An Adversarial Attack on Artificial Intelligence Malware Detection in Consumer Internet of Things

Chi-Hsin Yang, Bernard Mwangi Maina, Shin-Ming Cheng, Hahn-Ming Lee
National Taiwan University of Science and Technology

***Abstract*—The proliferation of the Internet of Things (IoT) and Artificial Intelligence (AI) has transformed traditional consumer electronics (CEs) into next-generation devices with enhanced intelligence and connectivity. However, this advancement has exposed CEs to cybersecurity threats such as IoT botnets. Consequently, researchers are employing AI for proactive threat detection and prevention. Unfortunately, AI algorithms are vulnerable to adversarial attacks, necessitating robustness studies, as evaded malware can cause significant damage to already susceptible IoT CEs. This paper presents a case study to evaluate the resilience of AI-based IoT malware detection systems against adversarial attacks. Specifically, our method involves inserting crafted binary code snippets (payloads) into the empty regions of malware executables. We leverage explainable AI (XAI) techniques to guide payload generation, coupled with an optimization procedure to efficiently identify optimal payload sequences. Our method, tested on real-world IoT datasets, yields a robust hybrid detection system with a detection rate of up to 99.11%. Our attack approach achieves evasion rates of up to 100% and generates transferable adversarial examples. The generated samples evade a prominent structural IoT malware detector with an evasion rate of 95.15% at a minimal attack cost. This study underscores the importance of enhancing the robustness of AI-based malware detection systems and implementing diverse strategies to safeguard consumer IoT devices.**

THE ADVANCEMENT of the Internet of Things (IoT) and Artificial Intelligence (AI) has led to a surge in interconnected, smart consumer electronics

Digital Object Identifier 10.1109/MCE.YYYY.Doi Number

Date of publication DD MM YYYY; date of current version DD MM YYYY

(CEs), revolutionizing how we interact with devices and enhancing our daily lives. From smart home assistants that manage our schedules and control household appliances to wearable fitness trackers that monitor our health in real time, IoT-enabled CEs have become integral components of modern living [1]. However, with this increased connectivity comes heightened cybersecurity concerns [2]. As these devices collect and exchange vast amounts of personal data, there is a growing need for robust cybersecurity measures to protect against potential threats such as IoT botnets [2]–[4]. Cybersecurity researchers employ AI for proactive threat detection and prevention [5], analyzing both benign and malicious files to extract features for training machine learning (ML) malware detectors. File executables can be represented as binary sequences [6], grayscale images [7], or analyzed for semantic and structural features. Structure-based features, such as Control Flow Graph (CFG) [8], [9] and Function Call Graph (FCG) [10], [11], have gained traction in IoT malware detection due to their ability to detect malware across diverse CPU architectures

Machine learning malware detectors are particularly vulnerable to adversarial attacks, where subtle perturbations to malware samples are designed to mislead the detector during training (poisoning attacks) [12] or induce misclassification of malware as benign during testing [3], [13], [14]. These vulnerabilities necessitate robustness studies of ML detectors, as undetected malware can cause substantial harm. Unlike in image classification [15], adversarial attacks in malware detection must preserve the original malicious functionality [3], making such attacks more challenging.

Numerous studies exist on adversarial attacks on malware detection, primarily focusing on Android [16]–[18] and Windows [19]–[21] malware detection. However, only a few of these studies focus on IoT malware detection [3], [14], and most are limited to code-level attacks, which may not be practical. To address this gap, we aim to conduct a novel binary-level adversarial attack on structure-based IoT malware detection by injecting benign payloads into existing empty spaces in the binary. Our approach targets structural IoT malware detection, as existing literature [8], [10], [11], [22], [23] demonstrates that most IoT detectors rely on structural features. Unlike opcode or byte sequences, high-level graphical features such as FCGs and CFGs are architecture-independent, making them particularly effective for detecting cross-architecture malware in IoT devices, as demonstrated by [11].

This study evaluates the resilience of machine-learning malware detectors used in consumer electronics and IoT devices against adversarial attacks. We disassemble IoT binaries, construct control flow graphs (CFGs), and extract features for training detectors. Using explainability analysis, we devise payloads that alter the file structure and evade detection. Our attack achieves evasion rates as high as 100%, and the generated samples are transferable to a prominent IoT malware detector [8] with a 95.15% success rate. Our main contributions are summarized below.

1) We introduce a stealthy binary-level, functionality-preserving adversarial attack on structural IoT malware detection. Unique to our work, we apply SHAP and LIME explainability techniques to guide the generation of targeted payloads to alter the malware binary and evade detection.
2) We build a hybrid target detector comprising structural and opcode features, achieving a detection rate of up to 99.11%. We conducted extensive experiments to assess and compare its robustness against structural and opcode-based detectors.
3) We devise unique payloads that evade detection at a low attack cost and generate transferable adversarial samples that evade a prominent IoT malware detector [8] with a 95.15% success rate. We make our code publicly available online.

RELATED WORK

Machine Learning (ML) Malware Detection

ML technologies have been widely applied in malware detection in IoT devices and edge consumer electronics [7], [34], [35]. In ML malware detection, features can be categorized into binary-based, signature-based, and structure-based. Binary-based features represent executables as byte sequences [5], [6] or even as grayscale images [7], [34].

Signature-based features are extracted from executables through disassembly. Typically, binary files are reverse-engineered and analyzed to extract features such as opcodes, API/System Calls, Header information, and Strings [5], [35].

