SHARKS: Smart Hacks, Attacks, RisKs and Security in Internet-of-Things and Cyber-Physical Systems based on Machine learning

Journal:	*Transactions on Emerging Topics in Computing*
Manuscript ID	Draft
Manuscript Type:	Technical Track (Regular Paper)
Keywords:	K.6.5.e Unauthorized access (hacking, phreaking) < K.6.5 Security and Protection < K.6 Management of Computing and Information S, K.6.m.b Security < K.6.m Miscellaneous < K.6 Management of Computing and Information Systems < K Computing Milieux

SHARKS: Smart Hacks, Attacks, RisKs, and Security in Internet-of-Things and Cyber-Physical Systems based on Machine Learning

Tanujay Saha, Najwa Aaraj, Neel Ajjarapu and Niraj K. Jha *(Fellow, IEEE)*

Abstract—Cyber-physical systems (CPS) and Internet-of-Things (IoT) devices are increasingly being deployed across multiple functionalities, ranging from healthcare devices and wearables to critical infrastructures, e.g., nuclear power plants, autonomous vehicles, smart cities, and smart homes. These devices are inherently insecure across their comprehensive software, hardware, and network stacks, thus presenting a large vulnerability surface that can be exploited by hackers. In this article, we present an innovative technique for detecting *unknown* system vulnerabilities, manage associated vulnerabilities, and improve incident response when such vulnerabilities are exploited. The novelty of this approach lies in extracting intelligence from known real-world CPS/IoT attacks, representing them in the form of regular expressions, and employing machine learning (ML) techniques on this ensemble of regular expressions to generate new attack vectors and security vulnerabilities. Our results show that 10 new attack vectors and 122 new vulnerability exploits can be successfully generated that have the potential to exploit a CPS or an IoT ecosystem. The ML methodology achieves an accuracy of 97.7% and enables us to predict these attacks efficiently with a 87.5% reduction in the search space. We demonstrate the application of our method to hacking the in-vehicle network of a connected car. To defend against the known attacks and possible novel exploits, we discuss a defense-in-depth mechanism for various classes of attacks and the classification of data targeted by such attacks. This defense mechanism optimizes the cost of security measures based on the sensitivity of the protected resource, thus incentivizing its adoption in real-world CPS/IoT by cybersecurity practitioners.

Index Terms—Artificial Intelligence; Attack Graphs; Cyber-Physical Systems; Cybersecurity; Embedded Systems; Internet-of-Things; Machine Learning.

1 Introduction

Cyber-physical systems (CPS) use sensors to feed data to computing elements that monitor and control physical systems and use actuators to elicit desired changes in the environment. Internet-of-Things (IoT) enables diverse, uniquely identifiable, and resource-constrained devices (sensors, processing elements, actuators) to exchange data through the Internet and optimize desired processes. CPS/IoT have a plethora of applications, like smart cities [1], [2], smart healthcare [3], smart homes [4], nuclear plants [5], smart grids [6], [7], autonomous vehicles [8], and in various other domains. With recent advances in CPS/IoT-facilitating technologies like machine learning (ML), cloud computing, and 5G communication systems [9], CPS/IoT are likely to have an even more widespread impact in the near future.

An unfortunate consequence of integrating multiple devices is the dramatic increase in the attack surface within the ecosystem. Most of the CPS/IoT devices are energy-constrained, which makes them unable to implement existing elaborate cryptographic protocols and primitives as well as other conventional security measures across the software, hardware, and network stacks [10], [11]. The diverse range of embedded devices in the network and inherent vulnerabilities in the design and implementation, coupled with an absence of standard cryptographic primitives, and network security protocols, make CPS/IoT a favorable playground for malicious attackers. Although lightweight cryptographic protocols [12], [13] and hardware-based (lightweight) authentication protocols [14], [15] mitigate some threats, most of the vulnerabilities remain unaddressed. Another challenge in securing CPS/IoT is the large amount of accessible data generated by the numerous communication channels amongst devices. Such data, in the absence of adequate cryptographic technologies, pose a threat to the CPS/IoT device and consequently impact user privacy, data confidentiality, and integrity. Moreover, CPS/IoT are vulnerable to a plethora of attacks [10], [16], e.g., buffer overflow exploits, race conditions, XSS attacks that target known vulnerabilities and new (undiscovered) vulnerabilities, the exploit of which is referred to as a zero-day attack.

In this article, we propose an ML-based approach to systematically generate new exploits in a CPS/IoT framework. ML has already found use in CPS/IoT cybersecurity [17]–[19], primarily in network intrusion and anomaly detection systems [20]. These systems execute ML algorithms on data generated by network logs and communication channels. In the methodology that we propose, ML instead operates at the device level, both system and user levels, to predict unknown exploits against CPS/IoT.

We analyze an exhaustive set of real-world CPS/IoT

• *T. Saha, N. Ajjarapu and N. K. Jha are with the Department of Electrical Engineering, Princeton University, New Jersey, NJ, 08544 ({tsaha,najjarapu,jha}@princeton.edu. N. Aaraj is with Technology Innovation Institute, UAE (najwa@tii.ae).*

attacks that have been documented and represent them as regular expressions. An ML algorithm is then trained with these regular expressions. The trained ML model can predict the feasibility of a new attack. The vulnerability exploits predicted to be highly feasible by the ML algorithm are reported as novel exploits. This approach successfully generated 122 novel exploits and 10 unexploited attack vectors. To demonstrate the applicability of our approach, we evaluate the trained model on the in-vehicle network of a connected car. The model was successful in discovering 45 vulnerability exploits in the car network.

The novelty of the proposed methodology lies in:

- Representation of real-world CPS/IoT attacks in the form of regular expressions and control-data flow graphs (CDFGs), where both control flow and data invariants are instrumented at low system levels.
- Creation of an aggregated attack directed acyclic graph (DAG) with an ensemble of such regular expressions.
- Use of an ML model trained with these regular expressions to generate novel exploits in a given CPS/IoT framework.

The article is organized as follows. Section 2 provides a summary of the work that has been done in the application of ML and automation to cybersecurity. Section 3 discusses background material. Section 4 provides motivation behind why our contribution may be beneficial to progress in CPS/IoT security research. Section 5 gives details of our methodology and the results obtained with it. Section 6 describes the application of our algorithm to a connected vehicle. Section 7 proposes a tiered-security framework, composed of defense DAGs, for protection against security vulnerabilities. Section 8 concludes the article.

2 Related Work

In this section, we discuss some of the major works that have been done to automate security for real-world threat mitigation. Many major classes of security vulnerabilities, like memory corruption bugs and network intrusion vulnerabilities, can be detected using automation techniques. The domain of cybersecurity that has been highly influenced by the popularity of ML is intrusion detection systems (IDSs), in particular an IDS targeted at network-level attacks. Prior to the rapid advancements in ML, IDSs consisted of signature-based methods and anomaly-based techniques to detect intrusions in the network or the host systems. Proposed IDSs perform quite well but have their drawbacks. Signature-based methods require regular updates of the software and are unable to detect zero-day vulnerabilities. Anomaly-based methods can detect zero-day vulnerabilities but have a very high false alarm rate (FAR). The advent of ML alleviated some of these drawbacks and thus ML was widely adopted in IDSs. Researchers have used a wide variety of ML methodologies to tackle this problem, such as artificial neural networks [21], [22], Bayesian networks [23], [24], clustering methods [25]–[27], decision trees [28], [29], ensemble learning like random forests [30], [31], hidden Markov models [32], and support vector machine (SVM) [33], [34]. More advanced deep learning based IDSs use generative adversarial networks [35] and autoencoders [36]. These methods provide a reactive security mechanism for detecting ongoing attacks. They also require significant computational overhead because the models need to be continuously trained on recent data and all incoming traffic must be processed by the ML model before it can be catered to by the system. Our method differs from these methods in that it provides proactive security and requires zero run-time overhead.

Attack graphs have been widely used for analyzing the security of systems and networks [37]. Generating attack graphs has been a longstanding challenge due to the state explosion problem. Various automation techniques, like model checking [38], rule-based artificial intelligence, and ML [39], have been used to tackle this challenge. Analysis of the attack graphs is also a challenge due to the enormous size and complexity of the graphs. Graph-based neural networks [40] and reinforcement learning [41] have been used to analyze attack graphs to detect vulnerabilities. This article uses attack graphs at a higher granularity to detect vulnerabilities and the exploits thereof across the entire hardware, software, and network stacks of CPS/IoT. In previous works, system-specific attack graphs have been used for vulnerability analysis. In this article, we propose a generalized attack graph that can be applied to detect vulnerabilities (and exploits thereof) in any CPS/IoT. We buttress this claim by applying our approach to detect vulnerabilities in the in-vehicle network of a connected car.

Memory corruption bugs have been a longstanding vulnerability in computer systems. A detailed analysis of this problem is provided in [42]. Automation attempts have also been made for detecting such bugs. In [43], static analysis is used to detect memory corruption vulnerabilities.

The discovery of hardware vulnerabilities like Spectre [44] and Meltdown [45] in 2018 opened the gateway to new classes of side-channel attacks on device microarchitecture. An automated side-channel vulnerability detection technique for microarchitectures is proposed in [46]. This article aims to achieve a similar goal, but across the entire hardware, software, and network stacks.

3 Background

We model existing CPS/IoT attacks as regular expressions and CDFGs. We train a popular ML model, namely SVM, with these CDFGs to predict new vulnerability exploits. This section provides an introduction to regular expressions, CDFGs, and SVM models that is required for ease of comprehending the rest of the article.

