This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2023-11339J

Machine Learning Based Resilience Testing of an Address Randomization Cyber Defense

Ganapathy Mani*(1), Marina Haliem*(1), Bharat Bhargava(1), Indu Manickam(2), Kevin Kochpatcharin(1), Myeongsu Kim(1), Eric Vugrin(2), Weichao Wang(3), Chris Jenkins(2), Pelin Angin(4), Meng Yu(5)

Abstract—Moving target defenses (MTDs) are widely used as an active defense strategy for thwarting cyberattacks on cyber-physical systems by increasing diversity of software and network paths. Recently, machine Learning (ML) and deep Learning (DL) models have been demonstrated to defeat some of the cyber defenses by learning attack detection patterns and defense strategies. It raises concerns about the susceptibility of MTD to ML and DL methods. In this paper, we analyze the effectiveness of ML and DL models when it comes to deciphering MTD methods and ultimately evade MTD-based protections in real-time systems. Specifically, we consider a MTD algorithm that periodically randomizes address assignments within the MIL-STD-1553 protocol—a military standard serial data bus. Two ML and DL-based tasks are performed on MIL-STD-1553 protocol to measure the effectiveness of the learning models in deciphering the MTD algorithm: 1) determining whether there is an address assignments change i.e. whether the given system employs a MTD protocol and if it does 2) predicting the future address assignments. The supervised learning models (random forest and k-nearest neighbors) effectively detected the address assignment changes and classified whether the given system is equipped with a specified MTD protocol. On the other hand, the unsupervised learning model (K-means) was significantly less effective. The DL model (long short-term memory) was able to predict the future addresses with varied effectiveness based on MTD algorithm's settings.

Index Terms—Moving Target Defense, Long-Short-Term-Memory, Cyber Security, Cyber-attacks, Cyber Resilience, Supervised Learning, Unsupervised Learning, Random Forest

1 Introduction

Moving target defenses (MTD) are adaptive defense strategies that have been proposed to protect information and cyber-physical systems against cyber attacks. MTDs periodically change key features of the system in a seemingly random manner. The dynamic and relatively unpredictable attack surface created by MTDs can potentially change, obscure, or hide system features that an attacker may try to learn during reconnaissance activities [1]. Ultimately, the uncertainty created by MTDs increases the effort and sophistication required to successfully execute cyber attacks. Numerous MTD approaches have been developed to protect networks against attacks [2], [3], [4], [5], [6]. For instance, operating system (OS) randomization is used for protecting the host's functionality by periodically rotating the host's OS for implementing mission-critical tasks [7]. This approach can prevent the attacker from designing OS-specific attacks to impede critical processes. Similarly, address space randomization, assembly instruction randomization, and code sequence randomization are forms of MTD for defending against buffer overflow attacks [8]. Buffer overflow attacks can act as a base for return oriented programming (ROP) and code injection attacks.

MTD algorithms have been proposed primarily for information technology (IT) networks, but MTDs are increasingly being explored for use in cyber-physical, real-time systems [9], [10]. Real-time systems include constraints that are typically not found in general-purpose computing systems and pose challenges to MTD implementations. For example,

1) Real time systems may be resource-constrained and may be limited to a small number of nodes and system features. The networking address space for real-time systems tend to be much lower than best-effort systems. For example, the MIL-STD-1553 protocol has 31 possible addresses while IPv4 has 4.3 billion [11].
2) Changing of system features must be done in a pseudo-random and not truly random manner for explainability of system actions. Previous work [12] has demonstrated this limitation can be exploited and randomization patterns can be learned through a limited number of observations.
3) Real-time systems tend to be size, weight, and power (SWaP) constrained. Hence, MTD implementations must be computationally effective and efficient.

These challenges can become opportunities for attackers to learn MTDs and defeat them. Machine learning (ML) and deep learning (DL) techniques are commonly used to identify patterns in data, so attackers could potentially apply ML/DL techniques to decipher MTD algorithms and to execute attacks against systems protected by MTDs.

In this paper, an MTD algorithm that is implemented using the real-time MIL-STD-1553 protocol is considered. The MTD algorithm randomizes node address assignments periodically. The timing of address changes and new addresses are passed via messages between nodes. Given

* *Equal Contribution.*
(1) *Dept. of Computer Science, Purdue University, West Lafayette, IN 47906*
(2) *Sandia National Laboratories, Albuquerque, NM 87185*
(3) *Software and Information Systems Department, University of North Carolina at Charlotte, NC 28223*
(4) *Dept. of Computer Engineering, Middle East Technical University, Turkey*
(5) *Dept. of Computer Science, Roosevelt University, Chicago IL 60605*

This article has been authored by an employee of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employee owns all right, title and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan https://www.energy.gov/downloads/doe-public-access-plan.

Sandia National Laboratories
U.S. DEPARTMENT OF ENERGY NNSA

the challenges in implementing MTDs, the paper's unique contributions are identified as the following:

- We assess the ability of both supervised and unsupervised learning-based methods to perform two tasks: 1) classify messages as either MTD address updates or non-MTD address updates and 2) predict the next address assignment.
- We show that the supervised learning approaches that we considered effectively classified MTD address updates, but the unsupervised method struggled to do so.
- We also demonstrate the ability of the DL model (LSTM) used to predict future addresses had mixed success and was greatly affected by the manner in which the MTD selected future addresses.
- In addition, we quantify the volume of data required by each model to effectively perform these tasks. In many instances, only limited training data was needed for accurate predictions.

The remainder of this paper is organized as follows: Section 2 provides literature review on MTDs and details the MIL-STD-1553 protocol. Section 3 describes a MTD implementation in the MIL-STD-1553 protocol. Section 4 describes the threat model and ML/DL models and tasks used for learning the MTD algorithm. Section 5 presents the research questions, experiments conducted to investigate those questions, datasets that are used for training the models, methodology of experimental approach, and the results of those experiments and analysis. Finally, we conclude the paper in section 7.

Abbreviations and Acronyms: MTD: Moving Target Defense. ML: Machine Learning. SVM: Support Vector Machine. LSTM: Long Short Term Memory. PRNG: Pseudo-Random Number Generator. RT: Remote Terminal. CO: Current Offset. SO: Static Offset.

2 Background

2.1 Moving Target Defense (MTD)

MTDs create dynamic, seemingly uncertain environments which seek to confuse attackers conducting reconnaissance in an attempt to defeat cyber threats. There are numerous implementations have been proposed in literature [13], [14], [15], [16]. Attackers attempting to perform network discovery, exfiltration of data, and other malicious activities will observe that key system attributes change in a seemingly chaotic and random manner. These changes are choreographed among the system components in a methodical way such that they know when and how the changes will occur [17].

Implementation of MTDs can vary broadly across applications [18]. For example, a webserver may change its IP address every day to a different and randomly selected address to thwart an attacker's attempt to hack the webserver. Another option is to randomize the port at which the webserver listens for requests. The specific features of the system that change will depend on the system as well as features and actions of the threat [18]. Similarly, there are wide-ranging shuffle-based MTD techniques such as virtual IP shuffling [19], [20] and Port hopping [21], [22].

Most MTD applications have been designed for general purpose computing systems [23]. The research work presented in this paper considers an MTD algorithm that is applied to a real-time network. The usage of the MTD algorithm provides a potential mechanism for increasing resilience against attacks on the systems that use these protocols [24]. Thus, any application of MTDs to these systems faces the added technical challenges of having to adhere to stringent determinism, reliability, predictability, and real-time requirements.

2.2 MIL-STD-1553 Protocol

MIL-STD-1553 is a military standard for transmitting data on a reliable data bus [24]. Most implementations use a dual redundant bus in failover mode to improve reliability. MIL-STD-1553 has heavy use in military and avionics systems as well as usage in space systems. The protocol was standardized in 1970s, and, due to its reliability, it has remained a staple command and control network.

The MIL-STD-1553 bus can include the following three types of devices: a single bus controller (BC), up to 31 remote terminals (RTs), and a bus monitor (BM). All communication and data transfers on the bus are initiated by the BC. The BC sends commands that are directed to all or individual RTs. RTs will consume the information and either remain silent or send data back to the BC. The BM remains silent and records all communication sent on the bus.

In the MIL-STD-1553 protocol, a *message* is a single data transfer. MIL-STD-1553 has two types of messages: unicast (i.e., non-broadcast) and broadcast. As the names imply, unicast messages result in communication between two devices (a sending device and a single receiving device), and broadcast messages result in communication between a sending device and all receiving devices on the network. Typically, these messages are organized into frames which repeat at a particular frequency.

A unicast message will contain the address of the device to which it is directed. In addition, the message will contain a sequence of one or more of the following data types, depending on which device is initiating the message (BC or RT): command word (CW), data word (DW), and status word (SW). Each word type is 16 bits (2 bytes) in length. Command words are only sent by the BC, and they command the RTs. Both the BC and RT send DWs based on the CW sent by the BC. A single message can include up to 32 DWs. Status words are sent only by RTs and inform the BC of correct reception of the CW and DWs (if present). Each bit on the bus consumes one microsecond, and the bus operates at 1 MHz. Each word consists of a 3-microsecond sync, 16 bits (16 microseconds), and one parity bit (1 microsecond), so approximately 20 microseconds are required to send a single word.

The MIL-STD-1553 protocol does not include authentication or encryption features. The lack of these protections has raised concerns about cyber attack risks for systems using this protocol.