Structure-based features graphically depict the re-

Table 1. Comparison of the Proposed Method with Related Works

Method	Attack Level	Platform	Detector	Detector Feature	Manipulation
GEA [3],SGEA [14]	Code, Binary	IoT	CNN	CFG	Benign Subgraph Injection
AndroidHIV [16]	Code	Android	MaMaDroid [24], Drebin [25]	API, CFG , Permission	Code Injection
HRAT [17]	Code	Windows	MaMaDroid [24], MalScan [26]	API, FCG	Block Size, Modify graph, Add Subgraph
Zhang *et al.* [27]	Binary	Windows	GCN, DGCNN	CFG, Opcode	Block Size, Opcode
Chen *et al.* [28]	Binary	IoT	RF	CFG, Opcode	Block Size, Opcode, Add Subgraph
Yuste *et al.* [29], Aryal *et al.* [30]	Binary	Windows	MalCov [6]	Byte-sequence	Code Cave Injections
Lucas *et al.* [19]	Binary	Windows	MalCov [6]	Byte-sequence	Binary Diversification Methods
Li *et al.* [20]	Binary	Windows	XGBoost	Opcode	NOPs Insertion
DeepMal [21]	Binary	Windows	General	Image	Payload Injection
Rosenberg [31]	Binary	Windows	EMBER [32]	Multiple PE Binary Features	Explainable-based Manipulations
Demetrio *et al.* [33]	Binary	Windows	MalConv [6]	Byte-sequence	Explainable-based Header Modifications
Our proposed	Binary	IoT	RF, SVM, XGBoost, DNN	CFG, Opcode	Explainable-based Block, Subgraph, Opcode Injections

lationships among various aspects of the binary. These features include Function Call Graphs (FCG) [10], [11], Control Flow Graphs (CFG) [8], Printable String (PSI) graphs [22] and opcode graphs [5]. Integrating signature-based and structure-based features improves the detailed representation of binary semantics [9].

Adversarial Attacks in Malware Detection

Adversarial attacks pose a significant threat to inadequately secured IoT devices used in smart homes, smart healthcare, smart entertainment, and other consumer applications [1], [15], [36]. A particularly insidious variant of these attacks targets malware detection systems, either by deceiving them into misclassifying malware as benign files [3] or by poisoning the data used to train the detectors [12]. While much of the existing research focuses on adversarial attacks against malware detection on Windows and Android [16], [17], [19], some of these attacks can also be adapted to detection systems deployed in consumer electronics within the IoT ecosystem.

Adversarial malware detection attacks can be categorized as code-level and binary-level attacks. In the case of code-level attacks, Abusnaina *et al.* [3] propose the GEA attack against detectors using CFG features by injecting benign subgraphs into malware files. In a subsequent work named SGEA, Abusnaina *et al.* [14] reduce the graph size needed to evade the target detector. Chen *et al.* [16] modify the source code of Android APK files and repackage them, altering their CFG structure. Zhao *et al.* [17] propose a code-level attack against FCG detectors by manipulating function calls.

When source code is absent, crafted binary snippets can be inserted directly into the binary [21]. Optimized via gradient methods, these payloads have been used to evade byte-based detectors like MalConv [6]. For stealthier attacks, payloads can be generated to replace existing binary instructions with semantic equivalents. For example, Lucas *et al.* [19] employ instruction substitution and binary diversification techniques to evade detection while preserving original malware functionality.

Some previous works have focused on inserting payloads targeting specific features on which the target detector is trained. For instance, Li *et al.* [20] replace malicious opcode sequences with semantic NOPs. Similarly, Chen *et al.* [28] insert executable opcode sequences to obfuscate CFG structure. Zhang *et al.* [27] attack GNN models using reinforcement learning, injecting semantic NOPs into specific nodes.

Closely related works include those by Yuste et al. [29] and Aryal *et al.* [30], which focus on the injection of code caves into PE binaries to mislead byte-based detectors such as MalConv [6]. Specifically, Yuste *et al.* [29] dynamically introduce empty blocks (caves) into the malware binary. They then apply a Genetic Algorithm to select suitable content to place in these code caves, thereby evading detection. In a similar vein, Aryal *et al.* [30] introduce intra-section code caves into Windows PE malware files. They use gradient-based approaches to generate adversarial samples, which are then injected into the created code caves to evade detection.

Adversarial Attacks via Explainable AI (XIA)

Explainable AI (XAI) algorithms, such as SHapley Additive exPlanations (SHAP) [37] and Local Interpretable Model-Agnostic Explanations (LIME) [38], have been leveraged for targeted adversarial attacks [31]. They are used to rank feature importance, and the attacks are orchestrated to manipulate the most influential features.

Demetrio *et al.* [33] used integrated gradients to evade a binary-based malware detector by targeting influential blocks. Li *et al.* [20] applied SHAP to an opcode-based detector, replacing the most malicious opcode sequence with semantic NOPs. Rosenberg *et al.* [31] employed diverse explainability approaches to evaluate feature importance in models, demonstrating that targeting the influential features is practical and efficient.

Table 2. Features and categories of features used by malware detectors

Structural	Opcode
`Nodes, Edges, Out_degree, In_degree, Density, Closeness_cent, Betweenness_cent, Connected_com, Diameter, Radius, Avg_block`	`Total_trans, Total_cal, Total_ctl, Avg_trans, Avg_cal, Avg_ctl, Avg_block_size`

Unlike existing studies, our methodology leverages the injection of payloads into the existing empty regions of the binary, combined with explainable AI techniques. This approach is tested on real-world IoT datasets, including widespread botnets like Mirai, frequently detected in consumer IoT devices.

SYSTEM MODEL

In this section, we detail the components (a), (b), (c), and (d) of Figure 1. We elaborate on how we utilize the feature importance analysis results to construct effective payloads to bypass the target detector.

Threat Model

Our attack scenario assumes the adversary has white-box and black-box access to the target detector. In the white-box setting, the adversary can access all detector details, including the model's architecture and parameters. The black-box setting, however, limits the adversary to the model's prediction result. If the adversary only has black-box access, they can execute the attack by training a substitute model and attempting to transfer the attack to the target detector.

The goal is to modify malware samples while preserving their original functionality until they are misclassified as benignware by the target detector. Each sample $x \in \mathcal{X}$ is associated with a label $y \in \mathcal{Y}$, where $\mathcal{Y} = \{0, 1\}$. Using reverse engineering, we transform each sample into an n-dimensional feature vector $z \in \mathcal{Z}$, which is then used to train the target detector.

Feature Importance Analysis

In our quest to develop a robust malware detector, we employ three categories of features to train three distinct detectors: Structure-based, opcode-based, and a combination of both (subsequently referred to as structure, opcode, and hybrid detectors). To enhance the applicability of our attack, the hybrid model is utilized as the target model for our explainability analysis. Table 2 presents the features used to train the detectors.

As shown in Figure 1 (a), after training the target detector, we use explainability analysis techniques to assess the feature importance. We utilize SHAP [37] to discern the correlation between features and model predictions. Additionally, we apply the LIME analysis method [38] for a detailed examination of the feature influence range of individual malicious samples.