3.1 Regular Expressions

A regular expression is used to denote a set of string patterns. We use regular expressions to represent known CPS/IoT attacks in a compact and coherent manner.

The set of all possible characters permissible in a regular expression is referred to as its alphabet Σ. The basic operations permitted in regular expressions are [47]:

- **Set union**: This represents the set union of two regular expressions. For example, if expression A denotes

$\{xy, z\}$ and B denotes $\{xy, r, pq\}$, then expression $A + B$ denotes $\{xy, z, r, pq\}$.

- **Concatenation**: This operation represents the set of strings obtained by attaching any string in the first expression with any string in the second expression. For example, if $A = \{xy, z\}$ and $B = \{r, pq\}$, then, $AB = \{xyr, xypq, zr, zpq\}$.
- **Kleene star**: A^* denotes the set of strings obtained by concatenating the strings in A any number of times. A^* also includes the null string λ. For example, if $A = \{xy, z\}$ then, $A^* = \{\lambda, xy, z, xyz, zxy, xyxy, zz, xyxyxy, xyzxy, ...\}$

In this article, we define regular expressions at a higher granularity for the sake of generality. Alphabet Σ of our regular expressions includes generic system-level operations like "*Access port 1234 of the system*," "*Overwrite pointer address during memory overflow*," etc.

3.2 Control-data Flow Graph

The CDFG of a program is a graphical representation of all possible control paths and data dependencies that the program might encounter during its execution. The basic blocks of the program constitute the nodes of the CDFG. A basic block is a block of sequential statements that satisfy the following properties:

- The control flow enters only at the beginning of the block.
- The control flow leaves only at the end of the block.
- A block contains a data invariant or a low-level system call.

For the sake of this article, we construct the CDFGs at the level of human-executable instructions rather than assembly-level instructions. We do this to ensure general applicability of our method and ease of understanding.

3.3 Support Vector Machine

Neural networks are capable of performing better than traditional ML algorithms in many scenarios. However, neural networks require a lot of training data. We employ ML at the system level. Our training dataset does not have enough training examples to train a robust neural network. Thus, we use traditional ML approaches, instead of deep learning, for classification. Among traditional ML classification algorithms, SVM is one of the most robust classifiers that generalizes quite well.

SVM is a class of supervised ML algorithms that analyzes a labeled training dataset to perform either classification or regression [48]. It is capable of predicting the label of a new example with high accuracy. It is inherently designed to be a linear binary classifier. However, kernel transformations can be used to perform nonlinear classification as well. For a dataset with an n-dimensional feature space, a trained SVM model learns an $(n-1)$-dimensional hyperplane that serves as the *decision boundary*, also referred to as the *separating hyperplane*.

Many contemporary ML algorithms, e.g., k-nearest-neighbor classification, use a greedy search approach. However, SVM uses a quadratic optimization algorithm to output an optimal decision boundary. The two main limitations of SVM are its natural binding to binary classification and the need to specify (rather than learn) a kernel function.

4 Motivation

This section illustrates how our approach fits into the larger CPS/IoT security picture. CPS/IoT are expected to be pervasively deployed and significantly impact various aspects of our daily lives, often performing functions that are highly critical to human safety (e.g., in medical, emergency response, and automotive systems) or the functioning of enterprises or society at large (e.g., smart cities, energy-efficient buildings, traffic monitoring, and smart power grid). While information security has clearly emerged as a grand challenge in CPS/IoT, the consequences of security attacks on CPS/IoT, the infrastructure of the future, can often be described as catastrophic – much too often, the cost of these attacks may have to be measured in lives as opposed to dollars and cents.

We propose a framework for securing IoT devices and CPS infrastructure based on developments along two important directions. Recognizing the need to depart from the traditional approaches to cybersecurity, we observe that the main objective of many security attacks on CPS/IoT is to modify the behavior of the end-system to cause unsafe operation. Based on this insight, we propose to model the behavior of CPS/IoT under attack, at the system and network levels, use ML to discover a more exhaustive potential attack space, and then map it to a defense space.

To demonstrate the practicality of our approach, we illustrate its applicability to connected cars, whose market size is expected to surpass USD 200 billion by 2025 [49], split between services and solutions related to vehicle data networks, vehicle-to-vehicle networks, sensor technologies, and vision systems. Our approach enables us to address the following questions:

- It enables a preemptive analysis of vulnerabilities across a large variety of devices by detecting new attacks and deploying patches ahead of time.
- It ensures security of communication between devices and bridges the CPS/IoT security gap.
- It enhances CPS/IoT data integrity, confidentiality, and availability while ensuring reliability of information collected from various sensors.

Coupled with other technologies, such as (i) lightweight cryptographic protocols [12], (ii) cryptographic primitives on devices for data-at-rest security (iii) security protocols for data-in-transit security, and (iv) data auditing using immutable databases [50], our methodology enhances end-to-end security.

5 Methodology

In our methodology, we extract intelligence from an ensemble of known CPS/IoT attacks and use this system-level adversarial intelligence to predict other possible exploits in a given CPS/IoT framework. The automated derivation of novel exploits and defenses broadly comprises extracting intelligence, discovering unexploited attack vectors, applying ML, and taking measures to secure the system. These processes are depicted in the flowchart of Fig. 1.

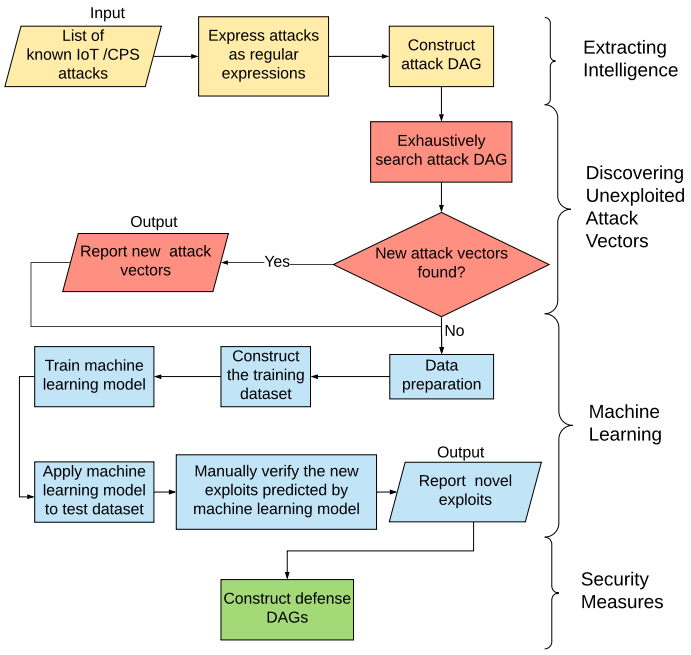

Fig. 1: Flowchart of the overall methodology

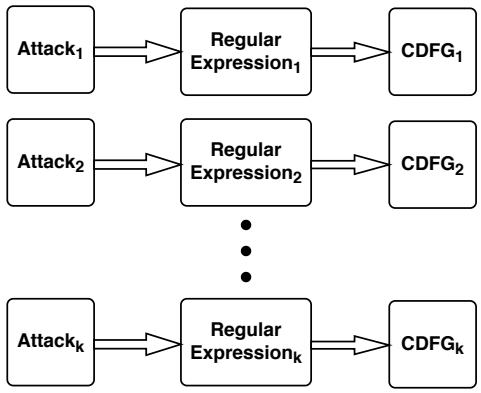

Fig. 2: Data transformation overview of a list containing k types of CPS/IoT attacks

5.1 Extracting Intelligence

We document existing CPS/IoT attacks and decompose them into their constituent system-level actions and used data invariants. We use regular expressions to represent these constituent system-level operations. Then we combine the regular expressions of all the attacks to form an ensemble of interconnected system-level operations. This ensemble is represented as a DAG. This DAG is henceforth referred to as the aggregated attack DAG.

5.1.1 Data Collection

Next, we discuss how to extract knowledge from known attack patterns. To achieve this objective, we create a list of known CPS/IoT attacks. Then we classify these attacks into various categories based on the type of vulnerability being exploited. This list consists of 41 different attacks [10], [51], [52]. The most popular attacks among these and their regular expressions are shown in Table 1.

5.1.2 Data Transformation

In this phase, we decompose each attack into its basic system-level operations. We express these sequences of operations as regular expressions that are then represented as CDFGs, as shown in Fig. 2. Each attack is now transformed into a CDFG with system-level operations as its basic blocks. The methodology of decomposing an attack into a CDFG is similar to the method used in [53].

An example of the data transformation procedure for a buffer overflow attack is given next. A buffer overflow attack can be expressed as a sequence of following actions:

1) dynamic memory allocation,
2) overflow of memory, and
3) frame pointer with overwritten memory.

Let bb_i denote the i^{th} basic block of the sequence. Then the corresponding regular expression is given by:

$$bb_i(\textit{Dynamic memory allocation})^*.bb_j(\textit{Overflow of memory}).$$
$$bb_k(\textit{Frame pointer with overwritten memory})$$

Here, bb_i denotes the dynamic memory allocation that occurs in the memory stack before a buffer overflow occurs. The Kleene star operation suggests that bb_i might be executed multiple times before bb_j is executed. Basic blocks bb_j and bb_k are similarly defined. This regular expression is then converted into a CDFG. Ideally, there should be a self-loop on bb_i, but we omit self-loops in our CDFGs so that it is a DAG. This facilitates analysis. The CDFG for buffer overflow is shown in Fig. 3.

