AdIoTack: Quantifying and Refining Resilience of Decision Tree Ensemble Inference Models against Adversarial Volumetric Attacks on IoT Networks

Arman Pashamokhtari, Gustavo Batista, and Hassan Habibi Gharakheili
UNSW Sydney, Australia
Emails: {a.pashamokhtari,g.batista,h.habibi}@unsw.edu.au

Abstract*—Machine Learning-based techniques have shown success in cyber intelligence. However, they are increasingly becoming targets of sophisticated data-driven adversarial attacks resulting in misprediction, eroding their ability to detect threats on network devices. In this paper, we present *AdIoTack*[1], a system that highlights vulnerabilities of decision trees against adversarial attacks, helping cybersecurity teams quantify and refine the resilience of their trained models for monitoring and protecting Internet-of-Things (IoT) networks. In order to assess the model for the worst-case scenario, AdIoTack performs white-box adversarial learning to launch successful volumetric attacks that decision tree ensemble network behavioral models cannot flag. Our *first* contribution is to develop a white-box algorithm that takes a trained decision tree ensemble model and the profile of an intended network-based attack (*e.g.,* TCP/UDP reflection) on a victim class as inputs. It then automatically generates recipes that specify certain packets on top of the indented attack packets (less than 15% overhead) that together can bypass the inference model unnoticed. We ensure that the generated attack instances are feasible for launching on Internet Protocol (IP) networks and effective in their volumetric impact. Our *second* contribution develops a method to monitor the network behavior of connected devices actively, inject adversarial traffic (when feasible) on behalf of a victim IoT device, and successfully launch the intended attack. Our *third* contribution prototypes AdIoTack and validates its efficacy on a testbed consisting of a handful of real IoT devices monitored by a trained inference model. We demonstrate how the model detects all non-adversarial volumetric attacks on IoT devices while missing many adversarial ones. The *fourth* contribution develops systematic methods for applying patches to trained decision tree ensemble models, improving their resilience against adversarial volumetric attacks. We demonstrate how our refined model detects 92% of adversarial volumetric attacks.

I. Introduction

IoT adoption is on the rise in both consumer and business mainstreams. Still, more than half of the connected IoT devices are found vulnerable [39] to a wide range of sophisticated cyber threats like botnets, malware, phishing, or DDoS attacks. According to a report recently published by Nokia [37], IoTs saw a 100% increase in infections in 2020 over the previous year. These vulnerabilities at scale can lead to significant disruption of critical enterprise operations [38], [53], [12] or exfiltration of sensitive data [31]. The lack of a real-time and detailed inventory of connected IoT assets, deployed in large numbers, leads enterprises to operate their network partially blind [44], [20]. This leaves vulnerable devices unmonitored and hence exposes their organization to grave risks [2].

While manufacturers are assumed (expected) to embed appropriate safeguards in the devices for securing them, many IoT devices have shown [39], [37] to be unprotected and can be compromised with little effort from attackers. This paper advocates "network-level" security measures, instead of "device-level" [49]. We note that embedded security implementation can be highly variable across various IoT devices depending on manufacturers, device capabilities, and mode of operation. Therefore, the network-level monitoring approach comes with a number of advantages including: (a) it can be applied to a range of heterogeneous IoT devices; (b) it can be implemented, operated, and upgraded in the cloud by network operators with no dependency to device manufacturers; and (c) it can augment any device-level security implemented by the manufacturer, providing an extra layer of protection.

Given the speed and complexity of modern cyber threats, network security teams are increasingly applying machine learning (ML) techniques to network traffic (packets and/or flows) of IoT devices to model their network behavior [13]. ML-based models [43] are used on the network (running on general computers fed by traffic features) to automatically classify assets from identifiable patterns in their network activity and detect anomalous behaviors [19], [20], [45], indicative of compromise, firmware upgrade, or emerging novel attacks. Learning-based methods offer the ability to respond to situations not explicitly encountered before, replacing processes that would have required formidable manual analysis by human experts.

In the context of IoT cybersecurity, well-trained ML models have proven to effectively capture the intended behavior of IoT devices on a per-type basis. These models can flag deviations in the volume and/or frequency of network activity without being impacted by limited patterns of certain known attacks [36], [45], [19]. This approach is successful primarily because IoT devices display a finite set of activities (with reasonably identifiable patterns) on the network during their regular operation. These behavioral characteristics present an opportunity to train models with purely benign instances obtained from IoT network traffic. The trained models would have the ability to distinguish clearly the "bounded" set of benign behavior from an "unbounded" set of malicious (anomalous/unintended) behavior resulted from of a network-based cyber attack. This gives a significant advantage to ML-based methods against traditional signature-based ones to infer from IoT network traffic.

[1]Funding for this project was provided by CyAmast Pty Ltd.

Traditionally, the "security of ML-based models" has not been the main objective for designing algorithms and developing inference systems in various domains, especially in cybersecurity. Therefore, potential adversaries with certain incentives aim to subvert the ML model either during training or operation which is known as adversarial attack. Attackers may poison the training instances to influence the resulted model. They may attempt to carefully manipulate network traffic at run-time to flip predictions, yielding a poor performance of the inference model in distinguishing the malicious instances.

Several countermeasures have been proposed, under the umbrella of Adversarial Machine Learning (AML), with practical successes in the area of machine vision and image recognition [52], [51], [14]. However, in the context of cybersecurity, it is in its nascent stage of development to the best of our knowledge. A recent report by McAfee [27] highlights a few malware families that have bypassed machine learning engines in 2018. It predicts how cyber-criminals will be increasingly employing artificial intelligence techniques to evade detection.