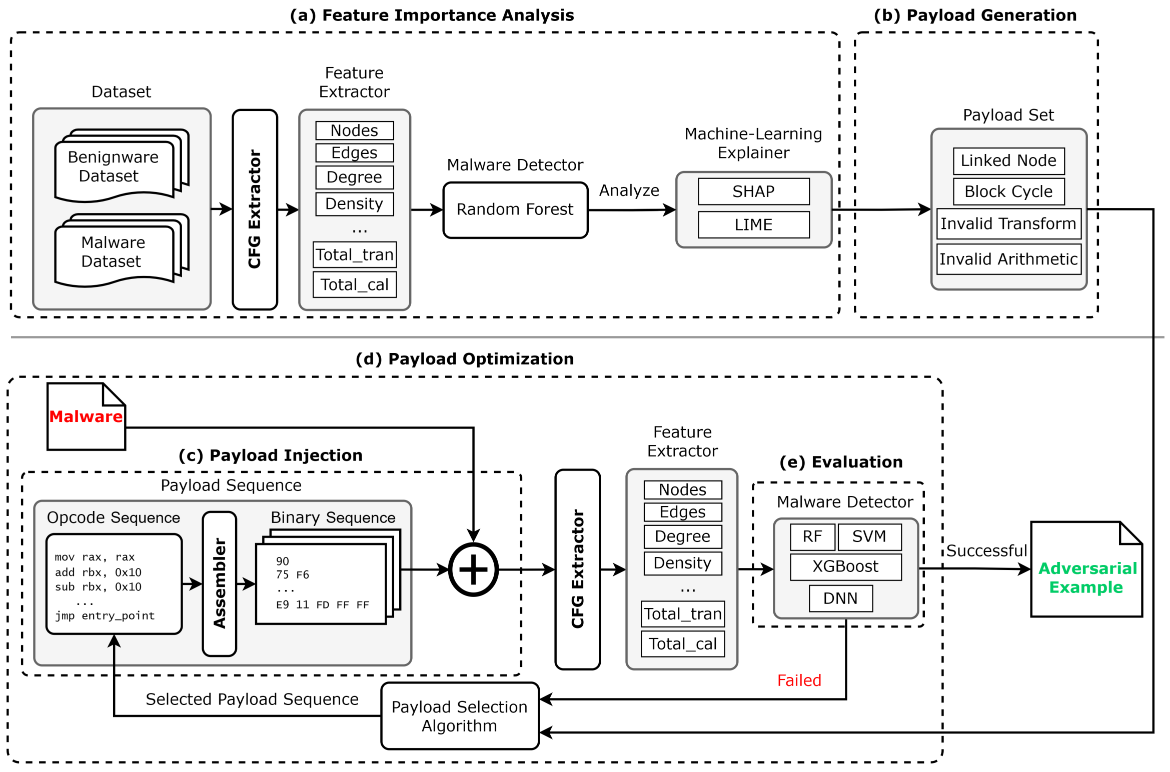
- **SHAP Analysis**

Figure 2 shows the SHAP value distribution of all features, allowing intuitive analysis of detector predictions. Each row represents the distribution of all samples in a specific feature, with earlier ranked features having more influence. The color intensity of each point, representing a test dataset sample, indicates the corresponding SHAP value. We selected the top 9 influential features for payload generation, including five structure-based (`Avg_block, Closeness_cent, Density, Out_Degree, Nodes`) and four opcode-based (`Avg_block_size, Avg_cal, Total_ctl, Total_trans`) features. We analyzed the influence of feature values on prediction results, considering the difference between malicious and benign samples. For example, high values of features such as `Avg_block, Avg_block_size, Avg_cal, Total_ctl, Nodes`, are positively correlated with the prediction of maliciousness. Therefore, our strategy is to decrease these values to mislead the detector. For features such as `Closeness_cent`, `Density`, `Out_degree`, and `Total_trans`, lower values are positively correlated with the prediction of maliciousness; thus, our strategy is to increase these values to evade detection.

- **LIME Analysis**

Following SHAP analysis, we employ the LIME explainability analysis [38] to probe the degree of influence that features have on malware predictions. This investigation reveals that certain ranges of feature values are associated with heightened predictions of maliciousness. For instance, if the range of values for `Avg_block_size` that contribute to a sample being classified as malicious is denoted as `Avg_block_size > 14.00`, it suggests that reducing the `Avg_block_size` value to below 14.00 could lead the detector to incorrectly classify the

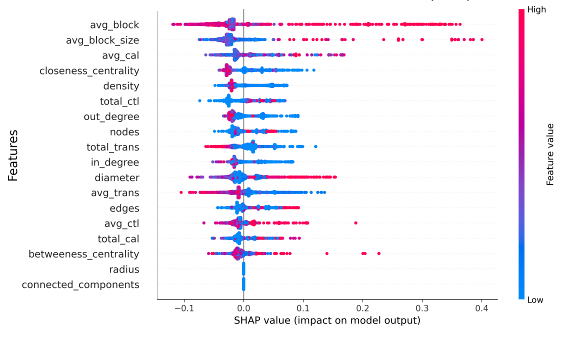
Figure 1. The Proposed Framework



```
mov rax, rax
add rbx, 0x10
sub rbx, 0x10
   ...
jmp entry_point
```

```
90
75 F6
...
E9 11 FD FF FF
```

Figure 2. The SHAP value distribution of testing dataset features when label = malicious (RF)



sample as benign.

Payload Generation

Finally, we construct our influential payloads based on the analysis results (i.e., part (b) in Figure 1). Our attack strategy comprises four payloads: two structure-based and two opcode-based.

Structure-based payloads primarily alter the flag register value via the compare instructions and control the execution path with the conditional instructions. This allows the injected payloads to be parsed as part of the CFG without affecting the file functionality. A jump instruction returns to the original program entry point. The two payloads aimed at altering the structure of the malware binaries are discussed below.

1) **Linked Node**
By injecting the opcode sequence into the binary file's executable segment, we form the CFG subgraph structure shown in Figure 3(a). We link the subgraph directly to the original CFG's entry point to perturb the structural and opcode features. Specifically, this modification reduces the values of influential features like `Avg_block_size`, `Avg_block`, and `Avg_cal`, thereby increasing the probability of a malware sample being classified as benign.