2.3 Machine Learning

Applied machine learning is a fast growing field and has been shown to be useful in a wide range of application domains [25]. Traditionally in the cybersecurity domain, ML has been used for anomaly detection [26], classification of spams [27], cyber threats detection [28], attack's next

phases/steps prediction, and attacker/attack behavior modeling [29]. ML is particularly effective in detecting network attacks [30], DDOS attacks [31], [32]. The survey in [33] provides a comprehensive study of several ML techniques that are effective in detecting phishing attacks. ML models have proven to be effective for malware detection [34] and identification of large scale attacks [35]. Even in hardware domain, particularly in embedded systems and other size, weight, and power (SWaP)-constrained systems, learning models are being increasingly leveraged. Significant improvements in low-cost DL models have been demonstrated on small mobile devices [36]. Deep learning on mobile devices with the inclusion of graphic processing units (GPUs) and neural network architectures have been used for text, image, and speech recognition tasks [37]. The development of accelerators [38] and dedicated DL libraries for on-device DL (e.g. tensorflow-lite), have made it possible to carry out ML classification and prediction on mobile hardware.

ML is not only being leveraged for cyber defense but also to attack systems by learning about the defense behaviors [29]. Hence, cyber defenses must be evaluated to determine their effectiveness against ML-based attacks. As such, this paper is interested in determining if the specified MTD algorithm is vulnerable to ML-based methods.

2.4 Applied ML in Embedded Protocols

Attempts have been made to improve the security of MIL-STD-1553 using a sequence-based anomaly detection methodology. Stan et al. used Markov models to detect unauthorized data transmissions and spoofing attacks on a MIL-STD-1553 data bus [39]. The authors also characterize the bus by learning what messages are sent and the frequency of such messages using classification models such as random forest and K-means. Genereux et al. proposed a histogram comparison approach and time-based features for detecting spoofing attacks, where only data words are manipulated [40]. Their method built upon and improved the histogram-based bus profiling method for anomaly detection, proposed by Losier et al. [41].

Some of the recent research works have examined MTD and other real-time protocols. The controller area network (CAN) bus has a similar network topology as those used with MIL-STD-1553. The CAN ID Shuffling Technique (CIST) utilizes encryption and hashing to produce a sequence-dependent shuffling of CAN IDs on a per-message basis [42]. For each message, up to two Hash-based Message Authentication Codes (HMACs) and one encryption is needed. The technique requires a variety of different crypto-variables: group keys, one-time keys, seeds, encryption keys, etc. With all the aforementioned keys and algorithms, the authors devised a scheme where the CAN ID can change per message (typically CAN IDs do not change).

Brown et al. suggested a model for CAN bus protocol, which uses a seed value and dynamic state called dynamic address validation array [43]. In their scheme, each engine control unit (ECU) invalidates its CAN ID when needed, and the other devices use an algorithm to synchronize the new CAN ID for the respective node. As with CIST the CAN IDs change but without the accompanying state and algorithm complexity.

Maintaining the system and security properties under cyberattacks through MTD is proposed in [44]. The authors consider the problems of detecting code injection and code reuse attacks and solve them through the MTD techniques: Address Space Randomization (ASR) and Instruction Space Randomization (ISR). But the solution only applies to resource-heavy cyber-physical systems. In this research, we should that ASR-based MTD defense can be thwarted by ML/DL based attacks.

3 MIL-STD-1553 MTD Implementation

3.1 Overview

If a system is employing the MIL-STD-1553 protocol, an adversary listening on the bus would have access to all communication between the BC and RTs. The focus of the MTD strategy presented in this paper is to mitigate threats due to this vulnerability by introducing a dynamic address scheme where each RT on the bus changes its address in real-time in response to a command from the BC. Use of MTD can mitigate threats from the adversary in two ways: 1) the adversary will have difficulty targeting malicious commands and data requests towards a particular RT, and 2) reconnaissance activity which requires an adversary to track messages sent to particular devices will be more difficult to conduct, particularly in real-time. In response to implementing MTD, the adversary could also collect data for offline processing and employ machine learning algorithms on the collected data to attempt to predict the pattern of changes. Given that only 31 addresses are are available in the MIL-STD-1553 protocol, the algorithm design must employ sufficient entropy to challenge the attacker and any ML algorithms they may deploy.

An MTD system is developed which can synchronize address updates across all participating devices on the network as long as each device has the correct internal MTD state (further detailed in this section). Note that while a summary of the algorithm is presented here, refer to [10] for more details on the MTD algorithm.

3.2 MIL-STD-1553 MTD Constraints

Use of the MTD algorithm within a real-time protocol requires certain design features to address specific constraints. These key constraints are as follows:

1) **Keep underlying protocol** - These protocols have been designed around the need for deterministic and reliable real-time communication. By using the existing protocol, this avoids introducing effects which detract from the protocol's benefits.
2) **Dynamic address generation** - Each node in the network should use the same MTD algorithm to produce the same state. Each node has to use disjoint addresses as compared to all other nodes in the network.
3) **Synchronization** - All MTD participating RTs on the network must be synced up during address changes/shifts. Synchronization prevents different RTs from sharing the same address at the same time. Due to the nature of these networks, extended loss of synchronization could have catastrophic consequences.
4) **Authenticity** - The devices on the network should only listen to authentic MTD commands. Using analog

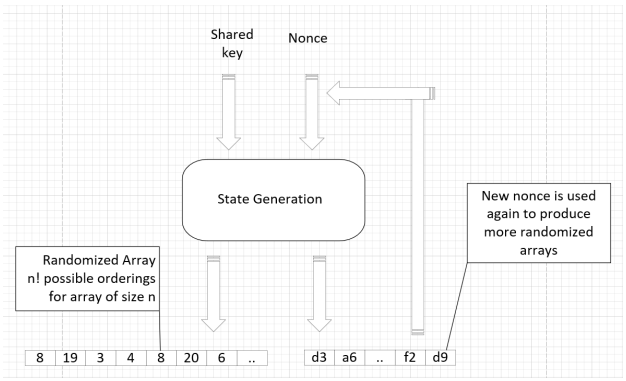


Fig. 1: State Generation Architecture

attributes, message authentication codes, message integrity checks, etc., with the MTD command provides an avenue to ensure only valid MTD commands are followed.

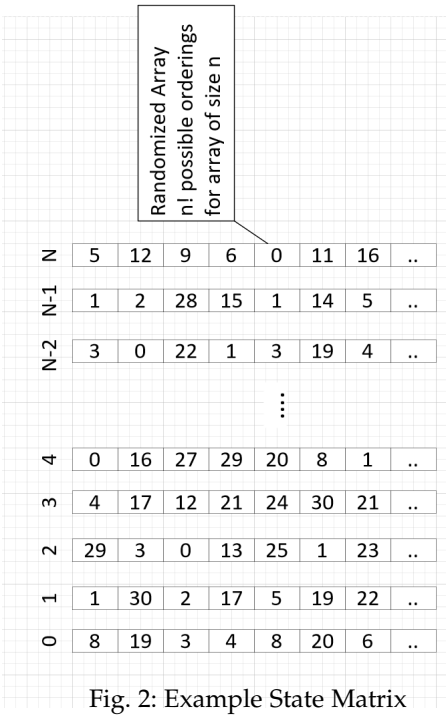


Fig. 2: Example State Matrix

3.3 MTD Algorithm

The MTD algorithm requires each device on the bus to have the following four key architectural components: 1) secret key, 2) nonce, 3) process to generate the state matrix, and 4) process to select an address from the state matrix that is unique for each RT on the bus.

The shared key should be shared only between participating MTD devices. The nonce is used by the MTD devices but does not necessarily have to be kept secret.

In order for the address selection algorithm to work, each RT on the bus must generate the same two-dimensional state matrix upon receiving a command from the BC. The state matrix has 31 rows (one per possible device address), and a configurable number of columns.

Each column of the matrix consists of a random ordering of possible addresses. As shown in Fig. 1, the columns of the matrix are iteratively generated. For each iteration, the state matrix generation process takes as input the shared key and a nonce, and outputs: 1) an array that is concatenated as a new matrix column and 2) a new nonce. The updated nonce is used as input to generate the next state matrix column using the same process. To generate the device addresses in each matrix column, the state generation process uses a pseudorandom number generator (PRNG). PRNGs utilize a variety of underlying algorithms to generate random bytes. An example state matrix array is shown in Fig. 2.

RTs update their address after receiving a command from the BC with an index number. The BC generates this index number pseudorandomly using the *rand()* function from the C standard library. The index value indicates which column index in the state matrix devices should access in order to find the updated address. Each individual device selects a row of the matrix, which we also refer to as the offset into the column, to determine which cell in state matrix contains their updated address value. We present two methods for devices to determine their unique offset values after receiving an update command from the BC, *static offset* (SO) and *current offset* (CO).

3.3.1 Static Offset

With the SO method, each device is assigned a unique, fixed offset value prior to beginning to receive address assignments. For example, using the state matrix shown in Fig. 2, if a device has a fixed address of 5, and received an index value of 1 from the BC, its new address value will be 19. If it receives an index value of 3, its new address will be 30. And if in the future that device receives the index value 1 again, its updated address will be 19 again.

3.3.2 Current Offset

With the CO, each device will have a different offset for each address update that is based upon its current cell value. The devices will also be assigned to a unique offset for the initial update. For example, referring to Fig. 2, if a device is initialized with offset 2, and receives index 2 from the BC, its first address will be column 2, row 2 in the state matrix, which has address value 0. The value 0 will be its new offset value. If next it receives index value 3 from the BC, the updated address will be 4, and 4 will also be its offset for the next update. Because each device is assigned a unique initial offset, and because the state matrix columns were generated so that each cell within the column has a unique value, no two devices will be assigned the same value upon receiving an update. The correct operation of the MTD has been verified in [10]

3.4 MIL-STD-1553 Integration

As listed in the design constraints, it is essential that the MTD algorithm keeps the underlying MIL-STD-1553 protocol. To integrate the MTD algorithm into the existing protocol, the BC can issue four different commands, which we will refer to as MTD commands, based on the existing MIL-STD-1553 message structure. The commands are as follows:

1) **MTD Start** – this command sends a 128-bit nonce, the size of the state to generate, offsetting mechanism, and

any addresses which are not permitted for use in the state matrix.
2) **MTD Verify** – this command sends a 256-bit keyed hash message authentication code (HMAC) to the requester. Use of this command enables the host node (e.g., BC) to verify all participating nodes have the same state without leaking information.
3) **MTD Update** – this command sends an index value as a single data word which participating nodes use to select their new address. In our experiments, the index value selects the state column.
4) **MTD Stop** – this command instructs all participating nodes to stop listening to MTD Update commands. The nodes can either revert to their initial address or stay at their current address. The host node would indicate this choice in the MTD Start command.

