Make your IoT environments robust against adversarial machine learning malware threats: a code-cave approach

Hamed Haddadpajouh
University of Guelph
hhaddadp@uoguelph.ca

Ali Dehghantanha
University of Guelph
adehghan@uoguelph.ca

Abstract—**As the integration of Internet of Things devices continues to increase, the security challenges associated with autonomous, self-executing Internet of Things devices become increasingly critical. This research addresses the vulnerability of deep learning-based malware threat-hunting models, particularly in the context of Industrial Internet of Things environments. The study introduces an innovative adversarial machine learning attack model tailored for generating adversarial payloads at the bytecode level of executable files.**

Our investigation focuses on the Malconv malware threat hunting model, employing the Fast Gradient Sign methodology as the attack model to craft adversarial instances. The proposed methodology is systematically evaluated using a comprehensive dataset sourced from instances of cloud-edge Internet of Things malware. The empirical findings reveal a significant reduction in the accuracy of the malware threat-hunting model, plummeting from an initial 99% to 82%. Moreover, our proposed approach sheds light on the effectiveness of adversarial attacks leveraging code repositories, showcasing their ability to evade AI-powered malware threat-hunting mechanisms.

This work not only offers a practical solution for bolstering deep learning-based malware threat-hunting models in Internet of Things environments but also underscores the pivotal role of code repositories as a potential attack vector. The outcomes of this investigation emphasize the imperative need to recognize code repositories as a distinct attack surface within the landscape of malware threat-hunting models deployed in the Internet of Things environments.

I. Introduction

The proliferation of IoT devices introduces significant security challenges due to the lack of robust defensive mechanisms at the application level, creating vulnerabilities that adversaries exploit by crafting malicious payloads [1]. In response, ML-based threat hunting models have been integrated to counteract these threats [2], utilizing properties of both malicious and benign files to detect potential threats. This approach, while promising, requires further scrutiny to assess its effectiveness in the dynamic IoT security landscape.

In recent years, the pervasiveness of malware threats has emerged as a formidable challenge across diverse IT-based infrastructures, as highlighted by [3]. Concurrently, the proliferation of IoT applications across various business domains has exposed IoT-based infrastructures to a substantial onslaught of malware threats.

The architectural framework of IoT infrastructures typically encompasses a tripartite structure comprising the application, network, and edge layers (perceptional layer) i.e. [4]. Malicious actors, aware of this structural sketch, tend to concentrate their efforts on crafting malicious payloads tailored for the edge and application layers. The edge layer, in particular, becomes a great entry point for attacks due to the accessibility of its nodes within end-user environments. A poignant exemplification of this vulnerability is observed in Mirai attacks [5], wherein malware of the same nomenclature orchestrates substantial disruptions in the IoT environment. This underscores the critical nature of addressing malware threats in the context of IoT infrastructures, especially within the vulnerable domains of the edge layer.

In recent years, machine learning ML-based solutions have emerged as the most promising approaches against malware threats, offering benefits in malware threat prevention, hunting, and attribution [6], [7]. Deep learning (DL)-based approaches have also shown significant performance in dealing with malware threats in both IT [8] and IoT environments [7]. While the use of end-to-end DL makes it easy to create new models for a variety of file types by exploiting the vast number of labeled samples available to organizations, it also opens up the possibility of attacking these models using adversarial evasion techniques popularized in the image classification space.

The vulnerability of ML-based malware threat-hunting engines to adversarial attacks has become a pressing issue, as evidenced by recent practical efforts to attack ML-based malware threat-hunting models using AEs [9]. Adversarial ML techniques, commonly categorized into white-box and black-box models, have gained prominence in this context. These models exploit vulnerabilities in the decision boundaries of ML-based threat-hunting models, requiring precise adjustments to preserve the functionality of the generated examples. While most recent works exploit the decision boundaries of ML-based models, these boundaries cannot be deliberately

Workshop on Security and Privacy in Standardized IoT (SDIoTSec) 2024
26 February 2024, San Diego, CA, USA
ISBN 979-8-9894372-6-9
https://dx.doi.org/10.14722/sdiotsec.2024.23066
www.ndss-symposium.org

altered and require a precise effort to preserve the functionality of the generated example. For instance, [10] proposed a technique that modifies byte features and metadata of the executable file to attack a gradient-boosted decision trees threat hunting model. Similarly, [11] developed an attack model against Malconv [8], a well-known malware threat hunting model, by editing padding bytes of the executable based on a gradient-driven approach. These efforts suggest that most ML-based threat-hunting models are vulnerable to adversarial gradient-based attacks.