2) **Block Cycle**
Using different conditional instructions, we form a CFG subgraph structure as shown in Figure 3(b) and inject it at the entry point of the original CFG. Injecting numerous CFG blocks simultaneously saves bytes needed for the entire payload required to mislead the detector. Specifically, this injection increases the number of blocks in the CFG, thereby reducing the values of features such as `Avg_block_size`, `Avg_block` and `Avg_cal` while increasing the value of

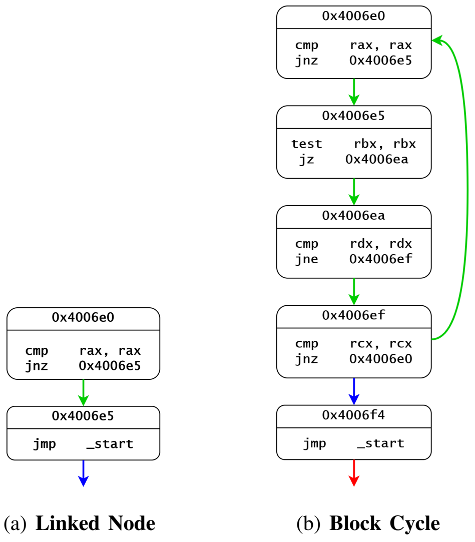


Figure 3. The structure-based payload

`density` thus reducing the malicious probability of a sample.

Opcode-based payloads, consisting of semantic NOPs, impact the opcode-based detector despite not affecting the CFG structure. We categorize general semantic NOPs into two types:

1) **Invalid Transform**
While preserving the original executable functionality, we inject Invalid Transform instructions such as mov rax, rax, push rax, and pop rax. This payload type influences features such as `Total_trans`, `Avg_trans`, and `Avg_block_size`. For instance, this perturbation increases the values of `Total_trans` and `Avg_trans` features, which, according to our analysis, will lead the detector to classify a malware sample as benign.

2) **Invalid Arithmetic**
Additionally, we insert invalid arithmetic instructions such as and rax, rax, sub rax, -0x10 and add rax, 0x10, which alter features like `Total_cal`, `Avg_cal` and `Avg_block_size`. For example, injecting invalid arithmetic instructions that form an additional block will not only increase the value of `Total_cal` but will also increase the number of blocks, which in turn will reduce `Avg_cal` and `Avg_block_size`, thereby increasing the probability of a malware sample being classified as benign.

Payload Injection

To insert the payload into the CFG structure without disrupting the file's functionality, we carefully select the injection site. We analyze the ELF file structure, identify unused space in the LOAD segment, typically at the end, and inject the payload sequences. We replace the program's original entry point with our payload, ensuring our instructions execute first, then redirect the payload's end back to the original entry point to maintain functionality. If the available space is insufficient, we modify the Program Header Table and Section Header Table to expand it.

Payload Injection Optimization Algorithm

Algorithm 1: IoT Structural Attack

```
Input  : Target Detector $\mathbb{D}$, original malware
         example $x$, payload set $P$, maximum
         stagnation $N$, constraints $t$
Output: Selected sequence $S$, adversarial
         example $\tilde{x}$
1  $\tilde{x} \leftarrow x$, $S \leftarrow \phi$, $step \leftarrow 1$, $\tilde{z} \leftarrow \mathcal{T}(\tilde{x})$
2  while $\mathbb{D}(\tilde{z}) > 0.5$ and $step \leq N$ do
3      $Probability_{min} \leftarrow \mathbb{D}(\tilde{z})$
4      for $p \in P$ do
5          $p_{seq} \leftarrow$
             $GeneratePayloadSeq(p_i, step)$
6          $S_{tmp} \leftarrow S \cup p_{seq}$
7          $\tilde{x}_{tmp} \leftarrow InjectPayload(\tilde{x}, S_{tmp})$
8          $\tilde{z}_{tmp} \leftarrow \mathcal{T}(\tilde{x}_{tmp})$
9          $r \leftarrow RandomizeFunction(t)$
10         if $\mathbb{D}(\tilde{z}_{tmp}) \leq Probability_{min} + r$
             then
11             $Probability_{min} \leftarrow \mathbb{D}(\tilde{z}_{tmp})$
12             $S' \leftarrow S_{tmp}$
13         end
14     end
15     $\tilde{x}_{tmp} \leftarrow InjectPayload(x, S')$
16     if $d(x, \tilde{x}_{tmp}) \leq \Delta$ then
17         $step \leftarrow$
             $DynamicAdjust(step, \mathbb{D}(\tilde{z}))$
18         $\tilde{x} \leftarrow InjectPayload(x, S')$
19         $S \leftarrow S'$
20         $\tilde{z} \leftarrow \mathcal{T}(\tilde{x})$
21     end
22     $t \leftarrow t - 1$
23 end
24 return $S, \tilde{x}$
```

Table 3. Detection Results. "Acc.", "Prec.", "Rec.", and "F1" stand for Accuracy, Precision, Recall, and F1-Score, respectively.

	RF				SVM				XGBoost				DNN			
	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
Structure	97.06	97.04	97.01	97.05	96.06	96.05	96.03	96.07	96.06	95.9	95.8	95.82	95.43	95.30	95.22	95.36
Opcode	98.84	98.42	98.37	98.85	97.83	97.01	96.82	97.78	98.72	98.02	97.96	98.62	95.17	94.86	94.75	95.11
Hybrid	98.97	98.42	98.06	98.88	98.47	98.11	97.98	98.63	99.11	99.01	98.96	99.04	98.35	98.12	98.01	98.33

The malware detector takes sample features z as input and outputs a confidence score. If the score exceeds 0.5, the sample is classified as malicious; otherwise, it is benign. We aim to alter a malicious sample $\tilde{x}$ to mislead the detector using its feature $\tilde{z}$, aiming for a confidence level below 0.5. The attack framework is depicted in Figure 1(d).

Our attack employs a greedy strategy specially tailored to our problem. In each iteration, we inject four payloads into the malicious samples sequentially and observe the model's predictions. We select the payload sequence that reduces confidence the most and initiate the next iteration. In the selection phase, we introduce a randomized parameter to allow the algorithm to accept suboptimal solutions and increase the likelihood of escaping local minima.

We monitor the model's confidence changes and note that the confidence decrease for most samples experiences periods of stagnation. Consequently, we introduce a function to dynamically adjust the number of steps, reducing the iteration count required to generate adversarial examples. Specifically, this function intensifies the rate of payload injection in a given iteration when the detector's confidence level stagnates.