The number of command and data words is minimized to limit the impact on existing MIL-STD-1553 designs. As stated before, these commands could be spoofed. A 256-bit HMAC (16 DWs) is suggested when sending any MTD commands for this design. This approach would increase the amount of time the MTD Update command consumes on the bus. A variety of schemes exist to construct an appropriate message authentication code (MAC) and are outside of the scope of this paper.

3.5 Analysis of the MTD Algorithm Unpredictability

An analysis was conducted in [10] on the unpredictability of the address assignments produced by the MTD algorithm. In this study, a set of metrics were applied to sequences of address assignments produced by the MTD algorithm under different system configurations. Each metric measured unpredictability using a different set of criteria, and the set of metrics included Shannon's entropy, Lempel-Ziv complexity, sample entropy, SVD entropy, permutation entropy, spectral entropy and characteristics of the address transition matrix such as the entropy, uniformity, and turbulence score.

This study determined that the use of the CO resulted in statistically significant higher scores than use of the SO. In addition, increasing the number of columns in the state matrix produced a statistically significant increase in scores across a subset of the metrics, though this impact diminished as the number of state matrix columns increased.

4 Machine Learning Tasks and Implementation

Network discovery and reconnaissance are essential steps in a cyber attack and most attackers use these steps to observe and understand the system and its defenses [45]. With these observations, attackers can identify vulnerabilities, select targets, and design several fail-safe attack sequences to penetrate the system. This study investigates these initial stages of an attack and evaluate whether an attacker using ML/DL models can: (a) detect whether a MTD protocol is actively defending the system and (b) if MTD is detected, predict the next random choice of system configuration (in our case, random address) to launch an attack. The adversarial model is designed based on these principles.

4.1 Adversarial Model

The adversary is assumed to have established a presence on one of the RTs in the MIL-STD-1553 bus due to a supply chain insertion or some other means. All messages sent on the bus are observable to all of the RTs due to the nature of the MIL-STD-1553 protocol, and, hence, the adversary can capture all traffic on the bus. In addition, the adversary understands the MIL-STD-1553 protocol, MTD algorithm, the 4 MTD commands as well as the different interpretations of the 16-bit unsigned value in the MTD update command. The adversary is able to observe the particular node of interest and figure out its previous address.

The goal of the adversary is to:
1) Observe and capture traffic on the bus.
2) Train ML/DL models with minimal amounts of the captured traffic data.
3) Determine whether the MTD protocol changes the system's address by performing classification tasks on the system.
4) Determine the next address based on the DL model's prediction.

4.2 MTD Classification Task

For this binary classification task, each message captured on the bus is labeled as an MTD (generated by a MTD protocol) or a non-MTD message. A non-MTD message is a regular transmission/reception without MTD protocol interruption i.e. non-MTD message is one that is not participating in the MTD protocol; instead the message is a simple *transmit* or *receive* message. Specifically, a non-MTD message does not contain any of the 4 MTD command types explained in Section 3.4.

4.2.1 Data Pre-processing

The following steps are used to pre-process the data for the MTD classification task:
1) Messages from a MIL-STD-1553 bus using the MTD algorithm were collected and stored as a collection of hexadecimal strings (messages). These data items are stored in a CSV file and each row corresponds to a different message.
2) Each message is split into its components (such as MTD commands, timestamp, and data bits) and each of these hexadecimal string words are converted into decimal string words.
3) Input Data: the only information used for training for each row sample is: (1) timestamp, (2) index, (3) MTD CW, and (4) MTD SW.
4) Data is normalized using min-max scaling. For each row X, the normalized form is $X' = \frac{X - X.min()}{X.max() - X.min()}$.
5) Each row is labelled as either MTD or non-MTD.
6) Finally, the training data is balanced using oversampling of the under-represented labels. Oversampling is preferred in order to avoid the loss of valuable information by undersampling the majority class.

4.2.2 Data Labeling

To label the data for the classification task, the ground truth is determined based on the history of MTD as well as prior knowledge of the collected data that is generated along the serial bus. In addition, by analyzing the data, the frequency of occurrence of MTD update command in messages is determined. The messages are labelled with MTD commands as MTD and other messages as non-MTD

i.e. binary classification labels. The labels are converted into a categorical vector of [0, 1] and [1, 0] for training purposes. After the over-sampling step, the balanced and labeled dataset of MTD and non-MTD messages used to train the machine learning classifiers.

4.2.3 Model Selection and Training

Feature engineering techniques [46], [47] are employed to determine the best features and we perform parameter optimizations [48] to select the models with high classification scores (F1 score, False Positive Rate, and False Negative Rate. Performance metrics and results are discussed in section 5.4). Based on the feature engineering and hyperparameters optimization results, the following supervised and unsupervised learning models are selected to cross-verify the classification task and to provide better quantifiable confidence to the classification task.

1) K-nearest neighbor (KNN): KNN is a supervised learning model that classifies objects based upon the majority vote of its neighbors [49]. The model is optimized for the number of neighbors n. KNN was chosen as it has been employed as an efficient mechanism for machine learning based attacks and resiliency analysis [50], [51], [52]. The model has shown to be effective in learning and classifying defensive patterns.
2) Random Forest (RF): RF consists of numerous decision trees and uses the classification results of all decision trees to make predictions [53]. Ensemble techniques are powerful as they can learn using an army of models and classify based on collective consensus. RF has shown to be an effective attack strategy for inference attacks [54], [55], [56] and thus selected for this paper.
3) K-means clustering: Unsupervised learning methods do not depend on ground truth labels [57]. This algorithm was selected to investigate how effective are the unsupervised learning models that lack ground truth data in classifying MTD messages. K-means was selected as a performance comparison model with supervised learning models as the distribution of the data is random and it is hard to form spherical clusters given the random distribution.
4) Gaussian Mixture Model: To investigate beyond the spherical clustering assumption in K-means, we experiment with Gaussian Mixture Models (GMMs). GMMs fit the data to a mixture of Gaussian distributions [58], so it can deal better with data distribution that doesn't consist of spherical clusters.

4.3 Address Prediction Task

For the address prediction task, the DL model's effectiveness to predict an RT's next address assignment, after an MTD update command is issued, is evaluated. The DL algorithm is provided with one, two, or three of the RT's previous addresses as labels. In this study, the effect of providing additional addresses has on predicting the next address is analyzed.

In the MTD message structure, addresses are embedded in the CW using 5 bits. Address labels range from 0 to 30 (Address 31 is reserved for broadcast messages in the MIL-STD-1553 protocol.) Consequently, the the address prediction task is a multi-class (31 classes in this case) classification task.

4.3.1 Data Pre-processing

The following steps are used to pre-process the data for the address prediction task:

1) Messages from a MIL-STD-1553 bus using the MTD algorithm were collected and stored as a collection of hexadecimal strings (messages). These data are stored in a CSV file, and each row corresponds to a different message.
2) Each message is separated into its CW strings.
3) Convert the CW into a bit string representation of 0s and 1s.
4) Extract the address number from the MTD message that follows the MTD update command. (The address change only follows a message with MTD update command.)
5) For each of these messages, we extract the addresses' 5 bits from the CW as its current address. For each message in row i, the current address is denoted as A_0^i.
6) Since we formulate our task to predict the next address given the previous n addresses ($n = 1, 2,$ or 3), for each row i, we add the relevant n columns of previous n addresses as: A_0^{i-j} and generate the dataset.
7) The dataset is normalized using min-max scaling, similar to the MTD classification task.
8) The only information used for training in each row sample is: (1) timestamp, (2) index, (3) previous n addresses where $n = \{1, 2, 3\}$.

4.3.2 Data Labeling

We label each row i as A_0^{i+1}, which represents the true *next address* for this row after the next MTD update command. This approach is a multi-class prediction task with 31 unique address classes. The generated training datasets will now include messages with n previous addresses to train on and 31 discrete *next address* labels. To ensure that the datasets have balanced distributions of each label, oversampling of under-represented labels is performed. This results in a dataset of MTD update commands along with the previous addresses of the target RT prior to the update, which are labeled appropriately with the next address of the target RT following the update. The dataset is balanced between all 31 possible class labels and used for training the model for n experiments. Each experiment will only consider n previous addresses to predict the next address of the RT, where $n = \{1, 2, 3\}$.

4.3.3 Model Selection and Training

Neural networks have been shown to be efficient in training and effective in classification for machine learning based attacks [59], [60], [61]. For the address prediction task, we considered recurrent neural networks (RNNs), which are capable of training on sequences of address values. As the complexity of the MTD address randomization increases, learning models need to be capable of remembering long-term dependencies in data sequences. In sequence-to-sequence prediction tasks (in our case, message sequences to predicting next one, two, or three address sequence), standard RNNs are shown to be effective but they are limited by vanishing gradient problem [62]. Long Short-Term Memory (LSTM) models can effectively solve the vanishing gradiant problem by remembering long-term dependencies [63], [64].

Compared to a standard RNN structure, the LSTM model adopts three additional gates: input, output and forget.

