. 31–35.
- [110] K. Kirkpatrick, “Software-defined networking,” *Communications of the ACM*, vol. 56, no. 9, pp. 16–19, Sep. 2013.
- [111] M. H. Marghny and A. I. Taloba, “Outlier detection using improved genetic k-means,” *International Journal of Computer Applications*, vol. 28, no. 11, pp. 33–36, August 2011.
- [112] L. M. Manevitz and M. Yousef, “One-class svms for document classification,” *Journal of Machine Learning Research*, vol. 2, pp. 139–154, Mar. 2002.

- [113] T. Garfinkel and M. Rosenblum, “A virtual machine introspection based architecture for intrusion detection,” in *Proceedings of the Network and Distributed Systems Security Symposium*, 2003, pp. 191–206.
- [114] DHS, “Moving target defense,” <https://www.dhs.gov/science-and-technology/csd-mtd>, Accessed Feb. 2017.
- [115] N. Ahmed and B. Bhargava, “Mayflies: A moving target defense framework for distributed systems,” in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, 2016, pp. 59–64.

VITA

## VITA

Miguel Villarreal-Vasquez received his B.S. degree in Electronics and Communications Engineering from University of Panama, Panama in 2006. He obtained his M.S. degree in Information Security at Purdue University in 2014. After completing this degree, he joined the Department of Computer Science as a Ph.D. student in the same institution, where he worked as a teaching and research assistant. He completed the Ph.D. degree under the supervision of Professor Bharat Bhargava in August 2020. His research interests are building resilient security systems using machine learning, with special attention to deep learning. His research also embraces improving the resiliency of deep learning architectures against adversarial settings.

During his Ph.D. studies, Miguel joined multiple companies as a research assistant intern. In 2018, he worked for the Center for Advanced Machine Learning (CAML) at Symantec, Mountain View CA, developing an anomaly detection system based on LSTM models to process high dimensional sequential data. In 2015 and 2017, he joined the Content-Protection group at IBM-Almaden Research, San Jose CA, where he developed decentralized applications using Blockchain technologies. Miguel also worked at Delphi Electronics and Safety, Kokomo IN, in 2014 right after his Master's, where he designed security mechanisms implemented in new infotainment systems.

Additionally, his Ph.D. research projects represented Purdue University in the Northrop Grumman Cybersecurity Consortium (NGCRC) for three years. The performed research received the CERIAS Symposium Best Poster Award in 2019.