The primary objective of adversarial cyber-attackers is to find loopholes in ML-based network security models like traffic classifiers, anomaly detectors, or intrusion detection systems. To fortify inference models against these targeted attacks, cyber-security teams need to quantify the resilience of their trained model and refine any loopholes. This paper focuses on developing techniques for launching volumetric attacks (known as adversarial evasion attack) on IoT devices without being noticed by a decision tree ensemble inference model that continuously monitors the network traffic. To make these ideas more concrete, let us consider a scenario whereby an attacker aims to launch a TCP SYN reflection attack (common in large-scale DDoS attacks sourced from IoT devices [26]) with 1000 attack packets.

The attacker sends 1000 TCP SYN packets with spoofed source IP address to a target IoT device which replies (reflects) them to the victim destination (the source entity specified in the SYN packets) with 1000 TCP SYN-ACK packets. IoT devices typically generate a small amount of traffic volume on their limited set of TCP/UDP flows; thus, reflection attacks with high rates can be detected by inference models relatively easily. That said, attackers with sufficient *prior knowledge* would have the ability to launch their intended attack (without being detected) by precisely generating and injecting some adversarial network traffic (say, 50 NTP packets along with 5 DNS packets and an SSDP packet) in addition to the intended attack packets (1000 TCP SYN packets). To humans, this additional traffic seems to be irrelevant and random; however, this well-crafted "*recipe*" subverts the model's internal decision-making, leading it to accept this network traffic instance as benign. Finding these attack recipes would depend on the inference model's traffic features and detection algorithm, which requires a well-developed adversarial learning algorithm to generate them precisely. It is important to note that this paper uses the term "attack" in two different contexts. First is the network-based volumetric attack (*e.g.,* TCP SYN reflection) on IoT devices, whereby target IoT devices reflect the incoming attack traffic towards an external victim. Second is the adversarial machine learning attack which is a technique that utilizes prior knowledge to subvert the ML-based inference model, and hence the volumetric attacks can go undetected.

In the literature, researchers have studied adversarial attacks on IoT networks [17], [50], [21], [42], [7]. Prior works primarily focus on developing adversarial models that can learn how to bypass neural network-based attack detection models. In contrast, the literature has not devoted much attention to decision tree-based models. Unlike neural networks, decision tree-based models are not differentiable; thus, well-known approaches like [18], [40], [11] are not compatible with their structure. Additionally, given the dynamics of network traffic and some constraints on Internet Protocol (IP) packets, adversarial attacks in the context of network security (unlike image processing) become relatively more challenging. Another gap in the current literature is that adversarial attacks, to the best of our knowledge, have never been executed in a real network protected by an ML-based inference model. Lastly, no prior systematic attempt has been made to improve the resilience of decision trees against these sophisticated attacks without manipulating the training process which requires re-training the model.

This paper presents AdIoTack, a systematic approach for learning and launching adversarial attacks on IoT networks protected by decision tree ensemble models. We make four main contributions: **(1)** We develop a novel adversarial learning algorithm named offline learning that automatically generates adversarial attack "recipes" given an intended volumetric attack on a target class, subverting a trained decision tree ensemble model (§III). We consider two representative network-based volumetric attacks, namely TCP SYN and SSDP reflection, which are widely used on IoT devices for launching DoS/DDoS attacks [22], [26], [30], [35]. Adversarial recipes specify certain overhead packets (2% for TCP SYN reflection and 14% for SSDP reflection) to be injected for an intended volumetric attack to go unnoticed; **(2)** We develop an online execution method that launches the adversarial instances (learned from offline learning) on a real IoT device network monitored by a decision tree ensemble inference model (§IV); **(3)** We demonstrate the performance of AdIoTack by applying it to our testbed comprising of nine consumer IoT devices. We show our online attack execution has at least a 95% chance of successfully launching an adversarial attack when the attacker has the prior knowledge of the inference model cycles (§V); and, **(4)** We develop a method for patching decision trees (without a need for re-training), making them more robust against adversarial volumetric attacks while maintaining the inference accuracy for benign traffic (§VI). Our refined model can detect all adversarial SSDP reflection attacks with low (less than 200 attack packets per minute), medium (between 200 and 700 attack packets per minute), and high (greater than 700 attack packets per minute) impact rates. For adversarial SYN reflection attacks, the refined model detects 63% of low, 75% of medium, and 100% of high impact attacks.

II. Related Work

Cyber Intelligence for IoT Infrastructure: The cybersecurity of IoT networks differs from that of non-IoT or traditional IT networks where general-purpose computers, smartphones, or tablets display an unbounded range of network behaviors reflecting their users' online activity. IoT devices, in contrast, have a limited range of activities (with slight variations but predictable) in the traffic pattern. While modeling benign behavior of non-IoTs is complicated (or even impossible), IoT devices' intended activity can be formally defined and enforced on the network. A "whitelist" of IoT behaviors, specified in the form of MUD (Manufacturer Usage Description) profiles, has been employed to detect anomalies [20]. Researchers have employed various ML techniques to learn from the benign behavior of IoT devices to monitor their health and detect malicious incidents [36], [45], [19], [41]. The use of pure benign instances from IoT network traffic in training inference models, without the inclusion of known attack (malicious) instances, enables them to become more robust against unseen and morphing cyber-attacks.