The forget gate is used to prevent saturation of cell status by deciding whether received information will be discarded or not. To make the decision, the forget gate computes:

$$f_t = \sigma(W_f \cdot [H_{t-1}, X_t] + Bias_f) \quad (1)$$

In the equation (1), σ is the *Sigmoid* function, H_{t-1} is the output (f_{t-1}) of the previous hidden node, X_t is the current input value, and W is the randomly initialized weight and adjusted using backpropagation. When the computed f_t value is 1 (or a specified threshold), the cell state keeps the information. Otherwise, the information is discarded from the cell state. The input gate consists of two parts, and it is used to determine amount of new information which will be updated into the cell state. The first part computes information to be updated into the state using the following equation:

$$i_t = \sigma(W_i \cdot [H_{t-1}, X_t] + Bias_i) \quad (2)$$

The second part generates a vector C'_t which contains new candidate information using tanh function and updates the information into the state:

$$C'_t = tanh(W_C \cdot [H_{t-1}, X_t] + Bias_C)\ C_t = f_t \cdot C_{t-1} + i_t \cdot C'_t$$

Finally, the output of the cell state is computed as follows by the output gate:

$$o_t = \sigma((W_o \cdot [H_{t-1}, X_t] + Bias_o)\ H_t = o_t \cdot tanh(C_t)$$

Unlike the standard RNN that fails to learn the presence of time lags greater than 5 – 10 discrete time steps between relevant input events and target signals, LSTM is not affected by this problem by adopting this reinforced structure [65]. LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing constant error flow through "constant error carrousels" (CECs) within special units, called cells [65]. It makes learning of LSTM more robust on both long-term and short-term sequential data. This is a behavior required in complex problem domains like machine translation, speech recognition, and text autocorrection [66]. These models include memory gates to remember information for arbitrary amounts of time. In this paper, we utilize LSTM models to predict the next address(es) of devices attached to a network implementing the MTD protocol. For this purpose, the last layer of LSTM executes a *softmax* function that condenses the output to the required range of multi-class classification (which is 0-30 in our case, to accommodate all 32 possible address labels).

5 Experiments and Results

5.1 Research questions

The effectiveness of the MTD algorithm relies on the notion that the algorithm contains enough entropy and randomness such that identifying when does an address change occur and predicting the next address are difficult. Hence, we have posed the following research questions:

1) How effectively can the ML models classify messages that contain the MTD update command?
2) How does the effectiveness of the MTD classification vary when we change:
   - the number of messages in the training data?
   - address selection mechanism from SO to CO in the MTD algorithm?
   - the period between MTD updates in the MTD algorithm?
3) How effectively can the LSTM method predict an RT's next address after an MTD update message is sent?
4) How does the effectiveness of the address prediction vary when we change:
   - the number of messages in the training data?
   - the number of previous addresses that the LSTM method can inspect?
   - address selection mechanism from SO to CO in the MTD algorithm?

5.2 Data

5.2.1 Application Scenario

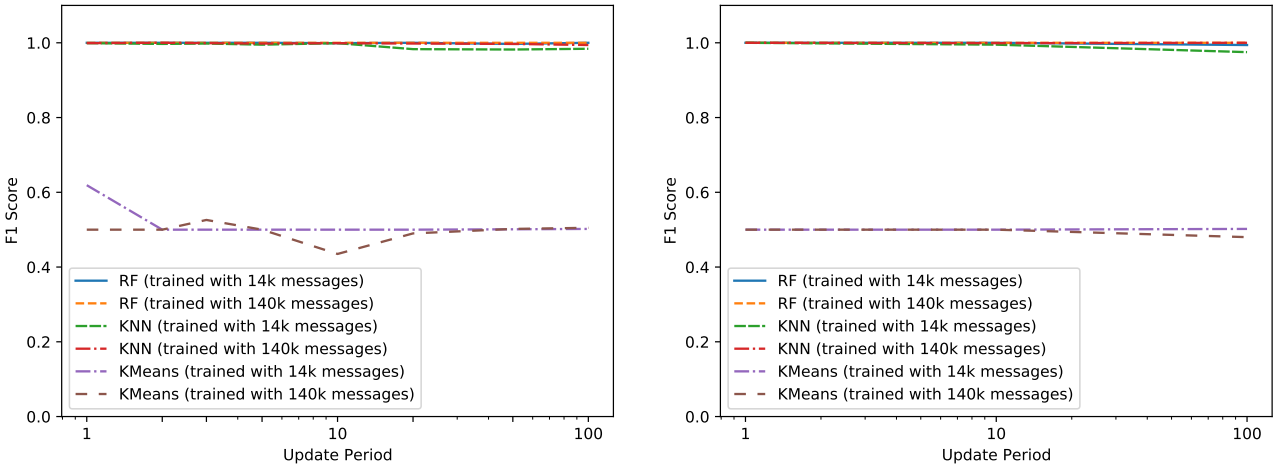
The datasets used for analysis were generated from real-hardware using the MTD protocol. The experimental setup consisted of a bus with one RT, one BC, and one BM. The BC sent MTD updates at a specified frequency as broadcast messages to the RT, and the RT updated its address as specified by the MTD algorithm.

To simulate real-world traffic on a control system, the RT and the BC communicated a series of data messages to each other. For this particular application, the BC sent a command request with two numbers, and the RT responded with a status message and a data message containing the sum of the numbers. Using this setup up, the devices communicated the first 24 values of the Fibonacci sequence. That is, the BC would send the numbers 0 and 1, and the RT would respond with 1. Then the BC would send the numbers 1 and 1, and the RT would respond with 2, etc. The process would occur in a loop, restarting after the 24th Fibonacci number was calculated. We refer to the BC request with the two numbers and the RT response with the sum as a *Fibonacci pair*. Thus, the messages passed over the bus were a mix of data messages and MTD update commands. The MTD update messages were sent periodically with a fixed number of Fibonacci pairs between update commands, which we refer to as the *update period*.

Note that most operational configurations involving a bus running the MIL-STD-1553 protocol generally include several more RTs. We selected a single RT for the simplicity of the experimental setup. However, we do note that use of only a single RT likely resulted in better MTD classification and address prediction results than would have occurred in a setup that included several RTs. An operational bus deployed in realistic conditions would likely contain a larger variety of communication messages that would likely add more noise to the data. In addition, with a single RT it is easy to follow the sequence of address assignments assigned to a particular device, while this information may be harder to extract in a setting with multiple RTs that are simultaneously updating their addresses.

5.2.2 Data Sets

Using the test setup described in Section 5.2.1, the communication on the bus was captured by the BM and later converted into CSV files where each row represents a single



(a) F1 Score of Classification Models on Static Offset Dataset using either 8% or 80% of the training samples.

(b) F1 Score of Classification Models on Current Offset Dataset using either 8% or 80% of the training samples.

Fig. 3: Performance Metrics of Three Models for MTD Classification Task with varying MTD update frequency

message. Each message component is encoded as a separate column, and includes the timestamp, the command word, the status word, and up to 31 data words. Any components not included in a particular message, for example a message sent by the BC will not contain a status word, were set to a hexadecimal constant, *0xffff*.

Because each experiment used the same nonce, secret key, and hash seed for the PRNG, the data sets produced for each experiment were based off of the same state matrix. The state matrix contained 128 address columns. Separate datasets were generated for MTD experiments in which the CO and SO indexing methods were used for MTD updates. The update period was also set at a fixed value per experiment, and varied between 1 (every other message was an MTD Update) and 100 (every 100^{th} message was an MTD update).

5.3 Performance Metrics

To evaluate the prediction performance, we measured these fundamental metrics:

- Precision $= \frac{TP}{TP+FP}$: to quantify the number MTD predictions that actually are MTD messages.
- Recall $= \frac{TP}{TP+FN}$: to quantify the number of MTD predictions made out of all MTD examples in the dataset.
- Accuracy $= \frac{TP+TN}{TP+FP+TN+FN}$
- $F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$,

where

- $TP =$ number of true positives: how many times out of the total number of messages did our ML algorithm detect/hit an MTD message (by classifying it to an MTD message)?
- $FP =$ number of false positives: how many times out of the total number of messages did our ML message miss an MTD message (by mistakenly identifying it to be a non-MTD message).
- $TN =$ number of true negatives: how many times out of the total number of messages did our ML algorithm detect/hit a non-MTD message (by classifying it to a non-MTD message)?
- $FN =$ number of false negatives: how many times out of the total number of messages did our ML algorithm miss a non-MTD message (by mistakenly identifying it to be MTD).

Because the F1-score is the harmonic mean of precision and recall, the F1-score intuitively presents a balanced view of the performance of the ML models. However, a critical problem is that the F1-score has the potential to create a falsely optimistic or falsely pessimistic interpretation of the performance of the model, especially in cases when the data set is unbalanced [67]. To avoid the inflation problem, we include the Matthews correlation coefficient (MCC) [68] for the address prediction task. For a data set with k labels, we represent the MCC scores as a $k \times k$ confusion matrix. The values are calculated by the following equations:

$$MCC = \frac{C * N - \vec{T} \cdot \vec{P}}{\sqrt{N^2 - \vec{P} \cdot \vec{P}} \sqrt{N^2 - \vec{T} \cdot \vec{T}}} \tag{3}$$

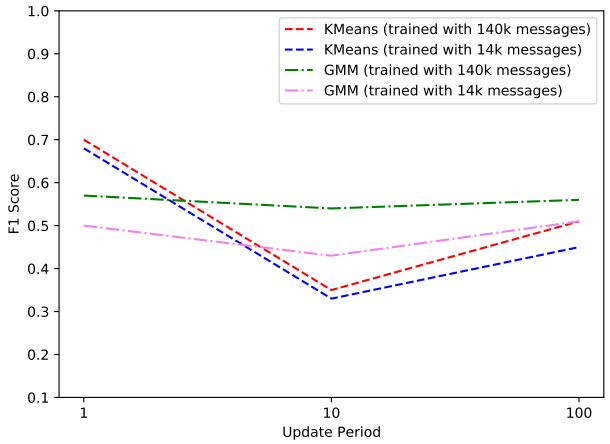
- $T_k = \sum_i CM_{ik}$, the number of times class k correctly predicted,
- $P_k = \sum_i CM_{ki}$, the number of times class k was predicted,
- $N = \sum_i \sum_j CM_{ij}$, the total number of data points.