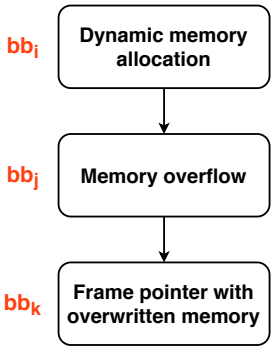

Fig. 3: CDFG of buffer overflow attacks

5.1.3 Attack DAG

Every attack in our list is represented by its corresponding CDFG. All the CDFGs are combined to form a single DAG. This is our aggregated attack DAG. This is shown in Fig. 4.

The attack DAG is a concise representation of the system and network-level operations of known categories of CPS/IoT attacks. Every path from a head node to a leaf node in the attack DAG corresponds to a unique attack vector.

We observe that certain basic blocks appear in multiple attacks. These basic blocks are represented as a single node in the attack DAG with in-degree and/or out-degree greater than 1. Our attack DAG has 37 nodes, represents 41 different attacks, and has a maximum depth of 6.

TABLE 1: Real-world CPS/IoT attacks and regular expressions

Attack	Vulnerability category	Regular expression
Therac-25 Radiation Poisoning	Race condition / TOCTOU vulnerability	bb_i(access system call)*. bb_j(open system call)*
Ariane 5 Rocket Explosion	Integer overflow	bb_i(data invariant > max integer)*
Worcester Airport Control Tower Communication Hack	Buffer overflow	bb_i(dynamic memory allocation)*.bb_j(overflow of memory).bb_k(frame pointer with overwritten memory)
Bellingham, Washington, Pipeline Rupture	Buffer overflow	bb_i(dynamic memory allocation)*.bb_j(overflow of memory).bb_k(frame pointer with overwritten memory)
Maroochy Shire Wastewater Plant Compromised	Access control/Privilege escalation	bb_i(critical component with one factor or one man authentication)*
Davis-Besse Nuclear Power Plant Worm	Malware/Privilege escalation	bb_i(critical component with one factor or one man authentication)*
Worm Cripples CSX Transport System	Malware/Privilege escalation	bb_i(critical component with one factor or one man authentication)*
Worm Cripples Industrial Plants	Malware/Privilege escalation	bb_i(critical component with one factor or one man authentication)*
Browns Ferry Nuclear Plant	Distributed Denial of Service (DDoS) attack	bb_i(port traffic per second > threshold)
LA Traffic System Attack	DDoS attack	bb_i(data invariant > threshold)
Aurora Generator Test	Protocol vulnerability	bb_i(access requested)*.bb_j(no mutual authentication)*
Internet Attack on Epileptics	SQL injection	bb_i(user input)*.bb_j(user input not compliant with database format)
Turkish Oil Pipeline Rupture	Privilege escalation / DDoS	bb_i(critical component with one factor or one man authentication)* + bb_j(data invariant > threshold)
Stuxnet Attack on Iranian Nuclear Power Facility	Malware through USB	bb_i(executive file of new executable at kernel level)*.bb_j(sending data through port to external C2)
Tests of Insulin Pumps	No authentication + No encryption Replay attacks	bb_i(transaction requested)*.bb_j(no time stamp check)*.bb_k(no mutual authentication)*.bb_l(no hash check)* bb_i(data in transit not encrypted)*
Houston, Texas, Water Distribution System Hack	Weak access management	bb_i(access requested)*.bb_j(no strong authentication, e.g., no public key infrastructure based authentication or two factor authentication)*
Researcher Defeats Key Card Locks	No authentication	bb_i(access requested)*.bb_j(no mutual authentication)*.bb_k(encryption key read from memory in unencrypted format)*
Test of Traffic Vulnerabilities	Weak cryptographic measures	bb_i(no encryption of data/commands)*+(bb_j(no digital sign on sensor firmware)*. bb_k(illegal access through unobstructed port)*. (bb_l(reconfigure the system specs)* + (bb_m(access memory buffer). bb_n(overwrite memory buffer))*))
German Steel Mill Attack	Malware/Privilege escalation	bb_h(open downloaded file from spear-phishing email)*.bb_i(executive downloaded file from email)*.bb_j(critical component with one factor or one man authentication)*.bb_k(access business network)*.bb_l(access ports of entry to production network)*.bb_m(manipulate commands to the system)*
Fatal Military Aircraft Crash Linked to Software fault	Software fault	bb_i(access system files)*.bb_j(rewrite code for updates)*.bb_k(delete/modify important system files)*
Test of Smart Rifles	Weak password	bb_i(weak WiFi password)*.(bb_j(alter state variables)*+bb_k(gain root access))
Black Energy Ukrainian Power Grid Attack	Weak authentication	bb_i(spear phishing emails to access business network)*.bb_j(maneuver into the production network)*.(bb_k(erased critical files on disk) + bb_m(took control over important network nodes)*)
Mirai Botnet Attack	Weak authentication + DDoS	bb_i(weak password)*.bb_j(port traffic per second > threshold)
Unidentified Water Distribution Facility Hack	Web vulnerabilities	(bb_i(phishing emails to access credentials)*+bb_j(SQL injection attacks to get credentials)*).bb_k(weak storage of credentials on front-end server)
WannaCry Ransomware Attacks	Buffer overflow Cryptographic key management	bb_i(dynamic memory allocation)*.bb_j(overflow of memory)*.bb_k(frame pointer with overwritten memory in SMBv1 buffer)* bb_i(process starts encrypting data)*bb_j(process new to the system and not whitelisted)*

5.2 Applying Machine Learning

Once we have represented the known attacks in the attack DAG, we observe that some of its unconnected nodes can be linked together. Every new feasible link that is predicted by the ML model is considered to be a novel exploit of vulnerabilities. A link or branch is considered to be feasible if the control data flow represented by that branch can be implemented in a real-world system. We use ML models to predict if directed branches between various pairs of nodes of the attack DAG are feasible. Manual verification of the feasibility of all possible branches in the attack DAG is too time-consuming. Let n be the number of nodes in the attack DAG and c be the number of examples in the training dataset. Then the size of the search space of possible branches is

$$\begin{aligned} 2\binom{n}{2} - c &= n(n-1) - c \\ &= n^2 - n - c \\ &= \Theta(n^2) \end{aligned} \quad (1)$$

This quadratic dependence makes it very expensive to perform manual checks to exhaustively examine the feasibility of all the possible branches. In our experiments, we show that using ML can reduce the search space by 87.5%.

We train the ML model using the attack DAG of known attack vectors. Once trained, it can predict the feasibility of

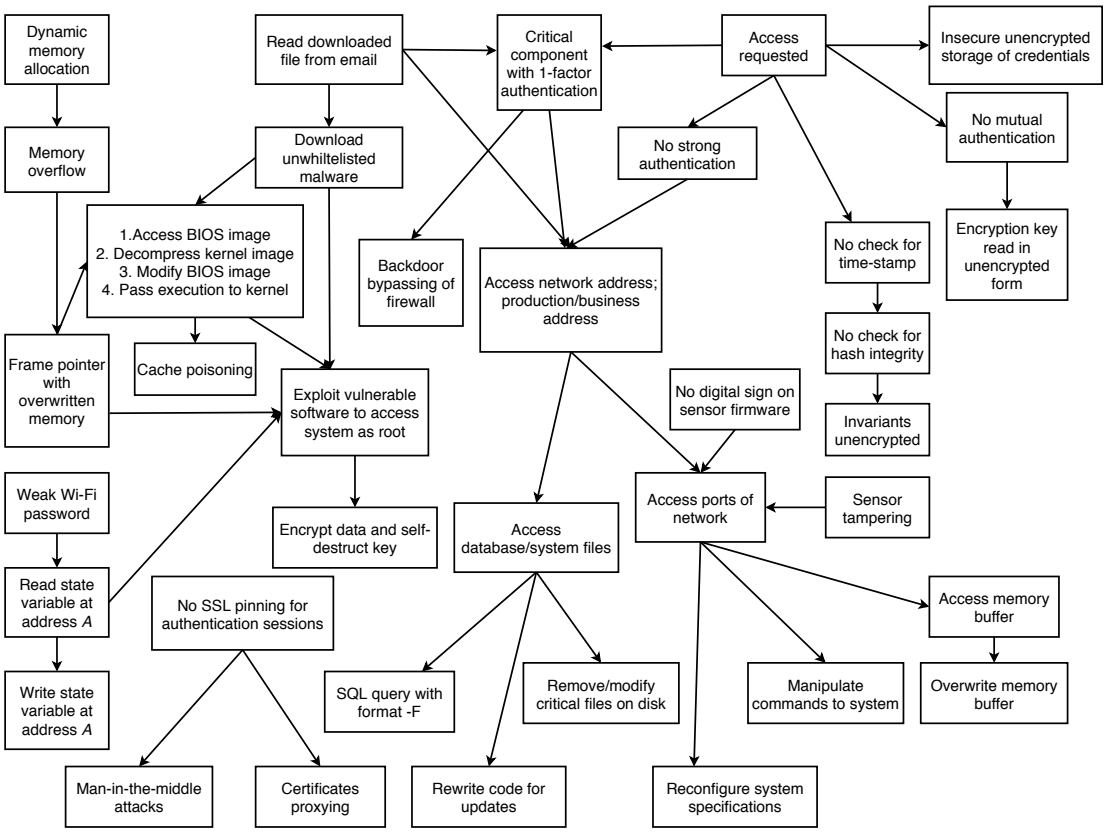

Fig. 4: The aggregated attack DAG

new branches in the attack DAG. We derive an SVM model for this purpose. Since the dataset is very small, consisting of just 140 datapoints, it prevented us from being able to adequately train a neural network [54]. However, when our methodology is applied to a larger scope of cyberattacks, a neural network model might be an effective tool [55].