This reduction in iterations decreases the time required for generating adversarial examples and the number of queries made to the model, effectively saving costs and making adversarial attacks more imperceptible. We repeat this process until the model classifies the target sample as benign or we reach our repetition limit. The algorithm used in our approach is shown in Algorithm 1.

EVALUATION

Dataset

To evaluate the proposed framework, we compiled a dataset of 14,320 IoT samples from various CPU architectures, including ARM, x86, x86-64, MIPS, PowerPC, and SPARC. These samples were verified by different antivirus vendors via VirusTotal [39]. The final classification, based on a majority vote from VirusTotal's report, yielded 8156 benign and 6164 malicious instances, with Mirai malware being prevalent. The dataset was split into 80% for training and 20% for testing.

Target Model and Experiment Setting

With the dataset prepared, we use the Angr [40] framework to extract control flow graphs (CFGs) from the binaries. We then employ NetworkX [41] to extract structural features from these CFGs. Additionally, we extract opcode features from the disassembled binaries. Specifically, still using the Angr framework, we extract the assembly instructions from the ELF binaries and categorize them into three types: "Transform Instructions," which include data movement instructions such as mov, push, and pop; "Arithmetic Instructions," which include instructions such as add, sub, and div; and "Control Instructions," which include control flow instructions such as jmp, je, and call. We then compute the total number of each type of instruction and the average number of each type per block to form our opcode features. These features are then combined with the structural features to create the feature set presented in Table 2. Next, we construct three detectors based on our feature categories: a structure-based detector, an opcode-based detector, and a hybrid detector that integrates both feature types. These models are tested with various machine learning algorithms, including Random Forest (RF), Support Vector Machine (SVM), eXtreme Gradient Boosting (XGBoost), and Deep Neural Networks (DNN). All detectors achieve an accuracy score of over 95%, with the hybrid detector surpassing 98% (see Table 3). Our preliminary analysis demonstrates that the hybrid detector is more robust than the other detectors; therefore, we selected it to generate our adversarial examples.

Analysis of Structural Attack

As depicted in Figure 1 (d), we execute a structural attack by inserting payloads into the malware binaries. Our results are compared with those of Chen *et al.* [28], who did not use explainability. We employ a similar approach, but ours is more influential as it incorporates explainability methods. It's worth noting that Chen *et al.* [28] utilized a Hill Climbing

Table 4. Comparing the performance of Chen *et al.* [28]'s Hill-Climbing (HC) and our Optimization Algorithm (OA) attack strategies under the same constraints(Δ).

Detector Feature	Δ	Method	Detector	Evasion Rate(%)	Append Size(%)	Iterations	Δ	Evasion Rate(%)	Append Size(%)	Iterations
Structure	$A = 40$	Chen *et al.* [28]	RF	97.03	10.60	45.79	$I = 50$	62.38	4.46	21.50
			SVM	92.08	8.17	40.43		66.34	5.53	25.61
			XGBoost	87.13	9.35	49.4		54.46	4.29	21.16
			DNN	91.09	14.9	73.92		38.61	8.06	37.16
		Our + HC	RF	98.02	6.69	28.49		87.13	4.90	21.24
			SVM	100.00	7.75	48.66		69.31	7.13	24.71
			XGBoost	99.01	7.92	47.4		66.34	7.47	21.72
			DNN	99.01	10.11	83.89		39.60	7.4	35.76
		Our + OA	RF	99.01	8.58	13.51		100.00	9.02	13.62
			SVM	100.00	8.09	13.11		100.00	8.09	13.11
			XGBoost	99.01	8.38	11.56		100.00	8.76	11.64
			DNN	99.01	9.69	16.69		100.00	10.18	16.78
Hybrid		Chen *et al.* [28]	RF	80.20	9.81	51.40		41.58	5.36	27.78
			SVM	100.00	12.08	63.65		40.59	10.1	40.00
			XGBoost	80.20	11.44	58.61		23.76	8.41	29.23
			DNN	78.22	13.02	76.85		23.76	7.16	23.12
		Our + HC	RF	97.03	10.40	50.24		71.29	5.20	27.49
			SVM	100.00	9.58	62.18		42.57	8.77	38.92
			XGBoost	98.02	10.73	92.38		24.75	7.46	26.55
			DNN	96.04	13.05	118.85		24.75	12.74	22.38
		Our + OA	RF	98.02	9.49	15.28		100.00	10.48	15.74
			SVM	100.00	9.71	21.95		100.00	9.71	21.95
			XGBoost	99.01	9.81	15.24		100.00	10.37	15.33
			DNN	96.04	12.39	23.30		98.02	13.36	23.48

optimization algorithm, while we opted for a greed-based optimization algorithm. We evaluate the models' robustness using 101 randomly selected malicious samples.

We introduce a threshold Δ to limit the attack cost in terms of append size and maximum number of iterations. We set Δ to $A = 40$ and $I = 50$, restricting the append size to 40% of the original file size and the maximum number of iterations to 50, respectively. Our method achieves a minimum evasion rate of 98.02% on the structural detector and above 96.04% on the hybrid detector, as detailed in Table 4.

The attack by Chen *et al.* [28] performs well on structure-based RF and hybrid SVM models, but the evasion rate decreases to around 90% for the other models and to 78% for the hybrid detector. When the iteration limit is 50, the evasion rate of Chen *et al.* [28] approach falls below 70% for both structure-based and hybrid detectors.

Using the Hill Climbing (HC) algorithm, our approach achieves evasion rates of up to 87% on the RF structure-based model and 71% on the hybrid detector. By substituting HC with our optimization algorithm, the evasion rate improves to over 98% under the same constraints.

We evaluated our attack under various constraints and mapped the correlation between append size, iteration count, and evasion rate (Figure 4). Our Optimization Algorithm (OA) outperforms the Hill-Climbing (HC) algorithm, achieving an evasion rate of 51.49% over HC's 47.52% under a 5% append size limit. At a 90% evasion rate, our method requires roughly 25% append size, less than Chen *et al.* [28]'s requirement of over 50%. Our OA is also more iteration-efficient, needing only 24 iterations to exceed a 90% evasion rate on hybrid XGBoost, versus the 250 iterations needed by HC.

Transferability of Adversarial Examples

We evaluated our attack's effectiveness by testing the transferability of the generated adversarial examples. Samples generated by one ML model successfully evaded other models trained on the same feature set (See Figure 6). The structure-based model was more susceptible to our attack, with over 50% transfer rate between similar models. The DNN algorithm proved to be the most robust compared to other ML models in all the detectors, as shown in Fig 6.