Adversarial Attacks: Adversarial attacks were first studied in the context of image recognition, where the objective is to create an adversarial copy of a benign image that is indistinguishable to human eyes. However, the image classifier mispredicts the adversarial copy. Adversarial learning in cybersecurity differs from that in the image recognition domain in (1) in image recognition, the similarity of the adversarial instance and the benign instance, expected to look identical to human eyes, is the primary constraint of the problem. However, this constraint is relaxed in cybersecurity and traffic inference problems; (2) there is no requirement for the impact or intensity of intended attacks in image recognition problems, *i.e.,* any adversarial instances (with any level of deviations from the benign instance) are accepted as long as they subvert the model; (3) Practicality of adversarial attacks (*i.e.,* executing them in a real environment) is of concerns in cybersecurity since every adversarial instance (a set of numerical values) may not be necessarily realized as network packets.

Adversarial attack on differentiable models like neural networks has been widely studied both in image recognition and cybersecurity research problems, while non-differential models like decision tree-based models remain relatively under-explored. Also, adversarial learning techniques (computing gradients of the model's loss function) for differentiable models cannot be readily applied to non-differentiable models. Work in [15] developed a method that replaces last few layers of a neural network model with a Random Forest, hide the gradients of the neural network and perhaps protecting it against adversarial attacks.

These prior works have studied evasion attacks on decision tree-based models [9], [23], [54]. Authors in [9] showed the vulnerability of Gradient Boosting and Random Forest models against adversarial evasion attacks using methods of [10], [23]. They also developed a robust technique to avoid adversarial attacks by incorporating adversarial instances into the training phase to shape the ensemble model in such a way it still gives high accuracy for attack inputs. Work in [54] defined its adversarial attack in a way to find minimal changes to a single feature of a benign instance that would result in misclassification irrespective of the model confidence. Our work, instead, incorporates classification scores so that adversarial instances receive relatively confident misclassification from the target model. Work in [23] developed mixed-integer linear programming (MILP) techniques for solving the adversarial attack optimization problem on tree ensembles. All existing works applied their methods to public datasets like MNIST [28] widely used in computer vision research studies. In contrast, our paper is the first (to the best of our knowledge) to develop techniques for adversarial attacks against tree ensembles in IoT cybersecurity, with some distinct features explained at the end of this section.

Adversarial Attack Approaches in IoT: Existing works in IoT cybersecurity primarily adopt techniques from the image recognition domain with little adaptation and contextualization. Hence, some of the fundamental challenges and differences (discussed above) remain unaddressed. Authors of [7] employ reinforcement learning (RL) to generate adversarial instances against their multi-class inference model (four classes of cyberattacks plus a class of benign traffic). In that work, the adversary iteratively adds random noises to samples of an attack dataset and presents them to the inference model until they get classified as benign. However, randomly generated numerical instances do not necessarily represent realistic cyberattacks. Authors of [1] studied the impact of adversarial attacks on models that detect malware-infected IoT applications. Their neural network-based model infers from graph-like features of the binary file of applications. Work in [17] trained a set of binary-class (malicious versus benign) inference models, each per unit of IoT devices. The authors used GAN (Generative Adversarial Networks), a well-known adversarial learning technique for neural networks (initially designed for image classification context). The primary objective is to improve the resilience of their model during training by including adversarial instances in the training set. Another work [21] focused on neural networks for detecting network-based attacks. Their detection is done by a multi-class inference model (with four classes of cyberattacks and one class of benign traffic). Their objective was to manipulate numerical instances of either attack or benign for which the model makes incorrect predictions. The authors directly borrowed techniques from image recognition and did not demonstrate the feasibility of subverting the model on real network. Importantly, they do not attempt to refine the vulnerable inference model.

Prior works in IoT cybersecurity were directly adopted from those in computer vision, whose primary focus was on neural network models. In this paper, we primarily aim to develop methods for generating instances and launching adversarial attacks on a live network of IoT devices. To the best of our knowledge, no prior work focused on decision tree-based models in this context while considering the practical challenges. Also, this is the first time in the literature that an adversarial attack is indeed executed on a real network of IoT devices. Our work is distinguished from the prior adversarial attack studies on decision tree-based models in the following ways: (1) the context of all existing works is computer vision, whereby objectives are different from those in cybersecurity; (2) analyzing inter-dependency across many features (this paper) versus tweaking only one feature (prior works); (3) investigating the impact of volumetric network attacks and the feasibility of being launched on real networks; (4) no existing work explored decision tree sequence order to find adversarial instances, while this paper analyzes how the order by which decision trees of an ensemble model are processed can affect the richness of adversarial instances; (5) our refinement technique (improving resilience) is done post-training, which does not require re-training of the model (in contrast to [6], [9]) and hence agnostic to a set of known adversarial instances.

III. AdIoTack: System Architecture and Adversarial Learning

This section describes our AdIoTack system, including major functional decisions and system components (§III-A), offline learning (§III-B), the core algorithms (§III-C) and consistency verification (§III-D).

A. *Threat Model: Functional Decisions and System Components*

The main objective of AdIoTack is to help cyber-security teams quantify the resilience of their decision tree-based models against adversarial volumetric attacks that target *integrity* of the inference models. Given a well-trained traffic inference model that is able to detect "non-adversarial" attacks relatively easily, AdIoTack generates adversarial instances that the model mispredicts as benign. Note that ML models may be tested for other aspects like their *availability*, *i.e.,* whether they remain operational while being flooded by malicious requests [3], which is beyond the scope of this paper. In terms of influence, adversarial attacks may be either *causative* or *exploratory* [3]. In causative attacks, training data instances are poisoned in order to manipulate the inference logic. In exploratory attacks which happens post-training, well-crafted malicious traffic instances that resemble benign instances are used to bypass the model while launching an attack. Our focus in AdIoTack is on exploratory attacks.