Adversarial attacks targeting malware threat-hunting models commonly entail introducing random perturbations to different attributes of executable files, including header files, demonstrating notable success rates. Nonetheless, a standardized methodology for delineating the attack surface of executable files, particularly within the context of IoT environments, is currently lacking. In this paper, we proposed an innovative approach, an attack surface analyzer, tailored for IoT malware executable files. This analyzer leverages the inherent code caves within the bytecode of these files, strategically exploiting these concealed spaces to circumvent conventional malware threat-hunting models. Our study's focus on Malconv and Linux environments was chosen due to Malconv's significance in IoT malware detection and the prevalence of Linux in IoT devices. We acknowledge the importance of generalizing our approach and are committed to exploring its applicability to other malware detectors and operating systems in future work.

The motivation for minimizing perturbations in the generation of adversarial examples is twofold. Firstly, it is essential to preserve the original functionality of the malware, ensuring that the adversarial modifications do not impair its intended malicious activities. Secondly, minimizing perturbations helps maintain the stealthiness of the malware, reducing the likelihood of detection by sophisticated malware detectors. This delicate balance is critical for the success of adversarial attacks in real-world scenarios, particularly in IoT environments where the integrity and undetectability of malware are paramount.

Our contributions in evaluating the robustness of IoT malware threat-hunting models are as follows:

- We propose a byte-level IoT executable file attack surface finder based on code-caves.
- We generate adversarial examples of IoT cloud-edge devices based on gradient descent weak points and the defined attack surfaces.
- We evaluate the robustness of the MalConv and multi-kernel malware threat-hunting models using the generated samples.

The remainder of this paper is structured as follows: Section II presents a review of recent work on adversarial attacks on deep malware threat hunting systems. Section III provides an overview of our proposed model for evading malware threat-hunting mechanisms and explains our methodology for addressing this challenge. In Section IV, we present our results and compare them with other works. Finally, Section V summarizes our work and discusses future extensions.

II. Related Work

Since the main aim of this work focuses on presenting a novel technique for generative adversarial malware examples for a could-edge layer of IoT environments, we review related prior contributions on this topic.

The adversarial ML attack was first introduced in [12] for image-based classifiers. This concept is then expanded in [13]. This concept presumes f is the target model (classifier) that the adversary tries to attack. For better understanding, let's assume that function $f(.)$ takes input and assigns a label to it. Therefore, An AE x' targeting f via perturbing an original input of x with δ so that $f(x) \neq f(x')$.

$$x' = x + \delta \tag{1}$$

$$f(x) \neq f(x') \tag{2}$$

In this section, we survey prior contributions related to our primary focus on introducing a novel technique for generating adversarial examples tailored for the cloud-edge layer of IoT environments.

The inception of adversarial ML attacks traces back to Goodfellow et al.'s seminal work on generative adversarial networks [12]. This concept was further elucidated in [13], where f represents the target model (classifier) under attack. In the context of our work, we assume a function $f(\cdot)$ that assigns a label to an input. An adversarial example (x') is generated by perturbing an original input (x) with δ such that $f(x) \neq f(x')$.

Adversarial attacks fall into two primary categories: white-box and black-box attacks [14]. White-box attacks assume complete knowledge of the target model, including weights, parameters, and trained data. In contrast, black-box attacks only have access to the input and output of the target model.

While most adversarial attacks are conducted on image datasets like MINIST, ImageNet, and CIFAR10 [15], approaches in the malware threat-hunting domain often focus on image-based models. However, translating malware properties into images introduces conversion overhead, rendering it impractical for IoT environments.

Adversarial ML attacks against malware threat-hunting models can preserve binaries' functionalities and fall into several categories:

- Modifying the import table by adding function names: Adding function names to mislead the model into classifying malware as legitimate [16].
- Altering binary section names: Renaming sections to evade model detection [17].
- Removing or changing signer names: Evading detection by modifying signer information [18].
- Creating new, unused section names: Generating unused section names to evade detection [19].
- Changing debugging data: Altering debugging data to confuse the model and evade detection [20].
- Inserting/copying bytes to unused space: Modifying unused space in sections to evade detection [21].

- Manipulating binary header information: Altering binary header information to evade model detection [22].
- Obfuscating binaries by wrapping/unpacking techniques: Using wrapping or unpacking techniques to obfuscate binaries and evade detection [23].

While traditional adversarial techniques often rely on manual interventions, advancements in machine learning, such as Reinforcement Learning (RL) and Generative Adversarial Networks (GANs), pave the way for automated adversarial example generation that maintains binary functionality. This shift towards automation, evidenced by works like [24] and [25], and models like [24] utilizing black-box approaches, marks a significant evolution from earlier ML and DNN methods that manually adjusted features like signatures and API calls, towards leveraging deep learning for more sophisticated, automated adversarial attacks.

A. Code-Caves: A Strategic Evasion Technique

Code-caves, unused spaces in executable files, provide a stealthy method for embedding adversarial payloads without affecting binary functionality. Particularly useful in IoT environments, these concealed segments allow for the subtle modification of binaries, facilitating evasion from ML-based malware detection systems.