5.2.1 Data Preparation

We assign various attributes (features) to the basic blocks of the attack DAG depending on the type of impact the attack would have on the system and network. The various attributes are memory, data/database, security vulnerability, port/gateway, sensor, malware, head node, leaf node, and mean depth of the node. Each node has a binary value (0 or 1) associated with every feature except the mean depth. The mean depth of a node denotes the average depth of the node in the attack DAG. For example, nodes *"Memory overflow"* and *"SQL query with format -F"* have the attributes shown in Table 2.

We represent a branch in the attack DAG by an ordered pair of nodes, i.e., (*origin node, destination node*). The features of the branches of the attack DAG are required to train the ML model. The concatenation of the attributes of the origin and destination nodes represents the feature vector of a branch.

5.2.2 Training Dataset

Our SVM model learns from the underlying patterns that exist in known CPS/IoT attacks, some of which are shown in Table 1. This knowledge is encoded in the attack DAG.

TABLE 2: Node attributes

Attribute	Memory overflow	SQL query with format -F
Memory	1	0
Data/Database	0	1
Security vulnerability	0	0
Port/Gateway	0	0
Sensor	0	0
Malware	0	0
Head node	0	0
Leaf node	0	1
Mean depth	1	3.75

Thus, the training dataset is composed of all the existing branches (positive examples) and some infeasible branches (negative examples) of the attack DAG. The labels of the training dataset are:

- 1, if the branch exists in the attack DAG.
- -1, if a branch from the origin to the destination node is not feasible.

A negatively labeled branch denotes an impossible control/data flow. Some negatively-labeled examples include branches from the leaf nodes to head nodes, branches that complete cycles in the attack DAG, and sequences of infeasible operations like exploitation of memory flow via certificate proxying.

Our training dataset consists of 140 examples, 39 of which have positive labels and the remaining have negative labels.

5.2.3 Training

The ML model has multiple parameters that can be tuned to achieve optimal performance [56]. The parameters of the SVM model that we experimentally tuned during training are mentioned below.

1) **Regularization parameter (C):** Regularization is used in ML models to prevent overfitting of the model to the training data. Overfitting causes the model to perform well on the training dataset but poorly on the test dataset. This parameter needs to be fine-tuned to obtain optimal performance of the model. The value of C is inversely proportional to the strength of regularization.
2) **Kernel:** The kernel function transforms the input vector x_i to a higher-dimensional vector space $\phi(x_i)$, such that separability of inputs with different labels increases. We use the radial basis function (RBF) as our kernel function. The RBF kernel is defined as:

$$k(x_i, x_j) = exp(-\gamma||(x_i - x_j)||^2) \quad (2)$$

3) γ : Parameter γ defines how strong the influence of each training example is on the separating hyperplane. Higher (lower) values of γ denote a smaller (larger) circle of influence.
4) **Shrinking heuristic:** The shrinking heuristic is used to train the model faster. The performance of our model does not change in the absence of this heuristic.
5) **Tolerance:** The tolerance value determines the error margin that is tolerable during training. A higher tolerance value causes early stopping of the optimization process, resulting in a higher training error. A higher tolerance value also helps in preventing overfitting.

The parameter values of our SVM model are shown in Table 3.

TABLE 3: SVM parameters

Parameter	Value
C	1.0
Kernel	RBF
γ	0.0556
Shrinking heuristic	Used
Tolerance for stopping	10^{-3}

5.2.4 Test Dataset

We use the SVM model to predict the feasibility of all possible branches of the attack DAG. Therefore, the test dataset contains all possible branches except the datapoints present in the training dataset. Our attack DAG has 37 nodes and our training set has 140 examples. Putting $n = 37$ and $c = 140$ in Eq. (1), we observe that our test dataset contains 1192 datapoints.

5.2.5 Verification

A test example is positive if the sequence of the two basic blocks is a permissible control/data flow in a given system. Determining the control/data flow in a program is generally a hard task. However, in this article, we define the basic blocks at a human-interpretative level. This makes it easier for a human expert to determine if the sequence of basic blocks in the test example is feasible or not.

The SVM model predicts 149 positive labels out of 1192 test datapoints. A positive label indicates that the test datapoint is a potential novel exploit. Manual verification of all the 1192 datapoints in the test dataset revealed that 1165 predictions by the SVM model are accurate, resulting in a test accuracy of 97.73%.

The parameters of SHARKS were chosen to achieve zero false negatives. However, our SVM model outputs a few false positives. To eliminate these false positives, manual verification is necessary. In the absence of SHARKS, a human expert would have to verify all 1192 potential vulnerability exploits manually. With the assistance of SHARKS, it is sufficient to verify only the 149 positive predictions of the SVM model. Thus, SHARKS helps reduce the search space of possible novel exploits from 1192 to 149, which is an 87.5% reduction in manual checks.

5.3 Discovering Unexploited Attack Vectors

The attack DAG has some unexploited attack vectors embedded in it that can be discovered through linear search on it. Every path from a head node to a leaf node corresponds to a unique attack vector. The attack DAG has 51 such paths. However, only 41 known attack vectors were considered while constructing the attack DAG. Thus, 10 unexploited attack vectors are obtained through a linear search of all the attack paths.

New attack vectors emerge due to the convergence of multiple attack paths at common basic block(s). Such an occurrence is illustrated in Fig. 5. Fig. 5a and Fig. 5b represent two subgraphs of the attack DAG in Fig. 4. Fig. 5c shows the graph obtained by combining Fig. 5a and Fig. 5b at the common node titled "*Access ports of network*." Fig. 5d depicts the new paths obtained from the combination of the two graphs. The five new paths thus discovered correspond to five attack vectors that have not yet been exploited in real-world CPS/IoT attacks.

5.4 Experimental Results

In this section, we present the experimental results. We begin by demonstrating why we chose an SVM model for novel exploit detection. In addition to SVM, we evaluated the following models: k-nearest neighbors (k-NN), naive Bayes, decision tree, and stochastic gradient descent (SGD) based linear SVM. We compare their accuracies, precision/recall values, false positive rates (FPR), and F1 scores in Table 4. It is clear that SVM performs the best.

Then we use SVM to predict the existence of new branches in the attack DAG. The SVM model successfully predicts the existence of 122 new feasible branches in the attack DAG. Each new branch corresponds to a unique novel exploit.

Some of the 122 feasible branches of the attack DAG that were predicted by ML are listed in Table 5. These attacks have been chosen to represent the most popular vulnerability categories. A linear search of the attack DAG also helps us discover 10 unexploited attack vectors, a subset of which is depicted in Fig. 5.

The confusion matrix in Table 6 shows the number of true negatives (TN), false positives (FP), false negatives (FN), and true positives (TP). The SVM model achieves zero

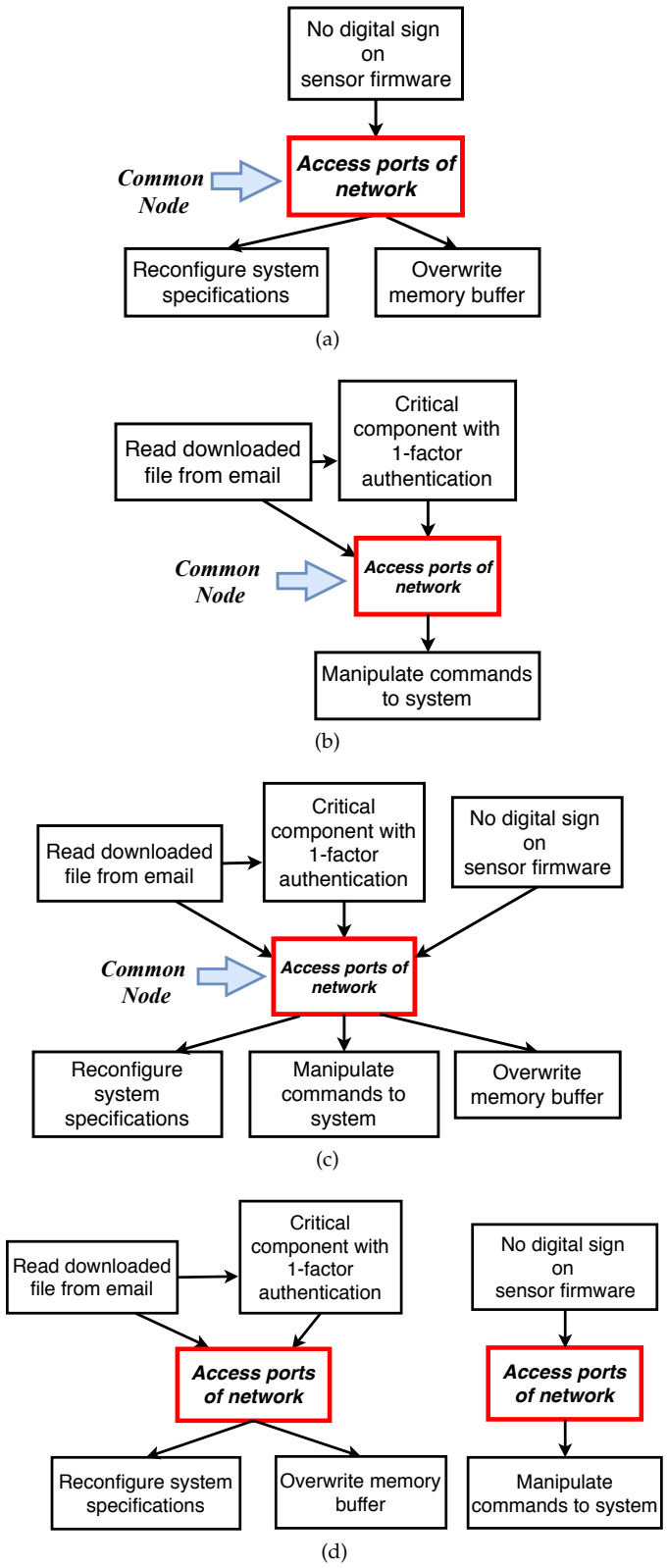

Fig. 5: Generating new exploits with linear search: (a) CDFG for $Attack_1$, (b) CDFG for $Attack_2$, (c) combined CDFG of both attacks, and (d) new attacks that emerge from a combination of the two CDFGs

FN, which indicates that a negative prediction is always correct.