Exploratory attacks require a procedure for finding adversarial instances. For models like neural networks, logistic regression, and SVM, this procedure can be done by using gradient-based methods such as gradient descent, fast gradient sign [18], Jacobian saliency map [40], or auto projected gradient descent [11]. While differentiable models like neural networks have been widely applied to computer vision tasks, non-differentiable models like decision trees have received less attention. In cybersecurity applications, instead, decision tree-based models are deemed more attractive because of some of their innate properties: (i) decision tree-based algorithms offer faster training time compared to neural networks [29]; (ii) decision tree-based models require less computing resources at runtime (post-training) [48] (iii) better handling of imbalanced datasets [25] without the need for data normalization or scaling prior to training; (iv) Random Forest algorithm is reasonably resilient to the overfitting problem [4]; (v) Decision tree-based models often yield good results in terms of accuracy and explainability, *i.e.,* their internal decision-making process is visible during testing, which is highly desirable for cyber-analysts (particularly for IoT network security [32], [16], [34], [43], [46]) in taking remedial and/or preventive actions. In contrast, competitors like neural networks do not provide insights into their internal process of inference; (vi) Random Forest classifiers are more robust than other models studied in [24] against adversarial evasion attacks, hence, become more difficult to evade. Because of these distinct features available in decision tree-based models and fewer existing works compared to a rich literature for neural networks, this paper focuses on decision tree-based models.

One can quantify the resilience of an ML model in three different scenarios: (a) White-box represents the "worst-case" scenario when the adversary comes with full knowledge of the model's structure, algorithm, and features; (b) Gray-box assumes that the adversary's knowledge is relatively limited (say, partial access to the training data); (c) Black-box is when the inference model is targeted with no prior knowledge – it can only be probed, say, via API calls. We choose the white-box approach because AdIoTack is primarily designed as a tool to empower defenders (network operators) in preparing for worst-case scenarios, identifying, and mitigating vulnerabilities of decision tree-based models against adversarial attacks. It is important to note that a white-box approach provides defenders with an upper-bound assessment by revealing all existing loopholes in their inference model that can be exploited (even with relatively lower probabilities) by adversarial attacks. Therefore, having an ultimate white-box resilience assessment is recommended as a best-practice [8]. Although gray/black box approaches highlight more realistic loopholes, they cannot serve as an ultimate resiliency check, *e.g.,* one model may be robust against gray-box attacks but falls short of expectations in white-box ones. It is important to note that although obtaining information about the properties of the model is practically challenging for adversaries, it is not impossible. Techniques (beyond the scope of this paper) such as social engineering, penetration, and scanning can be employed for that purpose [33]. Lastly, a white-box approach assesses the resilience of a model by its innate properties without making assumptions about how those properties are kept "confidential".

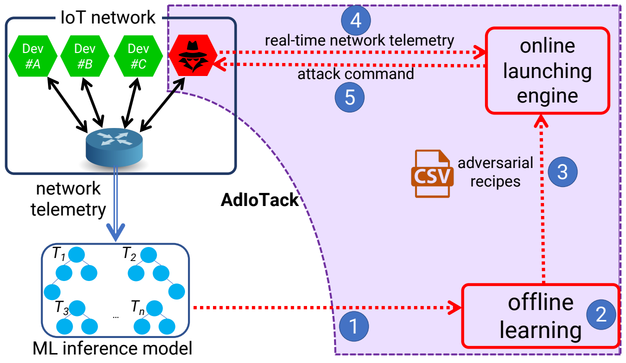


Fig. 1. System architecture of AdIoTack.

Fig. 1 shows the architecture of our AdIoTack system. On the top left, we see a network of IoT devices (green boxes) whose traffic is monitored periodically (*e.g.,* every one minute) by a decision tree ensemble model (*i.e.,* trees T_1 to T_n). The network router extracts the required telemetry (*e.g.,* packet/byte count of different traffic flows) from the network traffic of each IoT device and passes it to the ML inference model (bottom left). The model continuously infers the most probable class of the connected devices from the received telemetry. We use the classification score of the ML model to compute a device-specific threshold to distinguish benign behavior from malicious. That way, if the model yields a score below the given threshold, it is an indication for malicious behavior (details discussed in §III-B). For AdIoTack (shown by shaded region), there are two essential phases, namely: (a) learning and (b) execution (launching).

In the learning phase (steps $\boxed{1}$ and $\boxed{2}$ in Fig. 1), the offline learning module uses the inference model (note the white-box scenario) to learn blind spots of the model to generate adversarial recipes (step $\boxed{3}$). Each adversarial recipe consists of a number of conditions over features of the inference model (*e.g.,* $10 < f_1$, $f_2 \leq 50$, and $20 < f_3 < 40$); thus, each recipe can generate several adversarial instances that conform to the recipe's conditions. For example, an adversarial instance for the mentioned recipe could be $f_1 = 15$, $f_2 = 30$, and $f_3 = 25$. For execution, network telemetry (step $\boxed{4}$) similar to inputs of the inference model must be collected, so that an "appropriate" adversarial recipe can be selected based on the current state of the network. To obtain real-time telemetry and execute the intended adversarial attack, network traffic of IoT devices is snooped. As a part of our threat model, we assume a "malicious agent" like an infected smartphone or a computer is already inside the local network (the red box inside the IoT network on the top left) for this purpose.