We further evaluated the transferability of the generated adversarial samples on a leading IoT malware detector [8] that uses different structural features. The effectiveness of the samples was confirmed, achieving a high evasion rate of up to 95% with a minimal attack cost. The comparison of evasion rates is presented in Table 5.

Comparison with Similar Studies

We compared the adversarial examples generated by our approach with those from the GEA [3] and SGEA [14] frameworks. Both approaches, like ours, inject benign graphs into the CFGs of malware files to deceive CFG-based IoT malware detectors. We computed the additional CFG nodes introduced by each method. The results show that GEA requires $1,075$ nodes for 100% evasion, while SGEA and our approach require only 6.8 and 26.83, respectively. Although our method outperforms GEA, SGEA remains superior in terms of node count efficiency.

Figure 4. Comparison of the relationship between Evasion Rate and Append Size

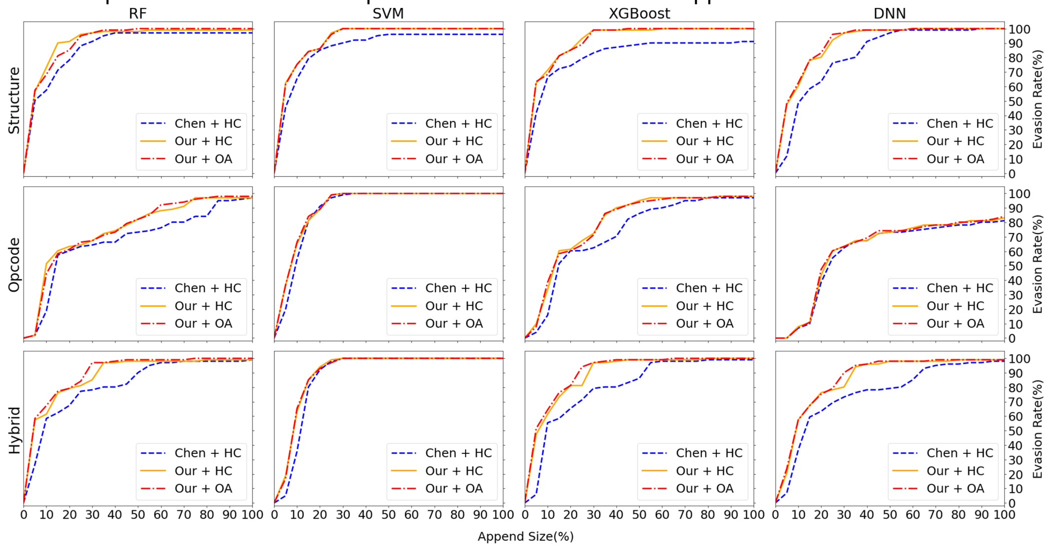


Figure 5. Comparison of the relationship between Evasion Rate and Iterations

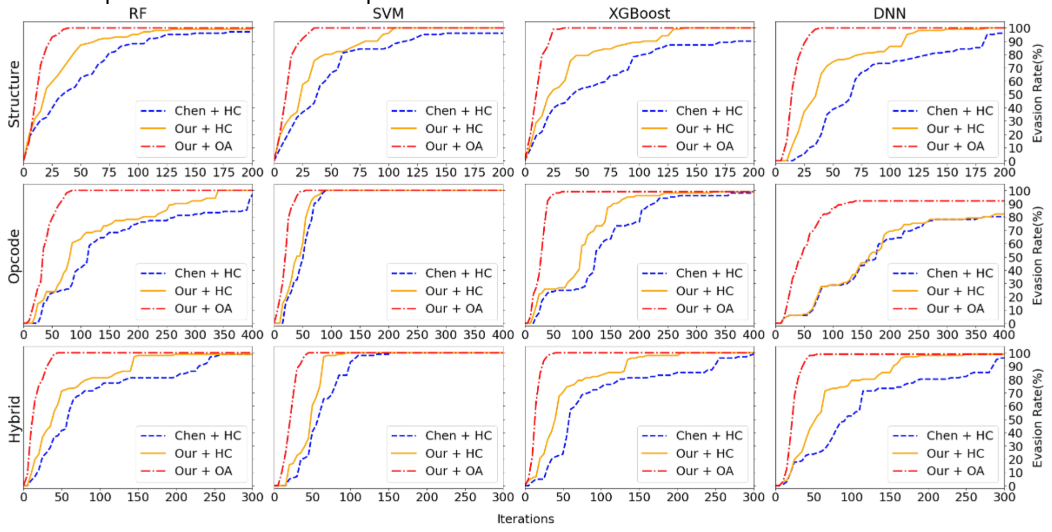


Figure 6. Transfer rate of adversarial examples on different model

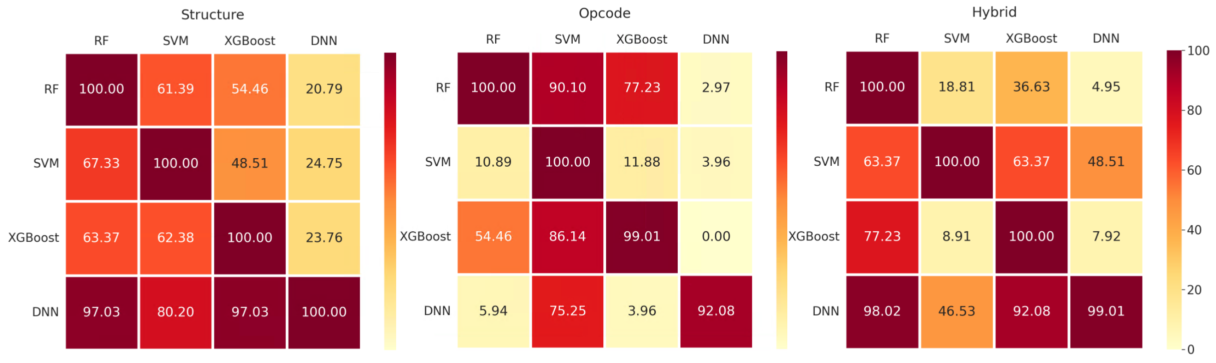


Table 5. The adversarial attack uses our payload set with the Optimization Algorithm (OA) on different structure-based detectors.