1) Code-Caves: Evasion Technique: Utilizing code-caves as an evasion tactic, our approach injects malicious code into these overlooked segments, thereby bypassing AI-driven security measures without compromising the executable's integrity. This strategy not only challenges traditional malware detection models but also suggests avenues for future research to enhance the transferability and effectiveness of adversarial example generation across diverse detection systems and computing environments.

III. Proposed Model and Methodology

In this section, we present a comprehensive overview of our innovative model, specifically crafted to effectively evade ML-based IoT malware threat hunting models, with the well-known Malconv serving as a representative example. We aim to provide a robust defense against evolving cyber threats targeting IoT environments.

Our exploration begins with a detailed examination of the intricate structure of IoT executable files, aiming to identify latent vulnerabilities that can be exploited by adversaries. We particularly focus on the ELF (Executable and Linkable Format) files commonly utilized in cloud-edge devices within IoT ecosystems.

As depicted in Figure 1, our proposed approach takes an executable sample and dissects it into three distinct byte-code layers inherent to ELF files: the `.text`, `.data`, and `.rodata` sections. Each of these segments plays a pivotal role in the execution of the binary, housing essential instructions, necessary data, and read-only data, respectively.

The foundation of our model lies in the strategic initiation of the code-cave vulnerability. Code caves, representing unused or spare spaces within the binary, are prime targets for exploitation. These vacant areas provide an opportune entry point for injecting adversarial payloads, allowing us to navigate through the intricacies of malware detection models. The strategic injection of these payloads aims to deceive and bypass the targeted malware threat-hunting models effectively.

In the subsequent sections, we delve deeper into the mechanisms of our proposed model, outlining the steps involved in code-cave exploitation, adversarial payload generation, and the overall strategy to preserve the functionality of the executable while evading detection. This multifaceted approach encapsulates our efforts to fortify IoT environments against the evolving landscape of sophisticated cyber threats. Through the lens of our innovative model, we provide a strategic and effective countermeasure to bolster the security posture of cloud-edge devices within the IoT ecosystem.

To substantiate our approach, we examined an IoT dataset [7], revealing Code-Caves in benign samples within the `.data` section, as depicted in Figure 2. The choice to focus perturbations within the `.data` section was driven by its unique characteristics that typically allow for modifications without disrupting the executable's functionality. This section often contains initialized global and static variables, making it an ideal target for embedding adversarial payloads while preserving the malware's operational integrity. However, we acknowledge the potential limitations of this focused approach and the importance of diversifying attack strategies to enhance evasion capabilities.

A. Adversarial Example Generation

Preserving functionality while generating malicious examples is a challenge in adversarial machine learning. We minimize perturbations (δ), Eq. 3 and 4, to maintain functionality while maximizing the evasion rate, bypassing the ML engines. The strategy of generating random adversarial payloads is predicated on the notion of unpredictability and broad exploration of the model's decision boundaries. This approach is designed to probe the malware detection model in a manner that is less predictable than targeted modifications, potentially uncovering vulnerabilities that are not apparent through deterministic methods.

$$x' = x + \delta \tag{3}$$

$$\max(\text{loss}(M(x'))) \text{ w.r.t. } x' = \lim_{\delta \to 0}(x + \delta) \tag{4}$$

B. Executable Structure and Payload Insertion

Code caves represent stealthy enclaves within executables, often overlooked by traditional malware detection techniques, that provide a clandestine avenue for embedding adversarial payloads. These unused spaces are particularly prevalent in the `.data` section, are strategically exploited to maintain the malware's operational stealth without altering its external behavior. This approach not only leverages the inherent biases of detection models, which typically focus on more active sections like `.text`, but also ensures that any modifications

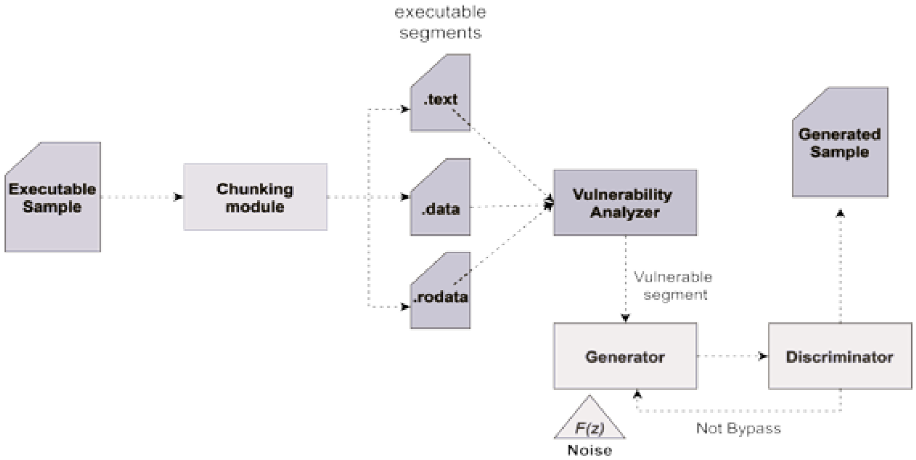
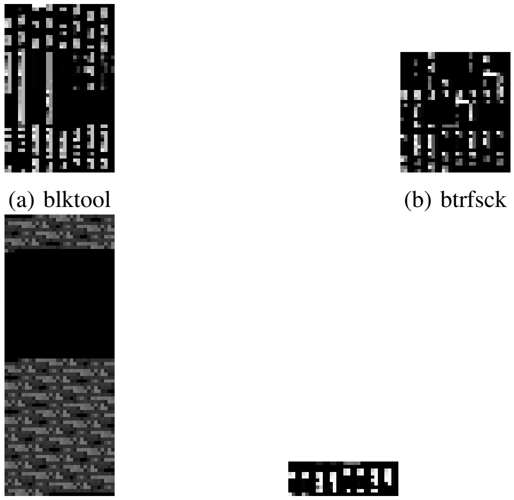