To obtain network telemetry in real-time, the malicious agent can passively (*e.g.,* sniffing) or actively (*e.g.,* man-in-the-middle via ARP poisoning) monitor the behavior of victim IoT devices. The passive approach is stealthier to perform but could be practically challenging in some cases, like when the malicious agent and its victims are on different physical access mediums (wireless versus wired). For AdIoTack, we employ the active mode for adversarial network monitoring. Also, our method only analyzes packet metadata (headers), so it has no issue with encrypted payloads.

In the next step, the online engine starts a search for feasible adversarial recipes, given the current state of the network. From those candidate recipes, the online engine selects the closest one to the current state of the network to minimize the amount of adversarial packets (overhead) to be injected (in real-time to the network traffic) on behalf of the victim. Eventually, the intended volumetric attack is launched (step $\boxed{5}$) by judiciously crafting network traffic matching the chosen recipe. Note that the local victim will reflect/amplify the attack traffic onto an ultimate victim on the Internet. It is important to note that the adversarial packets (overhead traffic) are injected on behalf of local victims for their reflected/amplified traffic to go undetected.

B. Offline Adversarial Learning

Offline adversarial learning is a process of generating adversarial recipes which: (a) result in desired malicious impact, and (b) can go undetected *i.e.,* the traffic inference model raises no anomaly flag. The results of offline adversarial learning show to what extent the given model is vulnerable to adversarial volumetric attacks.

We employ ML models' *classification score* as a measure for detecting behavioral changes. We define a classification score threshold for each class (device type) according to the training results. At run-time, any instance with a classification score below the given threshold is flagged as an anomaly. We can define the adversarial learning problem formally as below:

$$\forall d \in \mathbf{D},\ \{x \mid \text{pred}(x) = d \text{ and } \text{score}(x) \geq T_d\}$$
$$\text{s.t.} \quad \forall f \in \mathbf{F},\ x_f \geq f_{min}$$

This equation defines a set of adversarial instances $\mathbf{X}$ for each IoT device type $d \in \mathbf{D}$. The output of the inference model for each adversarial instance $x \in \mathbf{X}$, $\text{pred}(x)$, is d and the classification score, $\text{score}(x)$, is greater than or equal to the classification threshold of the corresponding class, T_d. Therefore, every adversarial instance x can bypass the inference model's detection as it satisfies the minimal classification score. Also, we define a set of constraints to guarantee that $\mathbf{X}$ fulfills the requirements of the desired attack. As the scope of this paper is volumetric attacks, we define these constraints as a set of lower bound conditions for each feature f that belongs to the target feature set $\mathbf{F}$. The set $\mathbf{F}$ contains the relevant features that would be affected by the desired attack, *e.g.,* TCP packet and byte count features in TCP SYN reflection attack.

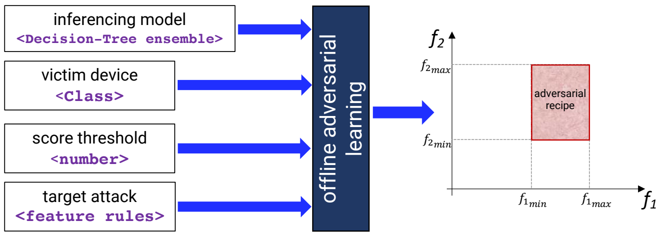


Fig. 2. Inputs and outputs of our adversarial learning process.

Fig. 2 illustrates inputs and outputs of our offline adversarial learning process. For illustration purpose we only show two features, f_1 and f_2. Suppose we want to test whether the model is vulnerable to a volumetric attack on a victim device (*e.g.,* Amazon Echo), where the attack offers more than 1000 packets over f_1. The desired attack impact is captured by a set of conditions for the target feature(s) (in our example, $f_1 > 1000$). The offline adversarial learning process then searches over a given tree ensemble model in order to determine recipes that the ensemble model predicts them as the class of the victim device with a classification score of greater than or equal to the given threshold. Adversarial recipes can be seen as a region in an N dimension space, being N the number of features required by the inference model. Conditions on target features (f_1 in our example) specified by the adversarial recipes must be consistent with those conditions given as input for the intended attack (*i.e.,* $f_1 > 1000$). In other words, any point in an adversarial recipe region is an adversarial instance that satisfies the conditions of the intended attack.

C. The AdIoTack Algorithms

AdIoTack focuses on performing adversarial attacks on a given decision tree ensemble model. The model consists of several decision trees. Weighted or unweighted voting gives the final classification, *i.e.,* the fraction of trees yielding the final output determines the prediction and classification score.