In Fig. 6, we categorize the novel exploits into six categories. We can see that access control vulnerabilities

TABLE 4: Performance of ML models

Model	Accuracy	Precision	Recall	FPR	F1
Decision Tree	86.8%	0.40	0.89	0.14	0.55
k-NN (k=2)	92.8%	0.60	0.62	0.04	0.61
k-NN (k=3)	92.0%	0.54	0.88	0.08	0.67
k-NN (k=4)	94.5%	0.70	0.70	0.03	0.70
k-NN (k=5)	93.0%	0.58	0.86	0.06	0.69
Naive Bayes	90.5%	0.46	0.26	0.03	0.34
SVM (C=1)	**97.7%**	**0.82**	**1.0**	**0.03**	**0.90**
Linear SVM with SGD	90.6%	0.49	0.75	0.08	0.59
SVM (C=2)	93.8%	0.60	0.97	0.06	0.76
SVM (C=3)	92.8%	0.56	0.96	0.08	0.71

TABLE 5: Novel exploits discovered

Branch discovered	Vulnerability category
Read downloaded file from email ⟶ Overflow of memory	Buffer overflow
Access network ports ⟶ Encrypt data and destroy key	Privilege escalation
Access system files and databases ⟶ Reconfigure system specifications	Access control
Download unwhitelisted malware ⟶ Bypass firewall using backdoor	Malware
Access network address ⟶ Encryption key read from memory in unencrypted form	Cryptographic flaw
Critical component with 1-factor authentication ⟶ Access Basic Input/Output System (BIOS) image	BIOS boot level attack
Exploit malware to access system as root ⟶ Cache poisoning	Cache poisoning

TABLE 6: Confusion matrix

N=1192	**Actual = No**	**Actual = Yes**	
Predicted = No	TN = 1043	FN = 0	1043
Predicted = Yes	FP = 27	TP = 122	149
	1070	122	

(including privilege escalation), weak cryptographic primitives, and network security flaws are most common vulnerabilities with high likelihood of exploit. We also observe that vulnerabilities with lower exploit likelihood are BIOS vulnerabilities and cache poisoning attacks - due to higher complexity in building the exploit chain across various system elements. This is expected because a successful BIOS attack or a cache poisoning attack involves one or more of the following: boot-stage execution, shared resources with adversary, side-channel access, kernel code execution, and close proximity to the CPS/IoT devices at very specific time instances.

Training accuracy refers to the accuracy of the SVM model when evaluated on the training dataset. Only four of the 140 training datapoints were incorrectly classified by the SVM model, yielding an accuracy of 97.14%. The test accuracy is manually determined by evaluating the feasibility of all the 1192 possible branches in the attack DAG. We observed that 27 of the 149 positive predictions were incorrect. On the other hand, all the negative predictions were accurate. Thus, 1165 of the 1192 datapoints of the test dataset were classified correctly by the SVM model, yielding a test accuracy of 97.73%.

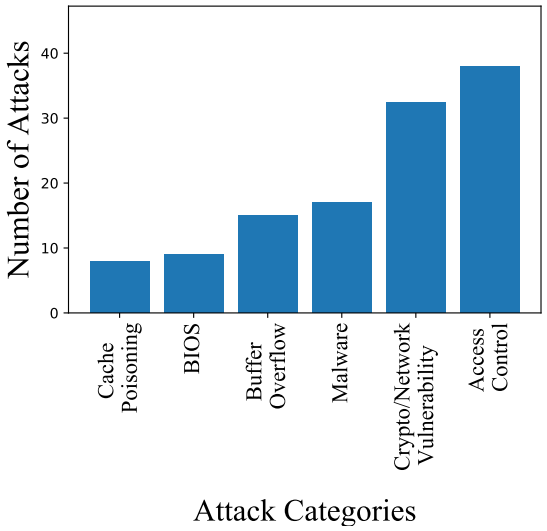

Fig. 6: Histogram depicting the number of novel exploits discovered in each category

6 IoT Case Study: Connected Car

The connected car is a complicated IoT system comprising various sensors, electronic control units (ECUs), system buses, and embedded software packages. It possesses a vast range of capabilities that includes Internet access, communication with multiple devices, and collection of real-time data from the surroundings. While these functionalities enhance user convenience, they also expand the attack surface of the system. The most common entry points for hackers are the ECUs, on-board diagnostics (OBD) port, WiFi, and GSM and bluetooth networks of the vehicle. Some of these are shown in Fig. 7.

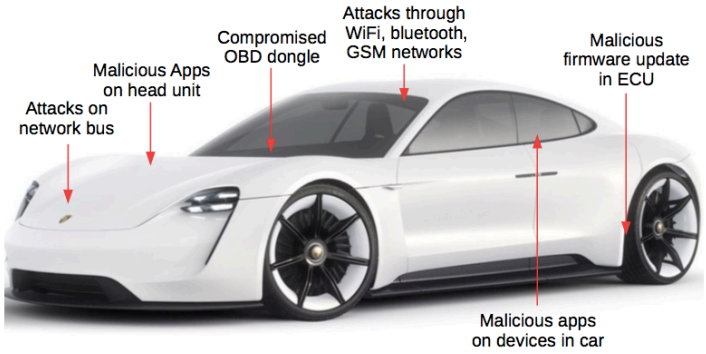

Fig. 7: Examples of hacker entry points into the connected car

The connected car has numerous ECUs that are responsible for different functionalities like anti-lock braking, lane departure warning, and engine management. All communications between ECUs occur over the network bus that connects all the ECUs to one another. There exist multiple networks for in-vehicle communications. Some of them are local interconnect network [57], FlexRay network [58], and media-oriented systems transport network [59]. One of the most popular in-vehicle networks is the Controller Area Network (CAN) [60]. CAN ensures real-time handling of all in-vehicle communications, including safety-critical data. This makes the security of the CAN bus critical to the safety and security of the smart vehicle. However, the CAN bus has been shown to be intrinsically insecure [61], [62]. Cryptographic techniques like encryption and message authentication cannot be applied to the data traversing the CAN bus. These operations increase the latency of processing the packets that leads to an increased ECU response time. This overhead is not permissible in the case of safety-critical, time-sensitive, and real-time applications. Cryptographic measures also prevent car mechanics from analyzing CAN traffic during troubleshooting. This is a major inconvenience for them because they generally use the CAN bus as a diagnostic tool during repair.

6.1 CAN Bus Vulnerabilities

Although CAN is the de-facto in-vehicle network in connected vehicles, it is insecure by design. The CAN protocol uses a broadcast mechanism for communication. Due to the absence of sender and receiver addresses in the data frames, every ECU can freely publish and receive messages from the bus. While this enables easier addition of new ECUs to the network, it poses a grave security threat to the system. We next discuss the popular vulnerabilities on the CAN bus that were detected by our approach.

1) **Frame sniffing:** The CAN protocol uses a broadcasting mechanism for ECU communications. This allows a malicious node on the CAN bus to receive all the data frames through sniffing. The absence of encryption makes it easier to analyze the collected frames. The range of valid messages on the CAN bus is small enough to be exhaustively analyzed. Fuzzing techniques can be used to decode the functionalities of various ECUs from the log of sniffed frames [63]. This is a breach of confidentiality of the system. Frame sniffing is often the precursor of more complex attacks.
2) **Frame spoofing:** Frame spoofing involves sniffing and reverse engineering of the data frames of the CAN bus. Using the details of the data frames, the adversary can broadcast malicious frames on the bus by spoofing a particular node. Absence of authentication schemes compromises the integrity of messages on the CAN bus. Spoofing attacks may result in incorrect speedometer readings, arbitrary acceleration of the vehicle, erroneous fuel level readings, and conveying malicious messages to the driver [64]. This poses grave safety concerns as the adversary can gain access to safety-critical ECUs like the braking system and engine management system.
3) **Denial of Service (DoS):** The CAN protocol implements a priority-based broadcasting communication scheme. For example, messages from the anti-lock braking system, which are critical to the safety of the passengers, are given higher priority for transmission on the bus than messages from climate control sensors.

The priority of a frame is determined by a parameter id (PID). Lower values of PID signify higher priority messages. To launch a DoS attack, the adversary needs to decode the smallest acceptable value of PID from the history of CAN messages (obtained by frame sniffing). Then he can continually broadcast messages with the highest priority on the bus, thus preventing any other message from being transmitted on it [63]. This compromises the availability of the CAN bus to legitimate messages, thus denying service to these messages.

4) **Replay attack:** Replay attacks involve sniffing the frames on the CAN bus prior to launching the attack. Sniffing and analyzing the frame packets using fuzzing techniques reveal knowledge about the frame functionalities. Since the CAN protocol is bereft of authentication schemes and time-stamp verification, the recorded frame packets can be sent on the CAN bus at inconvenient time instances to launch various attacks. For example, the frame packet to unlock the car door can be replayed by a thief when the owner is not around. Replay attacks on cars have been demonstrated both in simulations [65] and real cars [63].