Since the MCC value takes into consideration the weight of each class within the data set (as shown in Eq. 3), it provides a more reliable value of the model performance in multi-class problems with statistical significance, even with imbalanced data sets.

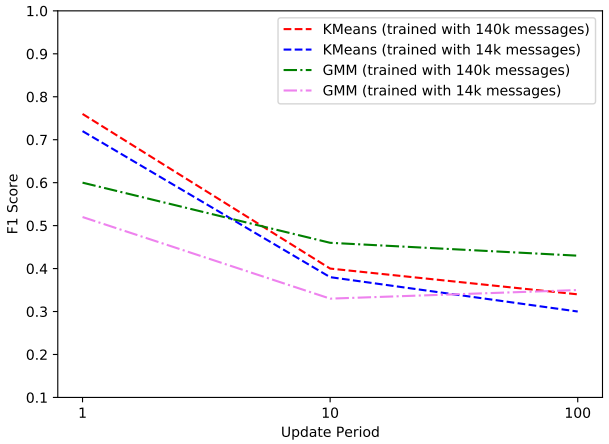
5.4 Results and Discussion

5.4.1 MTD Classification Task

For this task, four different models are trained and tested for their performance on a variety of data sets. When creating training and testing data for this task, the update period was varied (reflected on the x-axis of each of the experimental results figures). Therefore, for each experiment (reflected by



(a) F1 Score of Unsupervised Classification Models on Static Offset Dataset using either 8% or 80% of the training samples.



(b) F1 Score of Unsupervised Classification Models on Current Offset Dataset using either 8% or 80% of the training samples.

Fig. 4: Performance Metrics of Two Unsupervised Models for MTD Classification Task with varying MTD update frequency

an update period value on the x-axis), we perform training and testing on this particular dataset (corresponding to a specific update period), for each of the three models. As the number on the x-axis increases, the corresponding number of frames between each MTD update increases and the frequency of MTD updates decreases. For each experiment, around $\approx$ 175k messages were recorded from the bus monitor, 20% of which ($\approx$ 35k) are randomly sampled to be used as the test dataset. The remaining data ($\approx$ 140k messages) are used for training. From the training set, between 10% to 100% of the training dataset is randomly sampled in 5% increments to train the model. This incremental training process is used to determine the effectiveness of the model after accumulating certain number of messages. This paper presents the results for 8% ($\approx$ 14k) and 80% ($\approx$ 140k) of the total number of messages (the maximum and minimum amounts) in the following experiments to maximize interpretability of the figures. (Notee: the performance of the ML methods was bounded by these two extremes).

1) KNN: Fig. 3, shows F1 scores using KNN, in which we set $k = 3$. We observe that KNN achieves nearly perfect accuracy and F1 scores for both the CO and SO data sets.
2) RF: RF exhibits robust performance as shown in Fig. 3. F1 scores of 0.999 with very minimal variation for both the CO and SO data sets.
3) K-means clustering: Fig. 3 show that K-means clustering had relatively lower scores for the SO and CO data sets. The results demonstrate that larger amounts of training data doesn't improve the performance, as the accuracy scores using smaller training data sets were at least as good as those for the larger training data set. The scores for the SO data set oscillated up and down as the update period increased until it stabilizes at around F1 scores (of approximately 0.5) across higher update periods.
4) Gaussian Mixture Models: Fig. 4 compares GMM to K-means. We can observe that GMM outperforms K-means across the higher update periods where the MTD updates are less frequent, while K-means seems to do better with the lower update periods. But still, results demonstrate that larger amounts of training data only slightly improves the performance of GMMs. This indicates that the data distribution is not captured well by GMMs either.

The high accuracy and F1 scores in the MTD classification test for the supervised learning models are not surprising. In the training and testing data sets, MTD commands were the only broadcast messages, so these models only had to learn the difference between broadcast and non-broadcast messages to successfully classify MTD messages. The significant structural differences in broadcast and non-broadcast messages made the MTD classification task fairly simple. Having a larger variety of broadcast messages that are reflective of typical communication on a deployed MTD bus system would likely present a more significant challenge to the classification algorithms.

The relatively poor performance from the unsupervised algorithms could be explained by the fact that K-means relies on the assumption that all clusters in the dataset should have the same radius. The algorithm can produce erratic results when that assumption is violated. Although GMMs consider non-spherical data distribution, the improvement over K-means is not significant. In this dataset, the clusters around different index values varied for different MTD commands, which can offset the classification capability of these algorithm. This demonstrates the sensitivity of unsupervised algorithms to parsing message components.

5.4.2 *Address Prediction Task:*

The LSTM model requires specific hyperparameters to optimize learning performance and increase prediction accuracy. To get the right hyperparameters, we used grid search [69], and we validated the hyperparameters through 5-fold cross validation. Table 1 shows the values that were found to be optimal after hyperparameter tuning.

The model demonstrated high F1 scores (> 0.9) across all training date set sizes for the SO dataset, indicating high performance at predicting the next address. However, the model trained on the CO data set scored a maximum F1 of about 0.3 for all training data set sizes. We can observe it

Parameter	Value
Batch size	64
Learning rate	0.01
Epochs	50
Dropout rate	0.1
Number of hidden nodes	25
Number of dense layers	0.01
Activation function	`softmax`
Loss function	categorical cross entropy

TABLE 1: Hyperparameters and their values discovered through grid search

performed better with 2 and 3 previous addresses. We considered performance metrics for both training and testing, where training metrics mean that identical data items are used for both testing and training whereas testing metrics mean that the trained model classifies independent data items that were not used in training.

Dataset	F1 Train	MCC Train	F1 Test	MCC Test
SO	0.99	0.99	0.98	0.98
CO	0.17	0.15	0.17	0.14

TABLE 2: Address Prediction Performance Metrics for CO and SO datasets

The F1 and MCC scores in Table 2 further confirm the superior performance of the models trained and tested on SO datasets. Hence, the MTD algorithm generating addresses with the CO addressing mechanism would be more resilient to the same attack applied to a system, in which the MTD algorithm generated addresses with the SO addressing mechanism. When the static offset addressing mechanism is selected, one can observe a 1-to-1 mapping between the index value and the next address assignment. The LSTM needs to only observe all of the possible indices (in this case there are 128 columns in the state matrix) to learn all of the possible index-to-address mappings for this single device.

In contrast with the current offset addressing mechanism, there would be a 1-to-1 mapping between the combination of the previous device address and the index value. However, there are more state matrix entries and not all matrix entries are necessarily observed, increasing the uncertainty in the model's predictions.

A) Varying the Size of the Training Set:
In addition to the difference in resilience between CO and SO MTD algorithms, the rate at which a model can begin effectively hacking the MTD algorithm is a significant distinguishing factor as the attacker will try to hack the system quickly before detection.The experiments show that an attacker using the proposed DL model to predict the next address when the MTD algorithm uses the SO addressing mechanism would need to see less than 20,000 messages before being able to predict the next address with an F1 score of 98%. The model is able to predict the next address more effectively than a random guessing model even with fewer than 10,000 messages. This observation is shown in Figs. 5 and 6 where the MCC and F1-scores lies between 96% and 99.8% for all training sizes $\geq 20,000$ messages. This result

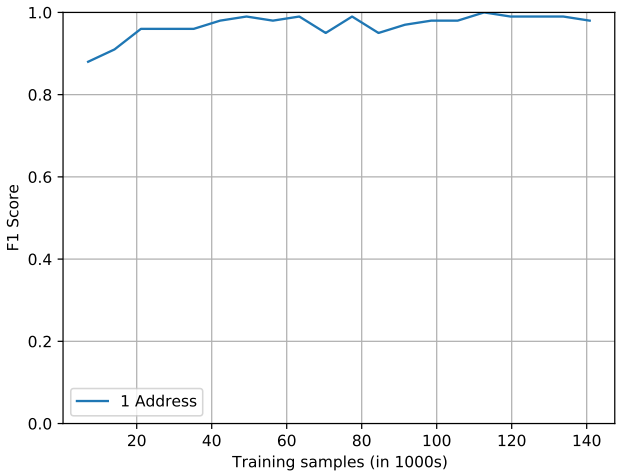


Fig. 5: F1 Score of address prediction model for Static Offset (SO) data set with varying training sizes

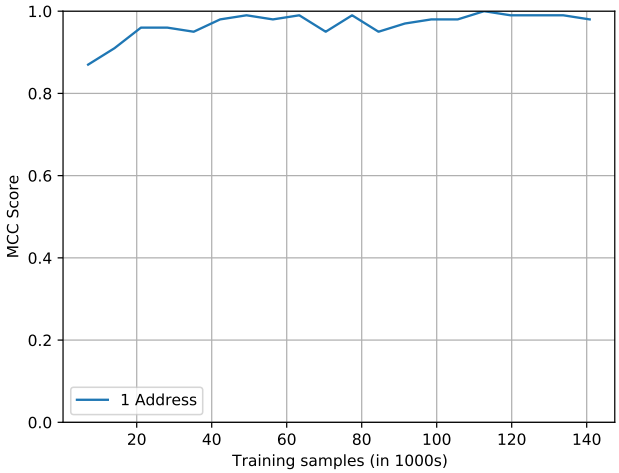


Fig. 6: MCC Score of address prediction model for Static Offset (SO) data set with varying training sizes

proves that an attacker can decipher the MTD algorithm using the SO addressing mechanism without a prolonged observation period. In addition, this result further implies that the MTD algorithm would need to generate a new matrix every 20,000 messages or less to thwart attackers. Conversely, the same model was unable to reach the same performance level at predicting the next address using the CO addressing mechanism. Although the model does not yet appear to have reached the limit of its performance with the largest number of messages, the model clearly requires a much larger set of messages in order to accurately predict the next address for the CO MTD algorithm. As shown in Figs. 7 and 8, even with $\geq 80,000$ samples of training data, both the MCC and F1 scores range between 0.12 and 0.16.