In order to explain how voting works, let us consider an example relevant to our device class inference problem. Let say, if a benign traffic instance x belongs to IoT device class "`A`" (ground-truth), and say 95% of the trees in the ensemble model classify it correctly, the final prediction would be "`A`" with the score of 0.95. Now, assume "`A`" is under a cyberattack, having a traffic instance $\hat{x}$. The model is expected to give a lower score [43] for $\hat{x}$ since because of the deviation from benign instance x there will be disagreement among various trees, each inferring from a specific set of features. Certain features of $\hat{x}$ will significantly deviate from their expected normal range, leading a portion of trees to classify $\hat{x}$ as a class other than "`A`", and other features may remain relatively

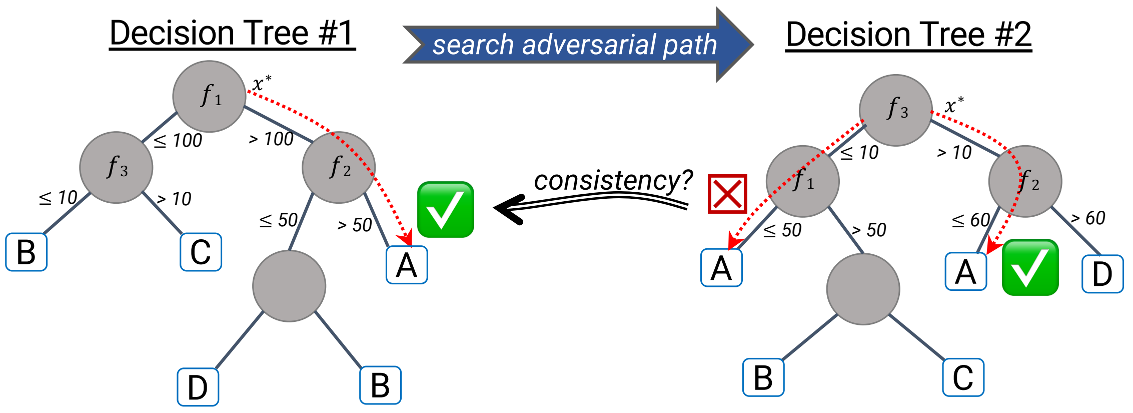


Fig. 3. Searching adversarial paths in two decision trees: The adversary starts traversing decision tree #1 with no prior rule – a path to target class "A" is found (✓). Moving to decision tree #2, the first path becomes inconsistent with rules of the chosen path from decision tree #1 (×), and thus an alternative path is sought and found (✓).

unchanged hence some other trees predict the class of $\hat{x}$ as "A". Therefore, the attack can be flagged with a good chance. In adversarial attack, we aim to find a recipe which has the same malicious impact of $\hat{x}$ in such a way that the majority of decision trees predict it as "A", resulting in high score and implying it is still benign.

In the context of decision trees, we define an *adversarial path* to be the path from the root node of a given decision tree to a leaf with the label of the victim device that we aim to launch an attack on it. Each adversarial path is associated with a set of conditions that are met across the path nodes. Fig. 3-*left* shows how the adversarial path is found on an illustrative decision tree. Gray circles are decision nodes, each checking a certain condition (*e.g.,* $f_1 \leq \tau_1$), proceeding to subsequent branch (left or right, depending on the check result). Adversarial path x^* is a complete path from the root to a leaf node, landing on the target victim class "A". AdIoTack performs tree traversals using the pre-order method – the root node is first visited, then recursively a pre-order traversal of the left subtree is performed, followed by a recursive pre-order traversal of the right subtree.

Note that searching the adversarial path needs to be extended to all decision trees inside the ensemble model where consistency of conditions across trees becomes a non-trivial challenge. Fig. 3 shows an illustrative example of searching for an adversarial path across two decision trees while conditions (of paths in the two trees) need to be consistent. Assume we want to target class "A" with a cyberattack that requires $\{f_1 > 1000\}$ (*i.e.,* target rule). Below, we show the initial recipe to begin the attack:

$$Recipe : f_1 > 1000 \Rightarrow A$$

Starting from decision tree #1 (on the left), AdIoTack finds the first path leading to a leaf with the target class "A" (by traversing the tree in pre-order mode). The adversarial path on decision tree #1 consists of two conditions: $\{f_1 > 100$ and $f_2 > 50\}$, which are consistent with the initial recipe; therefore, we can add them to the recipe:

$$Recipe : f_1 > 1000 \ \wedge \ f_1 > 100 \ \wedge \ f_2 > 50 \Rightarrow A$$

which can be simplified (*merged*) to:

$$Recipe : f_1 > 1000 \ \wedge \ f_2 > 50 \Rightarrow A$$

Algorithm 1 Adversarial evasion attack on tree ensemble model.

```
1: $\mathcal{IM} \leftarrow$ inference model
2: $TR \leftarrow$ set of target rules
3: $TD \leftarrow$ target device class
4: $T_d \leftarrow$ classification score threshold of $TD$
5: function FINDRECIPE($\mathcal{IM}$, $TR$, $TD$, $T_d$)
6:     $R \leftarrow (TR \Rightarrow TD)$                ▷ Recipe
7:     for $t$ in $\mathcal{IM}.trees$ do
8:         $AdvPath \leftarrow$ FINDADVPATH($t.root$, $TD$, $R$, [ ])
9:         if $AdvPath \neq Null$ then
10:            $R \leftarrow$ MERGE($R$, $AdvPath$)
11:        end if
12:    end for
13:    $x^* \leftarrow$ PROJECT($R$)
14:    if $\mathcal{IM}$.PRED($x^*$) = $TD$ & $\mathcal{IM}$.SCORE($x^*$) $\geq T_d$ then
15:        return $R$
16:    else
17:        return $Null$
18:    end if
19: end function
```

Moving on to decision tree #2, the first possible path (in pre-order search) to the target class "A" requires $\{f_1 \leq 50\}$, which is inconsistent with $\{f_1 > 1000\}$ in the current recipe – this path cannot be taken, and hence an alternative path needs to be sought. The other path to the other leaf with class "A" (on decision tree #2) results in two new conditions: $\{f_3 > 10$ and $f_2 \leq 60\}$, not violating the current recipe determined by the path from decision tree #1. The updated recipe is shown below:

$$Recipe : f_1 > 1000 \ \wedge \ f_3 > 10 \ \wedge \ 50 < f_2 \leq 60 \Rightarrow A$$

To generate an adversarial recipe that drives the majority vote to the target class, AdIoTack needs to keep track of conditions and progressively augment them as new adversarial paths are found across individual decision trees. Our search procedure visits each tree in a greedy manner, *i.e.,* it skips a tree if it could not find any consistent adversarial path on it. Also, the order of the trees influences the output of the method. We address this problem by running the algorithm multiple times with different tree ordering to generate new recipes.