The other vulnerabilities that we consider in our experiments are ECU buffer overflows [66] and malware injection through ECU firmware updates [67]. These attack vectors involve sending malicious packets to the ECUs over the CAN bus but do not involve exploiting any vulnerability of the CAN bus itself.

6.2 Application of SHARKS

In this section, we describe how we use our SHARKS approach to detect the aforementioned vulnerabilities in the given IoT system, namely the CAN bus. In our threat model, we assume that the attacker has already gained access to the internal CAN bus by compromising a gateway ECU. An adversary can gain access to the CAN bus through multiple entry points like the OBD port, WiFi, bluetooth or the GPS system of the car [68]. In our simulations, we assume that the OBD-II port was used to gain access to the CAN bus. We simulate the CAN bus with OpenGarages ICSim simulator [68] with LibSDL and Socket-CAN CAN-utils libraries.

To apply SHARKS to a specific CPS/IoT system, we have to design the attack DAG for it. The attack DAG shown in Fig. 4 is designed for a generic CPS/IoT system. The CAN bus has fewer functionalities than those considered during the design of the attack DAG in Fig. 4. This makes some of the nodes in the DAG in Fig. 4 redundant with respect to the CAN bus IoT system. We remove those nodes and obtain a subgraph of Fig. 4 that is relevant to the CAN bus. This subgraph, shown in Fig. 8, is referred to as the CAN attack DAG. It has 25 nodes, 19 branches, and denotes 14 high-level attack vectors relevant to the CAN bus.

6.3 Results

We run a pre-trained SVM model on the CAN attack DAG shown in Fig. 8. The SVM model was trained on the attack DAG in Fig. 4, and not on the CAN attack DAG. While testing the model's performance on the CAN attack DAG, we observe that it is able to discover 45 CAN vulnerability exploits that were initially absent in the CAN attack DAG. This indicates that our approach is generic enough to be deployed on any CPS/IoT system for vulnerability exploit detection. We classify the detected CAN bus vulnerabilities into the vulnerability categories mentioned in Section 6.1. Some of the attack branches predicted by the model and their corresponding categories are shown in Table 7.

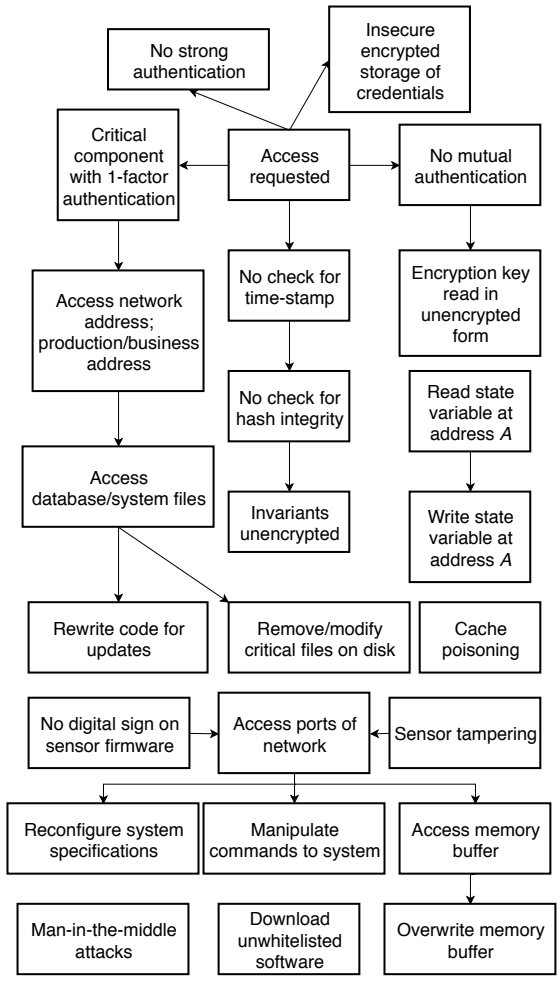

Fig. 8: The attack DAG for the CAN bus of the connected vehicle

TABLE 7: Novel exploits discovered

Branch discovered	Vulnerability category
Invariants unencrypted ⟶ Read state variable at address A	Frame sniffing
No mutual authentication ⟶ Man-in-the-middle attacks	Frame spoofing
No check for time-stamp ⟶ Manipulate commands to system	Replay attack
Access memory buffer ⟶ Write state variable at address A	ECU buffer overflow
Rewrite code for updates ⟶ Download unwhitelisted software	Malware injection through ECU updates

The CAN attack DAG has 25 nodes and 19 branches. Putting $n = 25$ and $c = 19$ in Eq. 1, we observe that there are 581 datapoints in the test set. The SVM model predicts 95 of these to be feasible novel exploits and eliminates the rest. Manually examining the feasibility of the 95 positive

predictions, we find that 45 of them were correct. All the branches that were predicted to be negative are infeasible control/data flows. Hence, the SVM model reduced our search space from 581 to 95, which represents a 83.65% reduction in human effort. The confusion matrix of the predictions by the model is shown in Table 8.

TABLE 8: Confusion matrix of SHARKS on CAN vulnerabilities

N=581	**Actual = No**	**Actual = Yes**	
Predicted = No	TN = 486	FN = 0	486
Predicted = Yes	FP = 50	TP = 45	95
	536	45	

7 Security Measures

In this section, the primary endeavor is to defend CPS/IoT against all known attacks and the novel exploits predicted by SHARKS at an optimal cost. Defense-in-depth and multi-level security (MLS) [69], [70] are the most appropriate schemes to adopt in such a scenario. Defense-in-depth refers to employing multiple defense strategies against a single weakness and is one of the seven properties of highly secure devices [71]. MLS categorizes data/resources into one of the following security levels: *Top Secret*, *Secret*, *Restricted*, and *Unclassified*. The first three levels have classified resources and require different levels of protection. Security measures become stricter as we move from Restricted to Top Secret. Many different policies can be employed to implement MLS in an organization. Some of the most popular policies are based on the Bell-La Padula (BLP) model [72] and the Biba model [73]. The BLP model prioritizes data confidentiality whereas the Biba model gives more importance to integrity.

The aggregated attack DAG is composed of multiple categories of attacks that are weaved together. Defense mechanisms can be systematically developed for each of these vulnerability categories in the form of defense DAGs. Defense DAGs mirror the corresponding attack subgraphs and make execution of the key basic blocks of the attack sequence infeasible. This ensures that no path from a head node to a leaf node in the attack DAG can be traversed in the presence of the suggested defense measures.

Many attacks have multiple defense strategies that can protect against them. The cost of our overall defense strategy increases with the complexity and number of defense measures that we enforce. Defense-in-depth helps us optimize this cost. The less sensitive resources (those belonging to the Restricted level) have basic defense measures against all attacks. As we move up the hierarchy to the Secret and Top Secret levels, we have more layers of security. Next, we demonstrate our defense strategies against access control and boot-stage attacks.

7.1 Defense against Access Control Attacks

Access control and privilege escalation attacks are the most common amongst real-world CPS/IoT attacks, as shown in Fig. 6. Access control attacks involve an unauthorized entity gaining access to a classified resource, thus compromising its confidentiality and/or integrity. Privilege escalation attacks involve an entity exploiting a vulnerability to gain elevated access to resources that it is not permitted to access. Implementation of strict policies can protect against such attacks. These security policies include multi-factor authentication, access control lists, role-based access control, and SQL queries input validation. More layers of authentication, authorization, and network masking can be added for more sensitive resources. An example of a defense DAG is shown in Fig. 9.

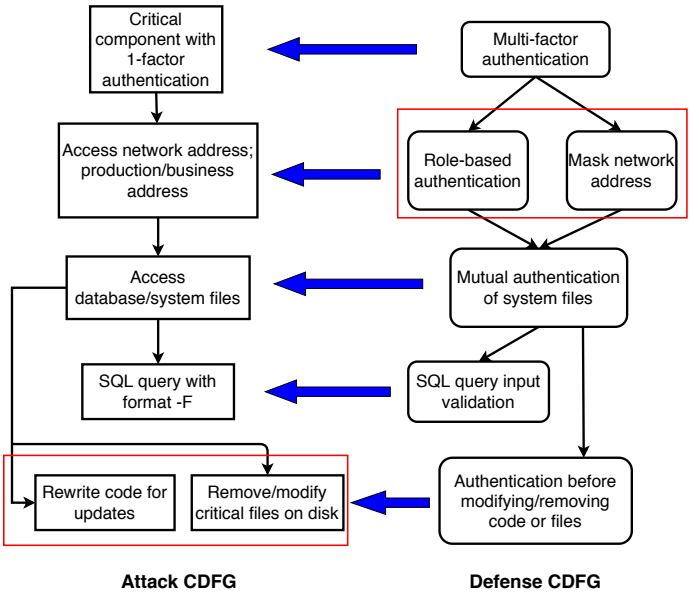

Fig. 9: Defense at the **Top Secret** level against access control and privilege escalation exploits. The CDFG on the left depicts the attack CDFG and the CDFG on the right depicts the defense CDFG. The arrows indicate the basic blocks of the defense CDFG making the corresponding basic blocks of the attack CDFG non-operational.

7.2 Defense against Boot-stage Attacks

This category of attacks is the most complicated among all the categories. While other attacks can be launched at the application level, these attacks typically require root access and have to be launched at the system level.