B) Varying the Number of Previously Seen Addresses:
As we increase the number of previous addresses that the model can see before making predictions, the performance of the model demonstrates marginal improvement on both the SO and CO datasets. The F1 and MCC scores for the

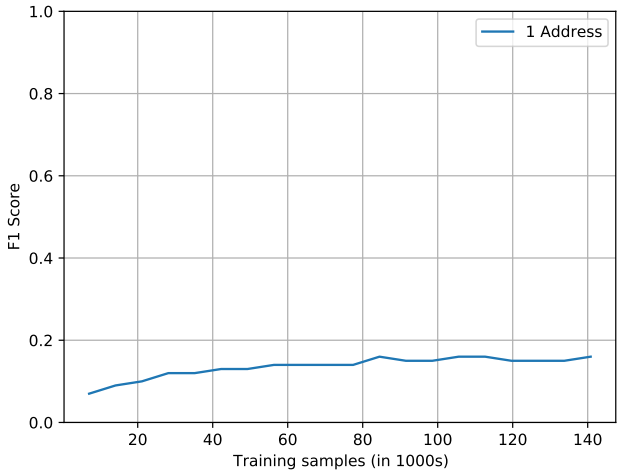


Fig. 7: F1 Score of address prediction model for Current Offset (CO) data set with varying training sizes

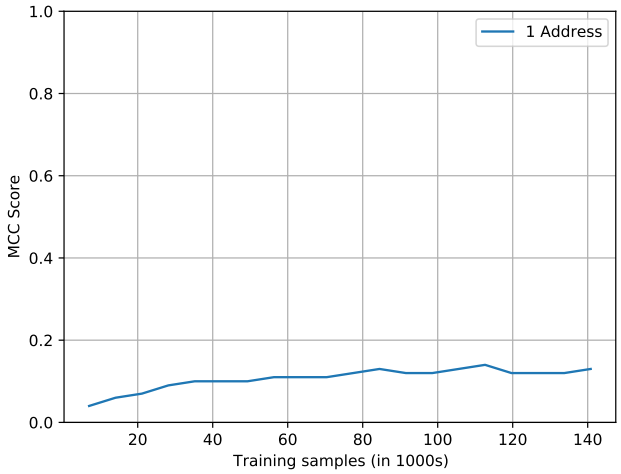


Fig. 8: MCC Score of address prediction model for Current Offset (CO) data set with varying training sizes

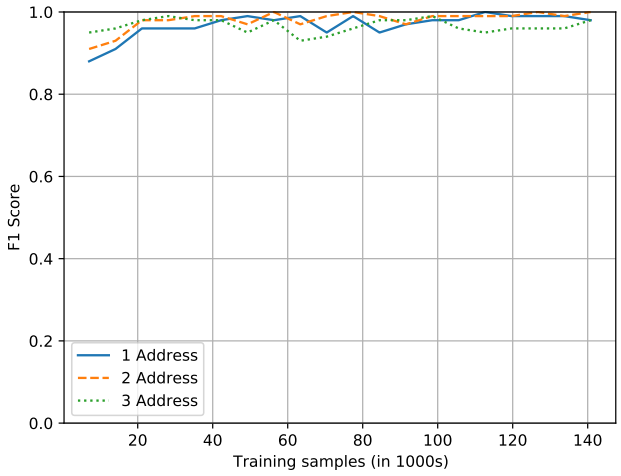


Fig. 9: F1 Score of address prediction model for Static Offset (SO) data set with varying training sizes and varying previous addresses

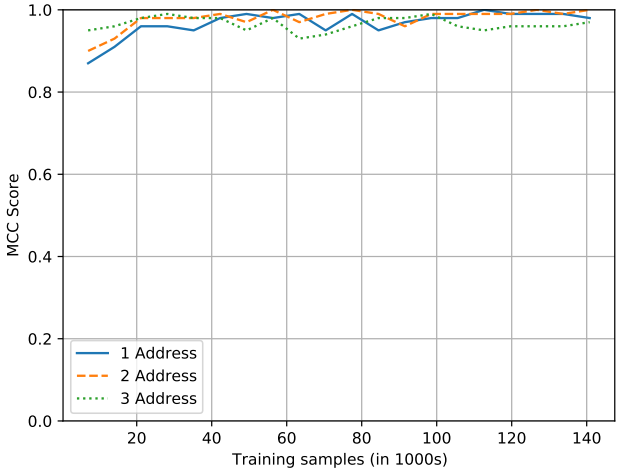


Fig. 10: MCC Score of address prediction model for Static Offset (SO) data set with varying training sizes and varying previous addresses

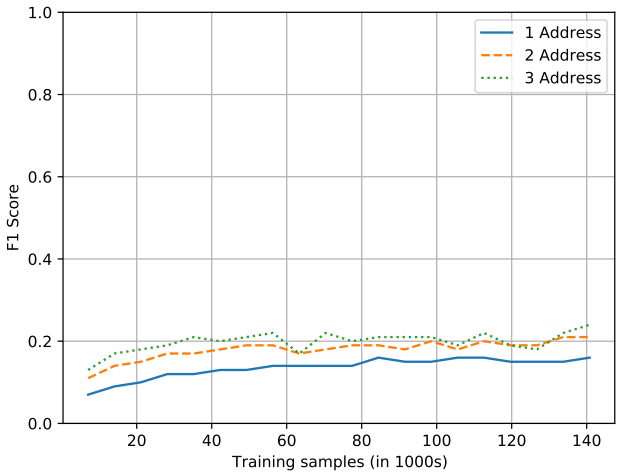


Fig. 11: F1 Score of address prediction model for Current Offset (CO) data set with varying training sizes and varying previous addresses

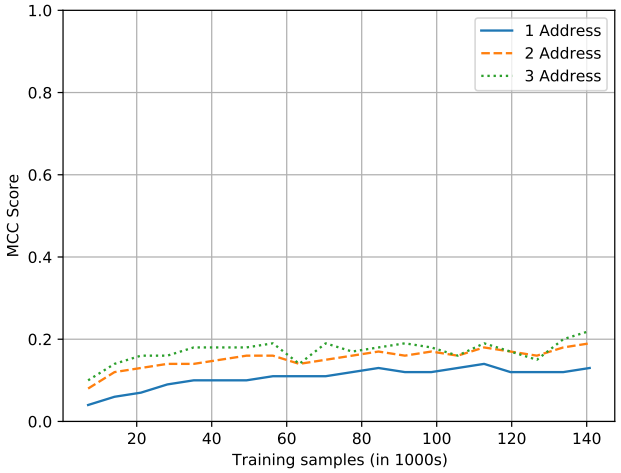


Fig. 12: MCC Score of address prediction model for Current Offset (CO) data set with varying training sizes and varying previous addresses

models trained on the SO data sets in Figures 9 and 10 range between 0.9 and 1 for both 2 and 3 previously seen addresses. Although the improvement on the CO dataset is stable as the size of the training dataset increases, it still does not yield satisfying MCC and F1 scores. Figures 11 and 12 show that the F1 and MCC scores are between approximately 0.1 and 0.2, suggesting that the model is not able to learn the MTD protocol when trained on the CO data sets, even when it observes up to 3 previous addresses.

6 Discussion on Mitigation Techniques

Thwarting ML based attacks requires a significant increase in variance of the data being generated i.e. increase in uncertainty through unique resource options and randomness lasting for longer period of time. Some of the following techniques can be effective in providing resiliency against ML based attacks.

6.1 Increasing Address Space

Increasing the physical address space is costly [70], particularly with real-systems such as space systems, where the computing resources are limited and they are SWaP-constrined. Virtualization provides an opportunity for light-weight resource availability. There have been a number of virtualization techniques proposed in the literature [71] [72]. Each limited number of physical addresses can have multiple virtual sub-addresses that can be randomized. Here, virtualization can be implemented at the physical system-level or at the network-level.

6.2 Obfuscation and Perturbation

Obfuscation and Perturbation Techniques can be used to hide or abstract the information that is available on the bus such that ML models cannot predict or classify accurately [73]. Data perturbation can be a powerful tool when it comes to thwarting ML based attacks. Adversarial machine learning techniques such as Trojan attacks [74] and adversarial sample attacks [75] can be used as reverse engineered to perturb the data i.e. place Trojans in the data to corrupt the attacker's ML models' training as well as testing data. This poisons the training as well as inferences of the model.

7 Conclusion and Future Work

This paper describes a series of experiments to evaluate how well ML and DL-based models can decipher an MTD algorithm in a real-time system. Our work evaluated the effectiveness of the models' predictions when an RT's address would change and to what address would it change. The models were trained and tested using messages from an MTD algorithm that was implemented in the MIL-STD-1553 protocol.

We observed that the supervised learning models were able to easily identify MTD update messages for all of the varying sizes of training and testing dataset. The performance of the unsupervised learning model was highly variable and the F1 scores were uniformly poor. Performance for the address prediction was greatly affected by the manner in which the MTD algorithm chose the next address. The LSTM model was able to predict the next address reasonably well when the SO addressing mechanism was used by the MTD. When the SO addressing mechanism is used, one can observe a 1-to-1 mapping between the index value and the next address assignment. The LSTM needs to only observe all of the possible indices (in this case, 128 columns in the state matrix) to learn all of the possible index-to-address mappings for this single device. Thus, this results in high F1 scores when the SO addressing mechanism is used.