Fig. 1: The proposed methodology for generating Adversarial Examples (AE) from cloud-edge gateway samples.



(a) blktool

(b) btrfsck

(c) auto-apt-pkgcdb

(d) btrfs-convert

Fig. 2: Plots of existing Code-Caves in IoT executable files within the *.data* section.

subtly distort the malware's signature to evade detection while preserving functionality. However, the traditional view is that modifying the `.text` section is impractical—primarily due to concerns over functionality preservation and increased detectability—and has been challenged by recent advancements in obfuscation techniques. These emerging methods suggest new possibilities for safe alterations within the `.text` section that maintains the malware's intent and evades detection, underscoring the necessity for adaptive strategies that respond to technological progress in the cybersecurity domain. To pinpoint the optimal attack space, we assess spare spaces within each section using Algorithm 1.

Algorithm 1 Assessing the Attack Surface

```
Function CodeCave($x$, $B_{size} := 256$)
    $ByteSegments \leftarrow [3]$ ;   // .text, .rodata, .data segments
    $text, rodata, data \leftarrow$ chunk($x$) ;   // extracting byte segments
    $ByteSegments \leftarrow text, rodata, data$
    for $i \leftarrow 0$ to $ByteSegments.length()$ do
        $vulSpace \leftarrow$ Codecave.find($ByteSegments[i]$)
        if $vulSpace.size > B_{size}$ then
            $CaveInfo \leftarrow vulSpace$.info ;   // information about code caves
        end
    end
    return $Caveinfo$ ;  // returning sparse space starting and ending address
```

C. Target Model: Malconv

Our experiments employ the Malconv model [8] as the target. Malconv, a convolutional neural network, specializes in detecting malicious executable files by analyzing raw bytecodes.

D. Generating Malicious Examples

Our model includes three key sections, as depicted in Figure 1. The initial step involves attacking the targeted model Eq. (5, 6). We preprocess data on the bytecode level, assessing Code-Cave space to exploit for generating malicious examples. Algorithm 2 demonstrates the proposed attack model pseudocode, aiming to generate random adversarial payloads (8) while preserving functionality. The iterative procedure continues until the evasion threshold is satisfied.

$$max\ C(M, x', y) \tag{5}$$

$$subject\ to\ x' = x + \delta x \tag{6}$$

$$|x|_d \leq \epsilon * |x| \tag{7}$$

$$\delta x = \epsilon * sign\left(\nabla x \; C(M, x', y)\right) \tag{8}$$

Equation 7 indicates that any dimensions allow being altered, although the number of dimensions should not differ from original samples [26]. Moreover, the goal of maximizing the error in 5 is to prevent misclassification of the model based on the computed δx. This value also reaches Eq. 8. Where the $sign\left(\nabla x \; C(M, x', y)\right)$ is the direction of minimized cost function corresponds to δx value from Eq. 8. In the end, the cost function of adversarial malicious payloads is $C(M, x', y)$ where $y \neq y$.

After getting the desired information for each segment of the benign sample, we are going to evaluate the code-cave space to exploit them for generating malicious examples. Algorithm 2 demonstrates the proposed attack model pseudo code. In proposed attack model takes three major inputs, the original sample batch size, associated code-caves information, and the target model. The attack model tries to generate a random adversarial payload based on Eq. 8 to find appropriate δ. To preserve the functionality of AEs, unlike image-based models, we use code caves and inject a certain payload size into the original sample then pass it to the model again for final classification. This iterative procedure continues until the evasion threshold $T > 0.5$ is satisfied.

Within our proposed model, the principle of minimizing perturbations is applied rigorously to ensure that adversarial payloads not only bypass malware detection models effectively but also retain their malicious functionality unaltered. This approach underscores our methodology's emphasis on creating stealthy, functional adversarial malware, highlighting the sophistication and practical applicability of our attack strategy in evading ML-based malware threat hunters.