To defend against such attacks, a Core Root of Trust for Measurement is required along with a Trusted Platform Module (TPM) or a Hardware Security Module. These are generally present at a level lower than the kernel and sometimes referred to as the Trusted Computing Base (TCB). In Fig. 10, the BOOTROM serves as the TCB. Defense against this attack involves a series of hierarchical and chained hash checks of binary files and secret keys stored in the Platform Configuration Register (PCR) of the TPM. The PCR is inaccessible to all entities except the TPM. The detection of an incorrect hash value at any stage of the boot sequence causes the boot sequence to halt due to the detection of an illegal modification of the binary boot files and/or the secret(s). SHA-2 is the most commonly used hash function at this stage. Fig. 10 gives an overview of the hash checks and execution of binary files at various levels.

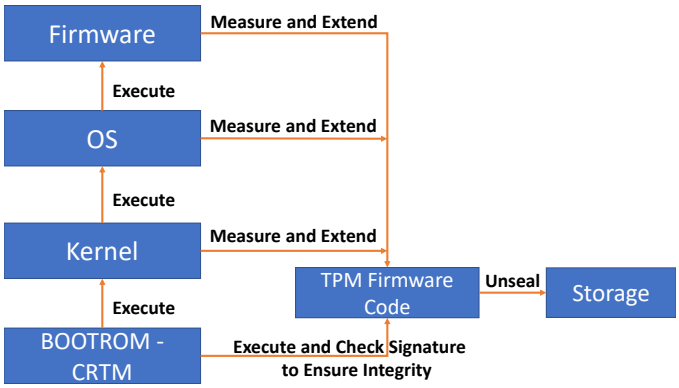

Fig. 10: Defensive measures against Boot-stage attacks

8 Conclusion

The rapid advancement of CPS/IoT-enabling technologies, like 5G communication systems and ML, increases the scope of their applications manifold. Unfortunately, this also increases the attack surface of such systems that can often result in catastrophic effects. We have demonstrated how ML can be used at the system and network levels to detect possible vulnerabilities (and their corresponding exploits) across the hardware, software, and network stacks of CPS/IoT. We discovered 10 unexploited attack vectors and 122 novel exploits using the proposed methodology and suggested appropriate defense measures to implement a tiered-security mechanism. We hope that this methodology will prove to be helpful for proactive threat detection and incident response in different types of CPS/IoT frameworks.

Acknowledgments

The authors would like to thank NSF for supporting this work under Grant CNS-1617628.

References

[1] Andrea Zanella, Nicola Bui, Angelo Paolo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of Things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32, 2014.

[2] Hamidreza Arasteh, Vahid Hosseinnezhad, Vincenzo Loia, Aurelio Tommasetti, Orlando Troisi, Miadreza Shafie-Khah, and Pierluigi Siano. IoT-based smart cities: A survey. In *Proc. IEEE Int. Conf. Environment and Electrical Engineering*, pages 1–6, 2016.

[3] Ayten Ozge Akmandor and Niraj K. Jha. Smart health care: An edge-side computing perspective. *IEEE Consumer Electronics Magazine*, 7(1):29–37, 2018.

[4] Biljana L. Risteska Stojkoska and Kire V. Trivodaliev. A review of Internet of Things for smart home: Challenges and solutions. *J. Cleaner Production*, 140:1454–1464, 2017.

[5] Ruonan Zhang and Xinbao Liu. IoT-based maintenance process design for fusion reactor remote handling system. *J. Fusion Energy*, 33(6):653–657, 2014.

[6] Miao Yun and Bu Yuxin. Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid. In *Proc. IEEE Int. Conf. Advances in Energy Engineering*, pages 69–72, 2010.

[7] Abdul Rahman Al-Ali and Raafat Aburukba. Role of Internet of Things in the smart grid technology. *J. Computer and Communications*, 3(05):229, 2015.

[8] Soumya Kanti Datta, Rui Pedro Ferreira Da Costa, Jérôme Härri, and Christian Bonnet. Integrating connected vehicles in Internet of Things ecosystems: Challenges and solutions. In *Proc. IEEE Int. Symp. A World of Wireless, Mobile and Multimedia Networks*, pages 1–6, 2016.

[9] Xin Huang, Paul Craig, Hangyu Lin, and Zheng Yan. SecIoT: A security framework for the Internet of Things. *Security and Communication Networks*, 9(16):3083–3094, 2016.

[10] Arsalan Mosenia and Niraj K. Jha. A comprehensive study of security of Internet-of-Things. *IEEE Trans. Emerging Topics in Computing*, 5(4):586–602, 2017.

[11] Teng Xu, James B. Wendt, and Miodrag Potkonjak. Security of IoT systems: Design challenges and opportunities. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pages 417–423, 2014.

[12] Masanobu Katagi and Shiho Moriai. Lightweight cryptography for the Internet of Things. *Sony Corporation*, pages 7–10, 2008.

[13] Jun-Ya Lee, Wei-Cheng Lin, and Yu-Hung Huang. A lightweight authentication protocol for Internet of Things. In *Proc. Int. Symp. Next-Generation Electronics*, pages 1–2, 2014.

[14] Gookwon Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proc. Design Automation Conference*, pages 9–14, 2007.

[15] Vikash Sehwag and Tanujay Saha. TV-PUF: A fast lightweight analog physical unclonable function. In *Proc. IEEE Int. Symp. Nanoelectronic and Information Systems*, pages 182–186, Dec. 2016.

[16] Hui Suo, Jiafu Wan, Caifeng Zou, and Jianqi Liu. Security in the Internet of Things: A review. In *Proc. Int. Conf. Computer Science and Electronics Engineering*, volume 3, pages 648–651, Mar. 2012.

[17] Ayten Ozge Akmandor, Hongxu Yin, and Niraj K. Jha. Smart, secure, yet energy-efficient, Internet-of-Things sensors. *IEEE Trans. Multi-Scale Computing Systems*, 4(4):914–930, 2018.

[18] Liang Xiao, Xiaoyue Wan, Xiaozhen Lu, Yanyong Zhang, and Di Wu. IoT security techniques based on machine learning: How do IoT devices use AI to enhance security? *IEEE Signal Processing Magazine*, 35(5):41–49, Sep. 2018.

[19] Fady Copty, Andre Kassis, Sharon Keidar-Barner, and Dov Murik. Deep ahead-of-threat virtual patching. In *Proc. Int. Wkshp. Information and Operational Technology Security Systems*, pages 99–109, 2018.

[20] Chunlin Zhang, Ju Jiang, and Mohamed Kamel. Intrusion detection using hierarchical neural networks. *Pattern Recognition Letters*, 26(6):779–791, 2005.

[21] James Cannady. Artificial neural networks for misuse detection. In *Proc. Nat. Inf. Syst. Secur. Conf.*, pages 443–456, 1998.

[22] Alan Bivens, Chandrika Palagiri, Rasheda Smith, Boleslaw Szymanski, and Mark Embrechts. Network-based intrusion detection using neural networks. *Intelligent Engineering Systems through Artificial Neural Networks*, 12(1):579–584, 2002.

[23] Carl Livadas, Robert Walsh, David Lapsley, and W. Timothy Strayer. Using machine learning techniques to identify botnet traffic. In *Proc. IEEE Conf. Local Computer Networks*, pages 967–974, 2006.

[24] Salem Benferhat, Tayeb Kenaza, and Aicha Mokhtari. A naive Bayes approach for detecting coordinated attacks. In *Proc. Annual IEEE Int. Computer Software and Applications Conference*, pages 704–709, 2008.

[25] Gilbert R. Hendry and Shanchieh J. Yang. Intrusion signature creation via clustering anomalies. In *Proc. Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security*, volume 6973, page 69730C, 2008.

[26] Misty Blowers and Jonathan Williams. Machine learning applied to cyber operations. In *Network Science and Cybersecurity*, pages 155–175. 2014.

[27] Karlton Sequeira and Mohammed Zaki. Admit: Anomaly-based data mining for intrusions. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pages 386–395, 2002.

[28] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. EXPOSURE: Finding malicious domains using passive DNS analysis. In *Proc. Symp. Network and Distributed Systems Security*, pages 1–17, 2011.

[29] Christopher Kruegel and Thomas Toth. Using decision trees to improve signature-based intrusion detection. In *Proc. Int. Wkshp. Recent Advances in Intrusion Detection*, pages 173–191, 2003.

[30] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: Detecting botnet command and control servers through large-scale netflow analysis. In *Proc. ACM Annual Computer Security Applications Conference*, pages 129–138, 2012.

[31] Farnaz Gharibian and Ali A Ghorbani. Comparative study of supervised machine learning techniques for intrusion detection. In *Proc. IEEE Annual Conference on Communication Networks and Services Research*, pages 350–358, 2007.

[32] André Årnes, Fredrik Valeur, Giovanni Vigna, and Richard A. Kemmerer. Using hidden Markov models to evaluate the risks of intrusions. In *Proc. Int. Wkshp. on Recent Advances in Intrusion Detection*, pages 145–164, 2006.

[33] Fatemeh Amiri, Mohammad Mahdi Rezaei Yousefi, Caro Lucas, Azadeh Shakery, and Nasser Yazdani. Mutual information-based feature selection for intrusion detection systems. *Journal of Network and Computer Applications*, 34(4):1184–1199, 2011.

[34] Yinhui Li, Jingbo Xia, Silan Zhang, Jiakai Yan, Xiaochuan Ai, and Kuobin Dai. An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert Systems with Applications*, 39(1):424–430, 2012.

[35] Hongyu Chen and Li Jiang. GAN-based method for cyber-intrusion detection. *CoRR*, abs/1904.02426, 2019.