Method	Detector Feature	Detector	Evasion Rate(%)	Append Size(%)	Iterations
Our + OA	**Alasmary *et al.* [8]**	**RF**	95.15	**4.69**	**9.26**
	Hybrid Detector	**RF**	**100.00**	9.02	13.62

Discussion and Recommendations

The experimental results demonstrate the inherent vulnerabilities of AI-based malware detection systems to adversarial attacks. Subtle input perturbations can trick detectors into misclassifying malicious files as benign, posing a serious threat to IoT networks, where undetected malware can cause significant damage. Our attack, though limited to structure-based IoT malware detectors, generates adversarial samples that can fool detectors trained on different features [8].

To enhance the robustness of AI-based malware detectors and mitigate these vulnerabilities, we propose several practical recommendations. Firstly, creating adversarial-aware detection models through adversarial training [18], [29] can help anticipate and counter potential attacks. Secondly, we recommend using anomaly detection systems and input preprocessing to identify unusual input patterns, which may signal adversarial attacks, and to assist in removing these perturbations. Thirdly, we emphasize the importance of robustness evaluation and testing of these AI detection systems to enhance their resilience against attacks, including continuous improvement to adapt to evolving threats.

Our experiments reveal that some machine learning models, such as Deep Neural Networks (DNN), are more resilient than others, such as Random Forest (RF). Therefore, we propose the use of ensemble methods, combining multiple models and utilizing majority voting to enhance resilience. Furthermore, we advocate for using robust features in model training that are less susceptible to manipulation by adversarial attacks or require significant effort to alter. We also recommend implementing adaptive defensive mechanisms, such as online learning and continuous monitoring, to dynamically adapt to new types of attacks.

Lastly, we stress the need to incorporate human expertise in the process, as human experts can verify and analyze suspicious cases flagged by AI systems, providing an additional layer of security. While some solutions are cost-effective and suitable for lightweight IoT malware detection, others are resource-intensive and complex, making them less feasible for resource-constrained IoT devices.

CONCLUSION

We proposed a practical, imperceptible adversarial attack against structure-based malware detectors. Using ML explainability methods, we identified the most impactful features of the target detector and created four payload categories to alter these features. We applied a refined greedy algorithm to inject payloads into the empty spaces of malware binaries, successfully evading detection. Notably, the samples generated were transferable to a prominent graph-based IoT malware detector trained on different features. Our study highlights the vulnerabilities of ML-based IoT malware detectors, underscoring the need for continuous defense efforts.

One limitation of our approach is its inability to handle obfuscated malware. Obfuscation can alter the structural features used in training our target detector and, in some cases, hinder the extraction of control flow graphs, which are central to our method. In the future, we plan to integrate both dynamic and static analysis to develop a more robust detector capable of handling obfuscated malware.

REFERENCES

1. C. K. Wu, C.-T. Cheng, Y. Uwate, G. Chen, S. Mumtaz, and K. F. Tsang, “State-of-the-art and research opportunities for next-generation consumer electronics,” *IEEE Trans. Consum. Electron.*, vol. 69, no. 4, pp. 937–948, Dec. 2022.
2. T. Alladi, V. Chamola, B. Sikdar, and K.-K. R. Choo, “Consumer iot: Security vulnerability case studies and solutions,” *IEEE Consum. Electron. Mag.*, vol. 9, no. 2, pp. 17–25, Feb. 2020.
3. A. Abusnaina, A. Khormali, H. Alasmary, J. Park, A. Anwar, and A. Mohaisen, “Adversarial learning attacks on graph-based IoT malware detection systems,” in *Proc. IEEE ICDCS 2019*, Jul. 2019, pp. 1296–1305.
4. V. Sharma, B. Varghese, J. McAllister, and S. P. Mohanty, “Abusive adversaries in 5g and beyond iot,”

IEEE Consum. Electron. Mag., vol. 11, no. 4, pp. 11–20, 2021.