Algorithm 2 IoT adversarial payload generator

```
function ($x$, $Codecsave$, $model$)
while $!evade$ do
    $payload := generator(x, p = 1)$ w.r.t Eq. 8
    $x' := inject(payload, x, Codecave)$
    probability = $model.predict(x')$ if $probability > T$
  then
        label := benign
        evade := True
  end
end
return $x'$
end function
```

IV. Experimental Results

To assess the efficacy of our proposed approach in evading deep-net-based malware threat-hunting models, we conducted experiments across various test beds. Before delving into the details of these experiments, we establish key evaluation metrics. The selection of appropriate metrics is crucial to comprehensively evaluate the performance and robustness of a model against adversarial attacks.

A. Rationale for Minimizing Modifications in Executable Binaries

Minimal modifications in executable binaries strike a critical balance between maintaining stealth and preserving functionality. This approach is driven by four key considerations:

1) **Detection Model Sensitivity:** Malware detection models, particularly ML-based ones, are finely tuned to recognize patterns in binary files. Even minor anomalies can significantly heighten detection risk, making subtle changes preferable for evading these models without altering the binary's intended function.
2) **Anomaly Detection Thresholds:** Security systems' anomaly detectors are less likely to flag files with minimal alterations as suspicious, enhancing the odds of evasion.
3) **Forensic Traceability:** Smaller modifications complicate forensic analysis, aiding in obfuscating the origin or method of an attack.
4) **Computational Efficiency:** The process of generating and testing adversarial examples is more resource-efficient with fewer modifications, a crucial factor for large-scale or resource-constrained operations.

Opting for minimalistic changes—altering as little as a single byte—can therefore be a more effective strategy for bypassing sophisticated detection mechanisms, emphasizing the need for precision in the creation of adversarial examples.

B. Evaluation Metrics

In this section, we introduce two key metrics employed in our study: Mean Square Error (MSE) and Maximum Mean Discrepancy (MMD).

1) MSE: We utilized MSE as the loss function for our target model to compute the gradient of the model output, as shown in Eq. 9. Here, n represents the number of data points, and $e_t^2 = (y_i - y_i')^2$, where y_i is the observed value and y_i' is the predicted value.

$$MSE = \frac{1}{n}\sum_{t=1}^{n} e_t^2 \tag{9}$$

In the context of generated malware bytecode and tracking training loss using MSE, this metric plays a pivotal role in assessing the effectiveness of the model in replicating the features of legitimate and adversarial samples.

When dealing with malware bytecode, the model aims to understand the patterns and characteristics inherent in both benign and malicious executable files. The observed values (y_i) represent the actual features extracted from the malware bytecode, while the predicted values (y_i') are the model's estimations based on its current set of parameters.

The equation for MSE, as given in Eq. 9, quantifies the average squared difference between the observed and predicted values across all instances in the training dataset. In the context of generated malware bytecode, this means that MSE measures how closely the model's predictions align with the actual features of both benign and adversarial examples.

$$e_t^2 = (\text{observed feature}_t - \text{predicted feature}_t)^2$$

Here, the subscript t denotes individual features within the bytecode. By minimizing the MSE loss during training, the model is essentially learning to generate malware bytecode that closely resembles the characteristics of real-world examples, both benign and adversarial.

2) MMD for Adversarial Sample Generation: We employed MMD as a statistical technique to quantify, Eq. 10, the similarity between the adversarial samples generated by our proposed model and non-adversarial (legitimate) examples. MMD assessed the density distribution of the top 20 feature components in adversarial and non-adversarial executable files, obtained through Principal Component Analysis (PCA), –See Figure 3.

$$\begin{aligned} \text{MMD}(P_{\text{adv}}, Q_{\text{non-adv}}) = & \frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{j=1}^{m} k(x_i, x_j) \\ & - \frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k(x_i, y_j) \\ & + \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1}^{n} k(y_i, y_j) \end{aligned} \tag{10}$$

C. Experiments and Test Beds

The primary experiment aimed to delineate the decision boundary of the Malconv model by examining different segments of each dataset sample. Table I presents the evaluation results for each malware segment within the IoT dataset, revealing the efficacy of various file sections in generating adversarial examples.

Section Name	Recall
.text	0.98
.data	0.48
.rodata	0.99

TABLE I: Decision boundary of IoT dataset malware for each section.

To execute our experiments, we initially trained the target model using the IoT dataset [7], focusing on primitive hunting for entire executable files. Subsequently, we employed a 70-30 splitting technique for testing the Malconv model with the IoT dataset, achieving over 99% detection accuracy for malicious payloads.

D. Adversarial Example Generation

After generating and evaluating samples using Malconv, we advanced to launch attacks on both target threat hunting models: Malconv and the Multi-kernel SVMs model. The outcomes, detailed in Table II, unequivocally showcase the efficacy of our approach in successfully evading detection by Malconv and the multi-kernel models. This evasion is achieved through the strategic injection of adversarial payloads into IoT executable files.