Having these intuitive illustrations of the adversarial attack method, let us formally develop it by Algorithms 1 and 2. Algorithm 1 searches for an adversarial recipe for a given target device. This algorithm expects the inference model

Algorithm 2 Searching for a consistent adversarial path.

```
function FindAdvPath($n$, $TD$, $R$, $P$)
    if $n$ is $leaf$ then
        if $n.label = target$ then
            return $P$                                  ▷ Adversarial path
        else
            return $Null$
        end if
    end if
    $r \leftarrow Null$
    if isConsistent($n.cond$, $R$) & isConsistent($n.cond$, $P$) then
        $r \leftarrow$ FindAdvPath($n.left$, $TD$, $R$, $[P \wedge n.cond]$)
        if $r \neq Null$ then
            return $r$
        end if
    end if
    if isConsistent($\neg n.cond$, $R$) & isConsistent($\neg n.cond$, $P$) then
        $r \leftarrow$ FindAdvPath($n.right$, $TD$, $R$, $[P \wedge \neg n.cond]$)
    end if
    return $r$
end function
```

($\mathcal{IM}$), a set of target rules (TR) that are defined based on the target cyberattack, the target class of victim IoT device (TD), and the classification score threshold (T_d) as inputs.

In essence, Algorithm 1 iterates over all trees in the ensemble model ($\mathcal{IM}$). Notice the tree order in a decision tree ensemble model is arbitrary, and different orders may give different results (new recipes or no recipe at all). In §V, we evaluate how different orders can affect the number of generated recipes. For each tree, the algorithm calls the function `FindAdvPath` in Algorithm 2 to find an adversarial path consistent with the current recipe generated so far. If such a path is found, the corresponding adversarial path's rules are merged with the current recipe rules (`Merge`). The following examples show how `Merge` function works:

`Merge`$(f \leq \tau_1,\ f \leq \tau_2,\ \tau_1 \leq \tau_2) = f \leq \tau_1$, or

`Merge`$(f > \tau_1,\ f > \tau_2,\ \tau_1 > \tau_2) = f > \tau_1$

As there is no guarantee that `FindAdvPath` will find an adversarial path for every tree, we need to validate the efficacy of the final recipe (whether it can bypass the model or not). We may find adversarial paths effective on only a small subset of the decision trees, or even no tree at all. In this case, the resulted adversarial instances will become ineffective in yielding the target class and/or the score threshold. We validate the efficacy of a recipe by generating a representative adversarial instance, x^*, using `Project` function. We note that every instance of a recipe would fall under the desired adversarial paths determined by Algorithm 1; therefore, if x^* can bypass the model, any other adversarial instance generated from the given recipe can do the same. The instance x^* is presented to the model to obtain the prediction. If it is classified as the target class with a classification score greater than or equal to the specified threshold (T_d), the algorithm approves and returns the recipe.

Algorithm 2 (invoked from within Algorithm 1) searches for a consistent adversarial path (P) to a leaf with the target class label (TD) on a given tree. The algorithm is a direct application of pre-order traversal for binary trees. During a search, we check for consistency (`IsConsistent`) between the nodes' condition and the recipe (explained in III-D). This helps to prune the search space, reducing the average execution time. Algorithm 2 assumes binary decision trees; therefore, the nodes are described by only three fields (`left`, `right` and `cond`). The left subtree is associated with the condition `cond` and the right subtree with logical negation of it (¬`cond`).

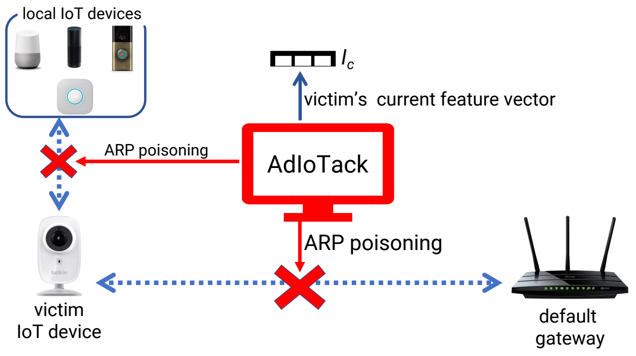


Fig. 4. Local adversary (an infected machine) collects traffic features before launching an online attack.