Unlike the SO data, no clear 1-to-1 mapping between the index value and next address is easily observed in CO data. There is increased entropy and unpredictability in the CO address selection mechanism. Therefore, the LSTM model has a much greater difficulty in predicting the next address in the address sequence. The relatively constant performance over training samples and increase in number of previous addresses suggest that more data will not greatly improve the model's performance. Overall, we show that ML/DL models can be effective in thwarting MTD-based protection mechanisms in real-time system with SWaP-constraints.

Besides the MTD periodical implementation of address randomization discussed in this paper, investigating varying system configurations with different state-of-the-art MTD techniques is left for future work. In addition, to consider more generalized attacks, we plan to experiment with some unsupervised DL approaches that can learn without labels.

Further, we emphasize some of the main insights obtained from this study. Firstly, the MIL-STD-1553 MTD implementation can be resilient against ML-based attacks. These results are somewhat surprising, given the relatively small address space (31 available addresses) that is available within the MIL-STD-1553 protocol. MTD algorithms have been more widely studied in IP-based networks with larger address spaces, and [10] demonstrated that MTD could be implemented in real-time operating systems with small address space and even reduce adversarial knowledge gained via data exfiltration. However, this study expands upon [10] and other previous work by further demonstrating that the algorithm is resilient against a different kind of attack- an ML-based attack.

A second insight from this study is providing an example of how cyber experimentation can be used to quantitatively evaluate the effectiveness of MTD. [10] evaluated features of MTD algorithm and assigned a higher unpredictability score to the CO method than the SO method (0.85 vs 0.84), but prior to this study, it was not clear whether that small difference in predictability scores was meaningful. By performing the set of experiments described herein on an actual hardware setup, we definitively demonstrated that the small difference had a significant impact against ML-based attacks.

Lastly, ML and DL-based attacks are a growing subject of research (and, perhaps, practice). Most proposed cybersecurity "solutions" are not evaluated against these types of attacks. This study showed the value of evaluating the MTD algorithm against these attacks, and the types of insights that can be gained by performing these kinds of evaluations. For this specific MTD, we observed that it was necessary to have some flexibility in the indexing of the state matrix. More generally, we believe that cybersecurity solutions will

need some flexibility to be resilient against ML-based and the many other types of attacks that they may encounter.

ACKNOWLEDGMENT

This work has been supported by the LDRD Program at the Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

REFERENCES

[1] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, *Moving target defense: creating asymmetric uncertainty for cyber threats*. Springer Science & Business Media, 2011, vol. 54.

[2] L. Chen and A. Avizienis, "N-version programminc: A fault-tolerance approach to rellablllty of software operatlon," in *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'.*, 1995, pp. 113–119.

[3] T. Jackson, B. Salamat, A. Homescu, K. Manivannan, G. Wagner, A. Gal, S. Brunthaler, C. Wimmer, and M. Franz, "Compiler-generated software diversity," in *Moving Target Defense*. Springer, 2011, pp. 77–98.

[4] A. Saidane, V. Nicomette, and Y. Deswarte, "The design of a generic intrusion-tolerant architecture for web servers," *IEEE Transactions on dependable and secure computing*, vol. 6, no. 1, pp. 45–58, 2008.

[5] J. Liu, S. Wang, A. Zhou, S. A. Kumar, F. Yang, and R. Buyya, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 1191–1202, 2016.

[6] M. C. Rinard, "Using early phase termination to eliminate load imbalances at barrier synchronization points," in *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, 2007, pp. 369–386.

[7] M. Thompson, N. Evans, and V. Kisekka, "Multiple os rotational environment an implemented moving target defense," in *2014 7th International Symposium on Resilient Control Systems (ISRCS)*. IEEE, 2014, pp. 1–6.

[8] M. Carvalho and R. Ford, "Moving-target defenses for computer networks," *IEEE Security & Privacy*, vol. 12, no. 2, pp. 73–76, 2014.

[9] A. Chavez, W. Stout, and S. Peisert, "Techniques for the dynamic randomization of network attributes," in *Proceedings of the 49th Annual IEEE International Carnahan Conference on Security Technology*, 2015.

[10] C. Jenkins, E. Vugrin, I. Manickam, N. Troutman, J. Hazelbaker, S. Krakowiak, J. Maxwell, and R. Brown, "Moving target defense for space systems," in *Proceedings of the 2021 IEEE Space Computing Conference*, Aug 2021.

[11] J. Beeharry and B. Nowbutsing, "Forecasting ipv4 exhaustion and ipv6 migration," in *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)*, 2016, pp. 336–340.

[12] G. de Koning Gans, J.-H. Hoepman, and F. D. Garcia, "A practical attack on the mifare classic," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2008, pp. 267–282.

[13] J. Zheng and A. S. Namin, "A survey on the moving target defense strategies: An architectural perspective," *Journal of Computer Science and Technology*, vol. 34, no. 1, pp. 207–233, 2019.

[14] N. O. Ahmed and B. Bhargava, "Bio-inspired formal model for space/time virtual machine randomization and diversification," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2020.

[15] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati, "A survey of moving target defenses for network security," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1909–1941, 2020.

[16] N. Ahmed and B. Bhargava, "From byzantine fault-tolerance to fault-avoidance: An architectural transformation to attack and failure resiliency," *IEEE Transactions on Cloud Computing*, vol. PP, pp. 1–1, 03 2018.

[17] S. Yoon, J.-H. Cho, D. S. Kim, T. J. Moore, F. Free-Nelson, and H. Lim, "Attack graph-based moving target defense in software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1653–1668, 2020.

[18] H. Alavizadeh, H. Alavizadeh, D. S. Kim, J. Jang-Jaccard, and M. N. Torshiz, "An automated security analysis framework and implementation for mtd techniques on cloud," in *Information Security and Cryptology – ICISC 2019*, J. H. Seo, Ed. Cham: Springer International Publishing, 2020, pp. 150–164.

[19] J. Narantuya, S. Yoon, H. Lim, J.-H. Cho, D. S. Kim, T. Moore, and F. Nelson, "Sdn-based ip shuffling moving target defense with multiple sdn controllers," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks–Supplemental Volume (DSN-S)*. IEEE, 2019, pp. 15–16.

[20] D. P. Sharma, D. S. Kim, S. Yoon, H. Lim, J.-H. Cho, and T. J. Moore, "Frvm: Flexible random virtual ip multiplexing in software-defined networks," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 579–587.

[21] Y. Jie, M. Li, C. Guo, and L. Chen, "Dynamic defense strategy against dos attacks over vehicular ad hoc networks based on port hopping," *IEEE Access*, vol. 6, pp. 51 374–51 383, 2018.

[22] L. Zhang, Y. Guo, H. Yuwen, and Y. Wang, "A port hopping based dos mitigation scheme in sdn network," in *2016 12th International Conference on Computational Intelligence and Security (CIS)*. IEEE, 2016, pp. 314–317.

[23] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati, "A survey of moving target defenses for network security," *IEEE Communications Surveys Tutorials*, vol. 22, no. 3, pp. 1909–1941, 2020.

[24] U. S. D. of Defense, "Digital time division command/response multiplex data bus," Tech. Rep., 2018. [Online]. Available: https://quicksearch.dla.mil/qsDocDetails.aspx?ident_number=36973

[25] M. Gopal, *Applied machine learning*. McGraw-Hill Education, 2019.

[26] M. Nawir, A. Amir, N. Yaakob, and O. B. Lynn, "Effective and efficient network anomaly detection system using machine learning algorithm," *Bulletin of Electrical Engineering and Informatics*, vol. 8, no. 1, pp. 46–51, 2019.

[27] N. S. Murugan and G. U. Devi, "Detecting spams in social networks using ml algorithms-a review," *International Journal of Environment and Waste Management*, vol. 21, no. 1, pp. 22–36, 2018.

[28] H. M. Farooq and N. M. Otaibi, "Optimal machine learning algorithms for cyber threat detection," in *2018 UKSim-AMSS 20th International Conference on Computer Modelling and Simulation (UKSim)*. IEEE, 2018, pp. 32–37.

[29] E. National Academies of Sciences, Medicine *et al.*, *Implications of artificial intelligence for cybersecurity: Proceedings of a workshop*. National Academies Press, 2020.

[30] M. S. Elsayed, N.-A. Le-Khac, S. Dev, and A. D. Jurcut, "Ddosnet: A deep-learning model for detecting network attacks," in *2020 IEEE 21st International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2020, pp. 391–396.

[31] A. Aljuhani, "Machine learning approaches for combating distributed denial of service attacks in modern networking environments," *IEEE Access*, vol. 9, pp. 42 236–42 264, 2021.

[32] M. Tayyab, B. Belaton, and M. Anbar, "Icmpv6-based dos and ddos attacks detection using machine learning techniques, open challenges, and blockchain applicability: a review," *IEEE Access*, vol. 8, pp. 170 529–170 547, 2020.

[33] A. Basit, M. Zafar, X. Liu, A. R. Javed, Z. Jalil, and K. Kifayat, "A comprehensive survey of ai-enabled phishing attacks detection techniques," *Telecommunication Systems*, vol. 76, no. 1, pp. 139–154, 2021.

[34] I. Firdausi, C. lim, A. Erwin, and A. S. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," in *2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, 2010, pp. 201–203.