To further illustrate the impact of our adversarial examples on the training process, we present Figure 6. This figure displays the training loss function (MSE) of Malconv, specifically when trained with the generated adversarial dataset. The observed fluctuations in the training loss underscore the model's adaptation to the adversarial examples, further emphasizing the efficacy of our approach in influencing the learning dynamics of the target threat hunting model.

Furthermore, to visually demonstrate the similarity between the generated adversarial examples and their non-adversarial counterparts, we present Figure 4. This figure encapsulates the distribution of the top 20 feature components in both adversarial and non-adversarial executable files, as assessed by MMD The striking similarity between the two distributions underlines the model's capability to craft adversarial examples that closely resemble legitimate samples, substantiating the effectiveness of our proposed adversarial example generation approach.

Our evaluation metrics, including evasion rates and detection accuracy, further underscore the potential of random adversarial payloads. For instance, an evasion rate of 24% was observed against the target model when employing randomly generated payloads, compared to 16% with deterministic approaches.

Target Model	Evasion method	Detection Accuracy	Evasion Rate
Malconv [8]	Code Cave Injection	98%	∼16%
Multi-kernel [6]	Code Cave Injection	99%	∼24%
Malconv [11]	Random Byte Addition	N/A	∼10%

TABLE II: Comparative Analysis of Evasion Rates: Assessing the Effectiveness of Our Proposed Attack Model Against Established Benchmarks.

E. Generalizability and Cybersecurity Impact

Our approach has broad potential across ML-based malware detectors, emphasizing the need for dynamic defenses in IoT devices. Enhanced security measures are critical given the demonstrated evasion capabilities, highlighting the urgency for evolving IoT security protocols.

F. Evaluation of Adversarial Payload Effectiveness

Our analysis of the generation of random adversarial payloads revealed a significant insight: the path to a successful evasion against the MalConv model typically began after approximately 1000 attempts. This variability underscores the iterative nature of crafting effective adversarial examples and the importance of understanding the target model's vulnerabilities. The convergence phenomenon, depicted in Figure 7, visualizes the iterative process toward achieving an effective adversarial payload, highlighting the blend of persistence and

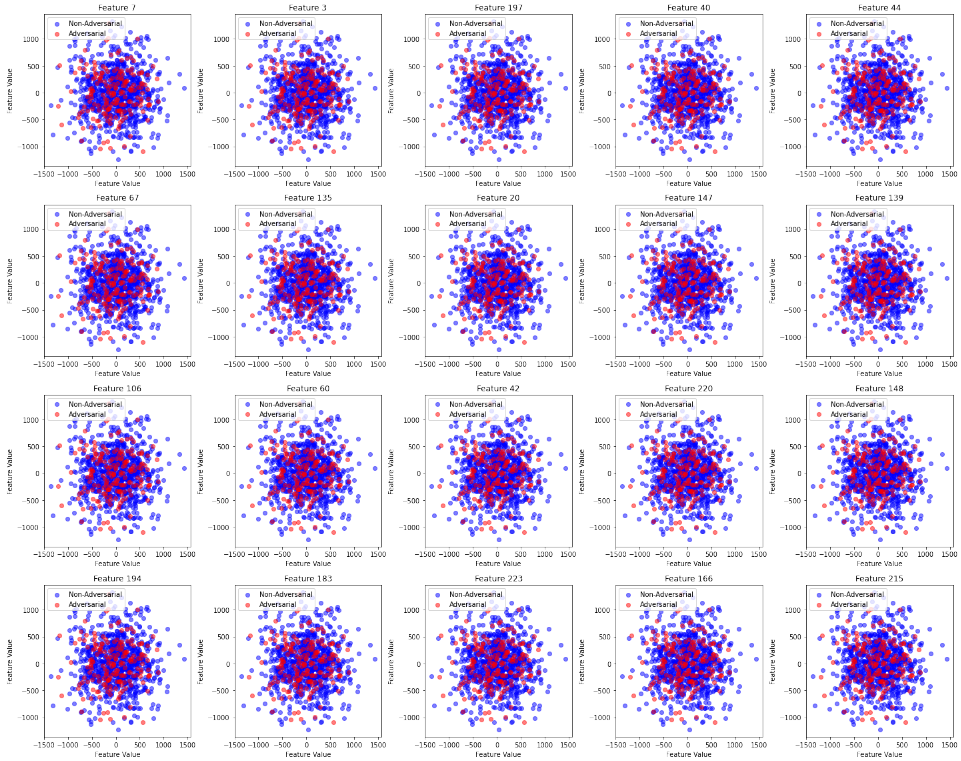


Fig. 3: PCA scatter plot of top 20 feature components for adversarial and non-adversarial samples

strategic insight required in this endeavor. Future work will aim to explore this area more comprehensively, addressing the complexities and challenges identified.