5. A. D. Raju, I. Y. Abualhaol, R. S. Giagone, Y. Zhou, and S. Huang, "A survey on cross-architectural IoT malware threat hunting," *IEEE Access*, vol. 9, pp. 91 686–91 709, Jun. 2021.
6. E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole EXE," in *Proc. AAAI 2018*, Jun. 2018.
7. L. Tan, K. Yu, F. Ming, X. Cheng, and G. Srivastava, "Secure and resilient artificial intelligence of things: a HoneyNet approach for threat detection and situational awareness," *IEEE Consum. Electron. Mag.*, vol. 11, no. 3, pp. 69–78, May 2021.
8. H. Alasmary, A. Khormali, A. Anwar, J. Park, J. Choi, A. Abusnaina, A. Awad, D. Nyang, and A. Mohaisen, "Analyzing and detecting emerging Internet of Things malware: A graph-based approach," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8977–8988, Oct. 2019.
9. B. Wu, Y. Xu, and F. Zou, "Malware classification by learning semantic and structural features of control flow graphs," in *Proc. IEEE TrustCom 2021*, Oct. 2021, pp. 540–547.
10. C.-Y. Wu, T. Ban, S.-M. Cheng, B. Sun, and T. Takahashi, "IoT malware detection using function-call-graph embedding," in *Proc. IEEE PST 2021*, Dec. 2021, pp. 1–9.
11. C. Li, G. Shen, and W. Sun, "Cross-architecture Internet-of-Things malware detection based on graph neural network," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2021, pp. 1–7.
12. R. Taheri, M. Shojafar, M. Alazab, and R. Tafazolli, "FED-IIoT: A robust federated malware detection architecture in industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 17, pp. 8442–8452, Dec 2020.
13. D. Puthal and S. P. Mohanty, "Cybersecurity issues in AI," *IEEE Consum. Electron. Mag.*, vol. 10, no. 4, pp. 33–35, June 2021.
14. A. Abusnaina, H. Alasmary, M. Abuhamad, S. Salem, D. Nyang, and A. Mohaisen, "Subgraph-based adversarial examples against graph-based IoT malware detection systems," in *Proc. Computational Data and Social Networks 2019*, Nov. 2019, pp. 268–281.
15. C. Song, H.-P. Cheng, H. Yang, S. Li, C. Wu, Q. Wu, and H. Li, "Adversarial attack: A new threat to smart devices and how to defend it," *IEEE Consum. Electron. Mag.*, vol. 9, no. 4, pp. 49–55, Jun. 2020.
16. X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android HIV: A study of repackaging malware for evading machine-learning detection," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 987–1001, 2020.
17. K. Zhao, H. Zhou, Y. Zhu, X. Zhan, K. Zhou, J. Li, L. Yu, W. Yuan, and X. Luo, "Structural attack against graph based android malware detection," in *Proc. ACM Asia CCS 2021*, Nov. 2021, p. 3218–3235.
18. R. Taheri, R. Javidan, M. Shojafar, P. Vinod, and M. Conti, "Can machine learning model with static features be fooled: An adversarial machine learning approach," *Clust. Comput.*, vol. 23, pp. 3233–3253, Mar 2020.
19. K. Lucas, M. Sharif, L. Bauer, M. K. Reiter, and S. Shintre, "Malware makeover: Breaking ML-based static analysis by modifying executable bytes," in *Proc. ACM Asia CCS 2021*, May 2021, pp. 744–758.
20. X. Li, K. Qiu, C. Qian, and G. Zhao, "An adversarial machine learning method based on opcode n-grams feature in malware detection," in *Proc. IEEE DSC 2020*, Jul. 2020, pp. 380–387.
21. C. Yang, J. Xu, S. Liang, Y. Wu, Y. Wen, B. Zhang, and D. Meng, "DeepMal: maliciousness-preserving adversarial instruction learning against static malware detection," *Cybersecurity*, vol. 4, May 2021.
22. H.-T. Nguyen, Q.-D. Ngo4, and V.-H. Le, "A novel graph-based approach for IoT botnet detection," *International Journal of Information Security*, vol. 19, no. 5, pp. 567–577, Oct. 2020.
23. Y. Hua, Y. Du, and D. He, "Classifying packed malware represented as control flow graphs using deep graph convolutional neural network," in *Int. Conf. Comput. Eng. Appl. (ICCEA)*, Mar 2020, pp. 254–258.
24. E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," *arX preprint arXiv:1612.04433*, 2016.
25. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, 2014, pp. 23–26.
26. Y. Wu, X. Li, D. Zou, W. Yang, X. Zhang, and H. Jin, "Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis," in *Proc. IEEE/ACM ASE. 2019*, 2019.
27. L. Zhang, P. Liu, Y.-H. Choi, and P. Chen, "Semantics-preserving reinforcement learning attack against graph neural networks for malware detection," *IEEE Trans. Dependable Secure Comput.*, Mar. 2022.
28. T.-Y. Chen, "Structural attack against graph-based IoT

malware detection at assembly level," Master, NTUST, Taipei, Taiwan, Jan. 2022.

29. J. Yuste, E. G. Pardo, and J. Tapiador, "Optimization of code caves in malware binaries to evade machine learning detectors," *Computers & Security*, vol. 116, p. 102643, Feb. 2022.
30. K. Aryal, M. Gupta, M. Abdelsalam, and M. Saleh, "Intra-section code cave injection for adversarial evasion attacks on windows PE malware file," *arXiv preprint arXiv:2403.06428*, Mar. 2024.
31. I. Rosenberg, S. Meir, J. Berrebi, I. Gordon, G. Sicard, and E. O. David, "Generating end-to-end adversarial examples for malware classifiers using explainability," in *Proc. IEEE IJCNN 2020*, Jul. 2020, pp. 1–10.
32. H. S. Anderson and P. Roth, "EMBER: an open dataset for training static PE malware machine learning models," *arXiv preprint arXiv:1804.04637*, Apr. 2018.
33. L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Explaining vulnerabilities of deep learning to adversarial malware binaries," *arXiv preprint arXiv:1901.03583*, Jan. 2019.
34. F. Ding, G. Zhu, M. Alazab, X. Li, and K. Yu, "Deep-learning-empowered digital forensics for edge consumer electronics in 5G HetNets," *IEEE Consum. Electron. Mag.*, vol. 11, no. 2, pp. 42–50, Feb. 2022.
35. Y.-T. Lee, T. Ban, T.-L. Wan, S.-M. Cheng, R. Isawa, T. Takahashi, and D. Inoue, "Cross platform IoT-malware family classification based on printable strings," in *Proc. IEEE TrustCom 2020*, Dec. 2020, pp. 775–784.
36. V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "Roadmap for cybersecurity in autonomous vehicles," *IEEE Consum. Electron. Mag.*, vol. 11, no. 6, pp. 13–23, Feb. 2022.
37. S. M.Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. NeurIPS 2017*, vol. 30, Dec. 2017, pp. 4768–4777.
38. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should I trust you?" Explaining the predictions of any classifier," in *Proc. ACM SIGKDD 2016*, Aug. 2016, pp. 1135–1144.
39. Developers, "VirusTotal," [online]. Available https://www.virustotal.com, 2024.
40. Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, and G. Vigna, "SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis," in *Proc. IEEE S&P 2016*, May 2016, pp. 138–157.
41. A. Hagberg, P. Swart, and D. Chult, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. SciPy 2008*, Aug. 2008, p. 11–15.

Chi-Hsin Yang received the M.Sc. degree in computer science and information engineering from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 2022. Her research interest is AI security.

Maina Bernard Mwangi received the M.Sc. degree in computer science and information engineering from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 2021 and is currently pursuing the Ph.D. degree under the same department in the same university. His research interests are IoT and AI security.

Shin-Ming Cheng earned his B.S. and Ph.D. in computer science and information engineering from National Taiwan University in 2000 and 2007. He served as a post-doctoral research fellow at the same university until 2012, when he joined the faculty of the Department of Computer Science and Information Engineering at National Taiwan University of Science and Technology, where he is now a professor. In 2017, he became a Joint Appointment Associate Research Fellow at the Research Center for Information Technology Innovation, Academia Sinica. His research focuses on secure mechanism design, cybersecurity platform development in 4G/5G and IoT networks, and machine learning robustness. He has received several awards, including the K. T. Li Young Researcher Award in 2014, the IEEE PIMRC 2013 Best Paper Award, the IEEE Trustcom 2020 Best Paper Award, and the CISC 2020 & 2021 Best Paper Award.

Hahn-Ming Lee Hahn-Ming Lee is a professor in the Department of Computer Science and Information Engineering at the National Taiwan University of Science and Technology, Taiwan. Lee received a PhD in computer science and information engineering from the National Taiwan University.