[35] T. A. Tuan, H. V. Long, L. H. Son, R. Kumar, I. Priyadarshini, and N. T. K. Son, "Performance evaluation of botnet ddos attack detection using machine learning," vol. 13, 2020, pp. 283–294. [Online]. Available: https://doi.org/10.1007/s12065-019-00310-w

[36] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. Van Gool, "AI benchmark: All about deep learning on smartphones in 2019," 2019.

[37] Y. Deng, "Deep learning on mobile devices - A review," *CoRR*, vol. abs/1904.09274, 2019. [Online]. Available: http://arxiv.org/abs/1904.09274

[38] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2016, pp. 1–12.

[39] O. Stan, A. Cohen, Y. Elovici, and A. Shabtai, "Intrusion detection system for the mil-std-1553 communication bus," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 4, pp. 3010–3027, 2020.

[40] S. J. J. Généreux, A. K. H. Lai, C. O. Fowles, V. Roberge, G. P. M. Vigeant, and J. R. Paquet, "MAIDENS: MIL-STD-1553 anomaly-based intrusion detection system using time-based histogram comparison," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 56, no. 1, pp. 276–284, 2020. [Online]. Available: https://doi.org/10.1109/TAES.2019.2914519

[41] B. Losier, R. Smith, and V. Roberge, "Design of a time-based intrusion detection algorithm for the mil-std-1553," Royal Military College of Canada, Tech. Rep., 2019.

[42] S. Woo, D. Moon, T. Youn, Y. Lee, and Y. Kim, "Can id shuffling technique (cist): Moving target defense strategy for protecting in-vehicle can," *IEEE Access*, vol. 1, pp. 15 521–15 536, 2019.

[43] R. Brown, A. Marti, C. Jenkins, and S. Shannigrahi, "Dynamic address validation array (dava): A moving target defense protocol for can bus," in *Proceedings of the 7th ACM Workshop on Moving Target Defense*, ser. MTD'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 11–19. [Online]. Available: https://doi.org/10.1145/3411496.3421221

[44] B. Potteiger, Z. Zhang, and X. Koutsoukos, "Integrated moving target defense and control reconfiguration for securing cyber-physical systems," *Microprocessors and microsystems*, vol. 73, p. 102954, 2020.

[45] S. Achleitner, T. F. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Deceiving network reconnaissance using sdn-based virtual topologies," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1098–1112, 2017.

[46] A. Zheng and A. Casari, *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

[47] J. Heaton, "An empirical analysis of feature engineering for predictive modeling," in *SoutheastCon 2016*. IEEE, 2016, pp. 1–6.

[48] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[49] I. Triguero, D. García-Gil, J. Maillo, J. Luengo, S. García, and F. Herrera, "Transforming big data into smart data: An insight on the use of the k-nearest neighbors algorithm to obtain quality data," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 2, p. e1289, 2019.

[50] W. Diao, X. Liu, Z. Li, and K. Zhang, "No pardon for the interruption: New inference attacks on android through interrupt timing analysis," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 414–432.

[51] R. Spreitzer, F. Kirchengast, D. Gruss, and S. Mangard, "Procharvester: Fully automated analysis of procfs side-channel leaks on android," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 749–763.

[52] X. Zhang, X. Wang, X. Bai, Y. Zhang, and X. Wang, "Os-level side channels without procfs: Exploring cross-app information leakage on ios," in *Proceedings of the Symposium on Network and Distributed System Security*, 2018.

[53] W. Lin, Z. Wu, L. Lin, A. Wen, and J. Li, "An ensemble random forest algorithm for insurance big data analysis," *IEEE Access*, vol. 5, pp. 16 568–16 575, 2017.

[54] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1273–1285.

[55] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models," *arXiv preprint arXiv:1806.01246*, 2018.

[56] J. Sun, X. Jin, Y. Chen, J. Zhang, Y. Zhang, and R. Zhang, "Visible: Video-assisted keystroke inference from tablet backside motion." in *NDSS*, 2016.

[57] L. Bai, X. Cheng, J. Liang, H. Shen, and Y. Guo, "Fast density clustering strategies based on the k-means algorithm," *Pattern Recognition*, vol. 71, pp. 375–386, 2017.

[58] M. Azizyan, A. Singh, and L. Wasserman, "Efficient Sparse Clustering of High-Dimensional Non-spherical Gaussian Mixtures," in *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Lebanon and S. V. N. Vishwanathan, Eds., vol. 38. San Diego, California, USA: PMLR, 09–12 May 2015, pp. 37–45. [Online]. Available: https://proceedings.mlr.press/v38/azizyan15.html

[59] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference attacks on fully connected neural networks using permutation invariant representations," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 619–633.

[60] N. Li, W. Qardaji, D. Su, Y. Wu, and W. Yang, "Membership privacy: a unifying framework for privacy definitions," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 889–900.

[61] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[62] H. H. Tan and K. H. Lim, "Vanishing gradient mitigation with deep learning neural network optimization," in *2019 7th international conference on smart computing & communications (ICSCC)*. IEEE, 2019, pp. 1–4.

[63] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, pp. 64–67, 2001.

[64] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[65] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Comput.*, vol. 12, no. 10, p. 2451–2471, Oct. 2000. [Online]. Available: https://doi.org/10.1162/089976600300015015

[66] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artificial Intelligence Review*, vol. 53, 12 2020.

[67] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, p. 6, 2020.

[68] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using matthews correlation coefficient metric," *PloS one*, vol. 12, no. 6, p. e0177678, 2017.

[69] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: A big comparison for nas," *ArXiv*, vol. abs/1912.06059, 2019.

[70] K. Lu, C. Song, B. Lee, S. P. Chung, T. Kim, and W. Lee, "Aslr-guard: Stopping address space leakage for code reuse attacks," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 280–291.

[71] M. Albanese, A. De Benedictis, S. Jajodia, and K. Sun, "A moving target defense mechanism for manets based on identity virtualization," in *2013 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2013, pp. 278–286.

[72] G. Lin, M. Dong, K. Ota, J. Li, W. Yang, and J. Wu, "Security function virtualization based moving target defense of sdn-enabled smart grid," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[73] Z. Zhang, R. Deng, D. K. Yau, P. Cheng, and J. Chen, "Analysis of moving target defense against false data injection attacks on power grid," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2320–2335, 2019.

[74] M. Zou, Y. Shi, C. Wang, F. Li, W. Song, and Y. Wang, "Potrojan: powerful neural-level trojan designs in deep learning models," *arXiv preprint arXiv:1802.03043*, 2018.

[75] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, 2018.



Ganapathy Mani received his Ph.D. in Computer Science from Purdue University, IN, USA and M.Sc. degree from George Washington University, Washington DC, USA. He worked as a senior research engineer at Qualcomm, Inc., applying deep learning techniques for vulnerability detection in source code. He is actively working on applying machine learning, NLP, and reinforcement learning for cybersecurity, behavior profiling, predicting market manipulation in finance, and adversarial machine learning.



Marina Haliem received the B.Sc. and M.Sc. degrees in Computer Science from the British University in Egypt, Cairo, Egypt, (validated from Loughborough University, UK) in 2015, and 2018 respectively. She received her Ph.D. degree from Purdue University, IN,USA in 2022. Currently, she is a software engineer with the behavior prediction team at Waymo, Inc.. Her research interests are in the areas of Machine Learning, deep learning, AI, reinforcement learning, and adversarial machine learning.



Bharat Bhargava received the BS degree from Punjab University, in Mathematics (1966), the BE, degree from the Indian Institute of Science, in Electrical and Computer Engineering (1969), and his PhD degree from Purdue University, Electrical Engineering (1974). He is currently a professor of the Department of Computer Science at Purdue University. His recent work is on Intelligent Autonomous Systems, security, data analytics and machine learning.



Indu Manickam received B.S.E.degrees in biomedical engineering and electrical and computer engineering from Duke University in Durham, NC, USA, and an M.Sc. degree in electrical and computer engineering from Rice University in Houston, TX, USA. She is currently a senior member of the technical staff at Sandia National Laboratories.



Myeongsu Kim is a Ph.D student at Purdue University department of Computer Science. His research interests include distributed systems, system security, and machine learning.



Eric Vugrin received BS and MS degrees in Mathematics from Texas Tech University in 1998 and 2000, respectively, and a PhD in Mathematics from Virginia Tech in 2004. He is a Senior Scientist with Sandia National Laboratories in Albuquerque, New Mexico. His research interests include cyber resilience and experimentation.



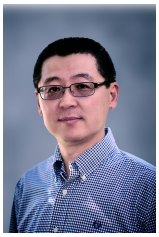
Weichao Wang 's research interests are in designing protocols and mechanisms to secure pervasive systems, wireless networks, cloud computing environments, and critical infrastructures. He is also very interested in security education for both students and working professionals. His research is supported by NSF, the Department of Labor, NIST, Duke Energy, NC Department of Public Instruction (NCDPI), and UNC Charlotte.



Chris Jenkins is a Principal Cybersecurity R&D staff member in the Systems Security Research Department. Chris supports our Cyber-Physical Mission R&D group as part of Sandia's Information Operations Center. Chris leads a team at Sandia, partners with academic, government and industry collaborators, and conducts research on topics related to moving target defense algorithms for operational technology cybersecurity.



Pelin Angin is an Assistant Professor of Computer Engineering at Middle East Technical University. She completed her B.S. in Computer Engineering at Bilkent University in 2007 and her Ph.D. in Computer Science at Purdue University, USA in 2013. Her research interests lie in the fields of cloud computing and IoT security, distributed systems, 5G networks and blockchain.



Meng Yu is a Robert Miner Endowed Chair Professor of Roosevelt University. He received his Ph.D. degree in Computer Science from Nanjing University, China. His research interests include systems and network security, cloud computing, virtualization and security. His research has been funded by NSF. He is a member of IEEE and ACM.