G. Comparison with Other Attack Strategies

Our method emphasizes stealth and operational integrity, leveraging code caves for enhanced evasion in IoT-specific malware detection models. Unlike [11]'s approach, which requires adding approximately 2000 bytes to achieve only a 10% evasion rate—a change likely to alert IoT malware detectors due to the substantial byte increase—our strategy employs strategic, minimalistic modifications (∼256-512 bytes code-cave). These adjustments yield adversarial examples indistinguishable from genuine files, significantly lowering the risk of detection. Our refined approach not only ensures better adaptability and sophistication within IoT application security but also maintains a lower computational footprint, crucial for cloud-edge devices with limited resources, as detailed in Table II.

V. Conclusion and Discussion

The advancement of IoT devices, equipped with sophisticated operating systems, has introduced new vulnerabilities, particularly in the form of malicious payloads. In addressing these challenges, this paper presented a novel adversarial model aimed at probing the vulnerabilities of ML-based malware detection systems, specifically targeting the Malconv model through strategic payload insertion into code caves. Our methodology demonstrated a notable reduction in detection accuracy, highlighting the potential for adversarial examples to bypass current security measures.

Our experiments on a real-world dataset from cloud-edge IoT devices underscored the effectiveness of our approach, reducing Malconv's classification accuracy significantly. Future endeavors will concentrate on developing robust defensive strategies to mitigate such adversarial threats, enhancing the resilience of malware detection frameworks within IoT environments.

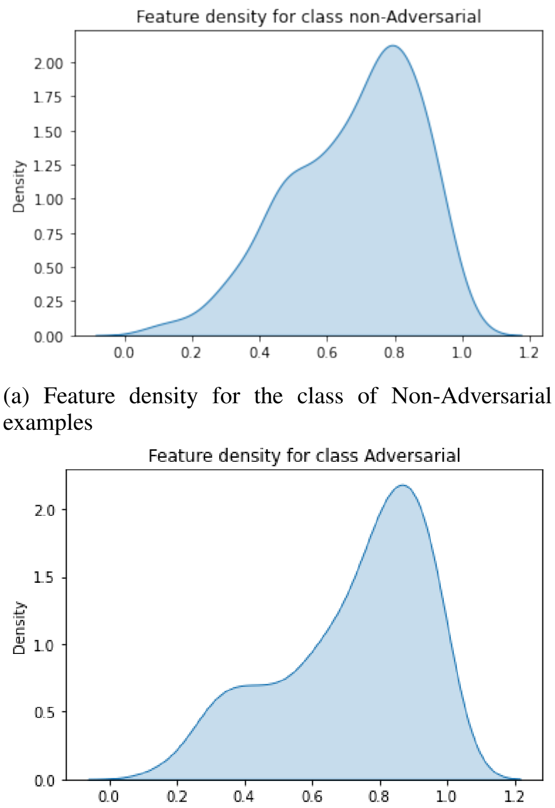
We have illustrated the strategic importance of minimizing modifications and incorporating randomness in adversarial payload generation, balancing operational stealth with evasion effectiveness. Moving forward, our research will expand to investigate these aspects further, alongside exploring the adaptability of our techniques across different detection models and environments. This will include a focus on enhancing the sophistication and unpredictability of adversarial attacks, aiming for broader applicability and robustness in countering malware threats across the IoT landscape.

References

[1] A. Al-Meer and S. Al-Kuwari, "Physical unclonable functions (puf) for iot devices," *ACM Computing Surveys*, vol. 55, no. 14s, pp. 1–31, 2023.

[2] X. Zhang, L. Hao, G. Gui, Y. Wang, B. Adebisi, and H. Sari, "An automatic and efficient malware traffic classification method for secure internet of things," *IEEE Internet of Things Journal*, 2023.

[3] H. HaddadPajouh, R. Khayami, A. Dehghantanha, K.-K. R. Choo, and R. M. Parizi, "Ai4safe-iot: An ai-powered secure architecture for edge layer of internet of things," *Neural Computing and Applications*, vol. 32, no. 20, pp. 16119–16133, 2020.



(a) Feature density for the class of Non-Adversarial examples

(b) Feature density for the class of Adversarial examples

Fig. 4: Feature density of generate samples bytecode for adversarial and non-adversarial train set classes

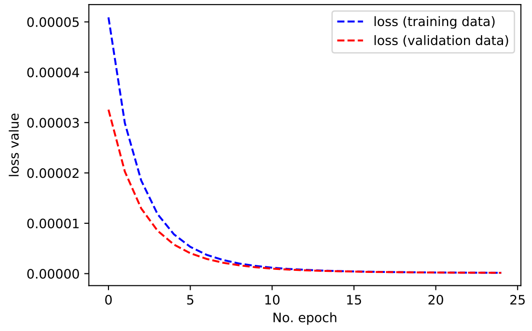


Fig. 5: Training loss function (MSE) of Malconv with IoT dataset.