[36] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. N-BaIoT: Network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22, 2018.

[37] Vivek Shandilya, Chris B. Simmons, and Sajjan Shiva. Use of attack graphs in security systems. *Journal of Computer Networks and Communications*, 2014.

[38] Somesh Jha, Oleg Sheyner, and Jeannette M. Wing. Two formal analyses of attack graphs. In *Proc. IEEE Computer Security Foundations Wkshp.*, pages 49–63, June 2002.

[39] M. Ugur Aksu, Kemal Bicakci, M. Hadi Dilek, A. Murat Ozbayoglu, and E. ısla m Tatli. Automated generation of attack graphs using NVD. In *Proc. ACM Conf. Data and Application Security and Privacy*, pages 135–142, 2018.

[40] Liang Lu, Rei Safavi-Naini, Markus Hagenbuchner, Willy Susilo, Jeffrey Horton, Sweah Liang Yong, and Ah Chung Tsoi. Ranking attack graphs with graph neural networks. In *Proc. Int. Conf. Information Security Practice and Experience*, pages 345–359, 2009.

[41] Mehdi Yousefi, Nhamo Mtetwa, Yan Zhang, and Huaglory Tianfield. A reinforcement learning approach for attack graph analysis. In *Proc. IEEE Int. Conf. Trust, Security and Privacy In Computing and Communications*, pages 212–217, Aug. 2018.

[42] Laszlo Szekeres, Mathias Payer, Tao Wei, and Dawn Song. SoK: Eternal war in memory. In *Proc. IEEE Symp. Security and Privacy*, pages 48–62, 2013.

[43] Yuhan Gao, Liwei Chen, Gang Shi, and Fei Zhang. A comprehensive detection of memory corruption vulnerabilities for C/C++ programs. In *Proc. IEEE Int. Symp. Parallel & Distributed Processing with Applications*, pages 354–360, 2018.

[44] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *Proc. IEEE Symp. Security and Privacy*, pages 1–19, 2019.

[45] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *arXiv preprint arXiv:1801.01207*, 2018.

[46] Caroline Trippel, Daniel Lustig, and Margaret Martonosi. Checkmate: Automated synthesis of hardware exploits and security litmus tests. In *Proc. IEEE/ACM Int. Symp. Microarchitecture*, pages 947–960, 2018.

[47] Zvi Kohavi and Niraj K. Jha. *Switching and Finite Automata Theory, 3rd ed.*, 2009.

[48] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[49] Melanie Ooi. Future trend in I&M: The smarter car. *IEEE Instrumentation & Measurement Magazine*, 22(2):33–34, 2019.

[50] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. Towards blockchain-based auditable storage and sharing of IoT data. In *Proc. ACM Cloud Computing Security Wkshp.*, pages 45–50, 2017.

[51] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3):49–51, 2011.

[52] Abdulmalek Humayed, Jinqiang Lin, Fengjun Li, and Bo Luo. Cyber-physical systems security: A survey. *IEEE Internet of Things Journal*, 4(6):1802–1831, Dec. 2017.

[53] Najwa Aaraj, Anand Raghunathan, and Niraj K. Jha. Dynamic binary instrumentation-based framework for malware defense. In *Proc. Int. Conf. Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 64–87, 2008.

[54] Fahimeh Ghasemi, Alireza Mehridehnavi, Alfonso Perez-Garrido, and Horacio Perez-Sanchez. Neural network and deep-learning algorithms used in QSAR studies: Merits and drawbacks. *Drug Discovery Today*, 2018.

[55] Shayan Hassantabar, Zeyu Wang, and Niraj K. Jha. SCANN: Synthesis of compact and accurate neural networks. *arXiv preprint arXiv:1904.09090*, 2019.

[56] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[57] Matthew Ruff. Evolution of local interconnect network (LIN) solutions. In *Proc. IEEE Vehicular Technology Conference*, volume 5, pages 3382–3389, 2003.

[58] Rainer Makowitz and Christopher Temple. FlexRay: A communication network for automotive control systems. In *Proc. IEEE Int. Wkshp. Factory Communication Systems*, pages 207–212, 2006.

[59] Bogdan T. Fijalkowski. Media oriented system transport (MOST) networking. In *Automotive Mechatronics: Operational and Practical Issues*, pages 73–74, 2011.

[60] Shane Tuohy, Martin Glavin, Ciarán Hughes, Edward Jones, Mohan Trivedi, and Liam Kilmartin. Intra-vehicle networks: A review. *IEEE Trans. Intelligent Transportation Systems*, 16(2):534–545, 2014.

[61] Omid Avatefipour, Azeem Hafeez, Muhammad Tayyab, and Hafiz Malik. Linking received packet to the transmitter through physical-fingerprinting of controller area network. In *Proc. IEEE Wkshp. on Information Forensics and Security (WIFS)*, pages 1–6, 2017.

[62] Wonsuk Choi, Hyo Jin Jo, Samuel Woo, Ji Young Chun, Jooyoung Park, and Dong Hoon Lee. Identifying ECUs using inimitable characteristics of signals in controller area networks. *IEEE Trans. Vehicular Technology*, 67(6):4757–4770, 2018.

[63] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. Experimental security analysis of a modern automobile. In *Proc. IEEE Symp. Security and Privacy*, pages 447–462, 2010.

[64] Jiajia Liu, Shubin Zhang, Wen Sun, and Yongpeng Shi. In-vehicle network attacks and countermeasures: Challenges and future directions. *IEEE Network*, 31(5):50–58, 2017.

[65] Tobias Hoppe and Jana Dittman. Sniffing/replay attacks on CAN buses: A simulated attack on the electric window lift classified using an adapted CERT taxonomy. In *Proc. 2nd Wkshp. Embedded Systems Security*, pages 1–6, 2007.

[66] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *Proc. USENIX Security*, pages 1–16, 2011.

[67] Dennis K. Nilsson and Ulf E. Larson. Conducting forensic investigations of cyber attacks on automobile in-vehicle networks. *Int. Journal of Digital Crime and Forensics*, 1(2):28–41, 2009.

[68] Bryson R. Payne. Car hacking: Accessing and exploiting the CAN bus protocol. *Journal of Cybersecurity Education, Research and Practice*, 2019(1):5, 2019.

[69] Department of Defense: Washington DC. Security requirements for automatic data processing (ADP) systems. *DoD Directive 5200.28*, Dec. 1972.

[70] Department of Defense: Washington DC. Techniques and procedures for implementing deactivating testing and evaluating secure resource-sharing ADP systems. *DoD 5200.28-M*, Jan. 1973.

[71] Galen Hunt, George Letey, and Ed Nightingale. The seven properties of highly secure devices. *Tech. Rep. MSR-TR-2017-16*, 2017.

[72] John Rushby. The Bell and La Padula security model. *Computer Science Laboratory, SRI International, Menlo Park, CA*, 1986.

[73] Kenneth J Biba. Integrity considerations for secure computer systems. Technical report, MITRE Corp., Bedford, MA, 1977.

Tanujay Saha Tanujay Saha is currently pursuing his Ph.D. degree at Princeton University, NJ, USA. He received his Master's Degree in Electrical Engineering from Princeton University and Bachelors in Technology in Electronics and Electrical Communications Engineering from Indian Institute of Technology, Kharagpur, India in 2017. He has held research positions in various organizations and institutes like Intel Corp., KU Leuven, and Indian Statistical Institute. His research interests lie at the intersection of IoT, cybersecurity, machine learning, embedded systems, and cryptography.

Najwa Aaraj Dr. Najwa Aaraj is a Chief Research Officer at the UAE Technology Innovation Institute. She holds a Ph.D. in Electrical Engineering from Princeton University and a Bachelors in Computer and Communications Engineering from the American University in Beirut. Her expertise lies in applied cryptography, trusted platforms, secure embedded systems, software exploit detection/prevention, and biometrics. She has over 15 years of experience working in the United States, Australia, Middle East, Africa, and Asia with global firms. She has two patents and 15 academic publications. She has worked in a cybersecurity start-up (DarkMatter). Prior to joining DarkMatter, she worked at Booz & Company, where she led consulting engagements in the communication and technology industry for clients across four continents. She has also held research positions at IBM T. J. Watson Center, New York, Intel Security Research Group, Portland, Oregon, and NEC Laboratories, Princeton, New Jersey.

Neel Ajjarapu Neel Ajjarapu is currently pursuing his B.S.E in the Department of Electrical Engineering at Princeton University, with a concentration in security and privacy as well as a certificate in technology and society from the Center for Information Technology Policy and Keller Center for Entrepreneurship. He has held intern positions at Microsoft Corp. and One Million Metrics Corp. (Kinetic) in product management and hardware engineering. His current research interests focus on automotive security, embedded systems, and cybersecurity.

Niraj K. Jha Niraj K. Jha received the B.Tech. degree in electronics and electrical communication engineering from I.I.T., Kharagpur, India, in 1981, and the Ph.D. degree in electrical engineering from the University of Illinois at Urbana-Champaign, Illinois, in 1985. He has been a faculty member of the Department of Electrical Engineering, Princeton University, since 1987. He was given the Distinguished Alumnus Award by I.I.T., Kharagpur. He has also received the Princeton Graduate Mentoring Award. He has served as the editor-in-chief of the IEEE Transactions on VLSI Systems and as an associate editor of several other journals. He has co-authored five books that are widely used. His research has won 20 best paper awards or nominations. His research interests include smart healthcare, cybersecurity, machine learning, and monolithic 3D IC design. He has given several keynote speeches in the area of nanoelectronic design/test and smart healthcare. He is a fellow of the IEEE and ACM.