[4] S. K. Smmarwar, G. P. Gupta, and S. Kumar, "Deep malware detection framework for iot-based smart agriculture," *Computers and Electrical Engineering*, vol. 104, p. 108410, 2022.

[5] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security)*, pp. 1093–1110, 2017.

[6] H. Haddadpajouh, A. Mohtadi, A. Dehghantanaha, H. Karimipour, X. Lin, and K.-K. R. Choo, "A multikernel and metaheuristic feature selection approach for iot malware threat hunting in the edge layer," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4540–4547, 2021.

[7] H. HaddadPajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, "A deep recurrent neural network based approach for internet of things malware threat hunting," *Future Generation Computer Systems*, vol. 85, pp. 88–96, 2018.

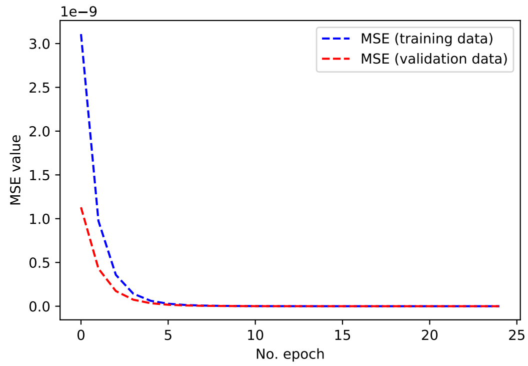


Fig. 6: Training loss function (MSE) of Malconv with the generated adversarial dataset.

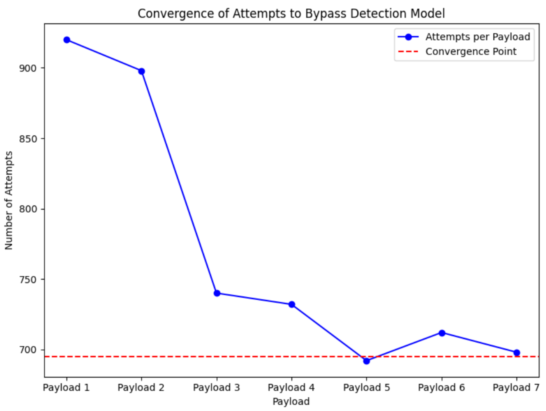


Fig. 7: Visualization of the convergence point in the process of generating successful adversarial payloads against the MalConv model.

[8] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole exe," *arXiv preprint arXiv:1710.09435*, 2017.

[9] W. Song, X. Li, S. Afroz, D. Garg, D. Kuznetsov, and H. Yin, "Automatic generation of adversarial examples for interpreting malware classifiers," *arXiv preprint arXiv:2003.03100*, 2020.

[10] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, "Learning to evade static pe machine learning malware models via reinforcement learning," *arXiv preprint arXiv:1801.08917*, 2018.

[11] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *IEEE 26th European signal processing conference (EUSIPCO)*, pp. 533–537, IEEE, 2018.

[12] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

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

[14] K. Aryal, M. Gupta, and M. Abdelsalam, "A survey on adversarial attacks for malware analysis," *arXiv preprint arXiv:2111.08223*, 2021.

[15] C. Xie, M. Tan, B. Gong, J. Wang, A. L. Yuille, and Q. V. Le, "Adversarial examples improve image recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 819–828, 2020.

[16] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and

defenses for deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[17] H. Gao, S. Cheng, Y. Xue, and W. Zhang, "A lightweight framework for function name reassignment based on large-scale stripped binaries," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 607–619, 2021.

[18] X. Li and Q. Li, "An irl-based malware adversarial generation method to evade anti-malware engines," *Computers & Security*, vol. 104, p. 102118, 2021.

[19] W. Song, X. Li, S. Afroz, D. Garg, D. Kuznetsov, and H. Yin, "Mab-malware: A reinforcement learning framework for blackbox generation of adversarial malware," in *Proceedings of the 2022 ACM on Asia conference on computer and communications security*, pp. 990–1003, 2022.

[20] Z. Wan, X. Xia, D. Lo, and G. C. Murphy, "How does machine learning change software development practices?," *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1857–1871, 2019.

[21] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Functionality-preserving black-box optimization of adversarial windows malware," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3469–3478, 2021.

[22] 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*, 2019.

[23] M. Schloegel, T. Blazytko, M. Contag, C. Aschermann, J. Basler, T. Holz, and A. Abbasi, "Loki: Hardening code obfuscation against automated attacks," in *31st USENIX Security Symposium (USENIX Security 22)*, pp. 3055–3073, 2022.

[24] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," *arXiv preprint arXiv:1702.05983*, 2017.

[25] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach, "Adversarial learning in the cyber security domain," *arXiv preprint arXiv:2007.02407*, 2020.

[26] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.