Note that an adversary may choose to stop searching when a certain level of majority across trees (*i.e.,* classification score) is obtained. Our algorithm, instead, aggressively aims to maximize its chance of subverting the model by (greedily) searching all possible trees. Given this greedy nature of our offline adversarial learning, its time complexity is $\mathcal{O}(\mathcal{T}.\mathcal{N}.\mathcal{F})$, proportional to the number trees ($\mathcal{T}$), maximum number of nodes in the trees ($\mathcal{N}$), and total number of features ($\mathcal{F}$). It is important to note that our search across trees progresses only in a forward direction. Hence, in some cases, it may not yield any consistent adversarial path on a decision tree (in the middle of iterating over trees). In other words, if a consistent path on a tree is not found, then that tree is skipped without going backward, seeking alternative adversarial paths on the previous trees, and updating the recipe correspondingly.

D. Verifying Consistency of Conditions

Our adversarial learning algorithm is general and can be applied to any decision tree ensemble model. However, certain constraints may be needed when employing the algorithm for specific domain problems. Given the primary use-case of this paper is network-based volumetric attacks, three types of consistency checks are performed before taking a new branch (left or right subtree) while searching for an adversarial path:

Single-feature consistency: This type of consistency is required when different conditions are imposed on a single feature across trees. Let us consider two illustrative conditions $f \leq \tau_1$ and $f > \tau_2$. They are considered consistent if $\tau_2 < \tau_1$, where f denotes a feature that the inference model uses. A single-feature consistency check is required in any problem of adversarial machine learning. However, in the context of cybersecurity, we identified two additional checks essential for launching attacks; otherwise, the offline learning algorithm may generate impractical adversarial recipes that cannot be launched during the online phase.

Frame size consistency: This constraint is specific to the use-case of this paper. Frame size is measured in bytes and has a minimum and maximum length, depending on the implemented technology. For example, an Ethernet frame must be at least 64 bytes for collision detection to work and can be a maximum of 1518 bytes to avoid IP fragmentation. This expected characteristic may get violated by a condition. For example, let us consider a set of conditions on two features of incoming (↓) DNS traffic for a given adversarial recipe:

$$\begin{cases} C_1 : 9 < \boxed{\downarrow\text{DNS packet count}} \\ C_2 : \boxed{\downarrow\text{DNS byte count}} \leq 100 \end{cases}$$

These translate into an average size of incoming DNS frames to be less than 10 bytes, which is impossible on Ethernet networks – sending at least ten frames even with no payload would result in 640 bytes of traffic, violating the condition C_2 above.

Boundary consistency: Average frame size obtained by the ratio of byte count and packet count must be rounded to integer values (cannot be float values). There are certain corner cases where packet count and byte count conditions become inconsistent. The following conditions exemplify this situation:

$$\begin{cases} C_3 : 10 < \boxed{\downarrow\text{DNS packet count}} \leq 15 \\ C_4 : 998 < \boxed{\downarrow\text{DNS byte count}} \leq 1,000 \end{cases}$$

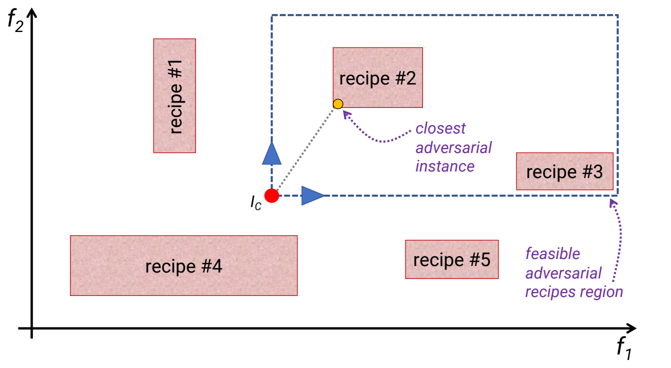
Choosing the combination of $\boxed{\downarrow\text{DNS packet count}} = 11$ and $\boxed{\downarrow\text{DNS byte count}} = 999$ gives a minimum average frame size of 91 B (equally sized). At a high level, this combination would violate C_4 since 11 DNS incoming messages of size 91 B each will amount to 1001 B. Indeed, one may choose to solve this inequality problem by judiciously setting packet sizes (*e.g.,* nine packets of each 100 B and a packet of 99 B) – this is beyond the scope of this paper. We only consider recipes that allow adversarial attacks of equal packet size.

Note that in a single tree, conditions inside each path (from root to a leaf) satisfy single-feature consistency by default. Still, even conditions within a given path of a single tree may violate frame size and boundary consistencies.

IV. Launching Adversarial Volumetric Attack

This section describes how we use adversarial recipes to launch real volumetric attacks on an operational IoT network whose traffic is continuously monitored by a trained inference model. To clarify the method better, we explain this section with a set of the flow-based features that the model uses for inference. Inspired by [43], we choose packet count and byte count of certain network flows: ↓`DNS`, ↑`DNS`, ↓`NTP`, ↑`NTP`, ↑`SSDP`, ↓`LAN`, ↓`WAN`, and ↑`WAN`, proven to be reflective of IoT behavior, during fixed time windows (of length one minute); where ↓ and ↑ indicate incoming and outgoing directions, respectively.

This phase aims to project the current state (traffic feature vector) of a victim IoT device to an adversarial instance using one of the adversarial recipes obtained in the offline learning phase. A naive approach is to launch a cyberattack blindly based on a random recipe. This approach is not guaranteed to bypass the inference model given the variability in volume and frequency of traffic sent/received by the victim



f_2
f_1
recipe #1
recipe #2
recipe #3
recipe #4
recipe #5
I_c
closest adversarial instance
feasible adversarial recipes region

Fig. 5. Finding feasible recipes' region and the closest adversarial instance given current state I_c. For illustration purpose, only two features are shown.

IoT devices during the last time window (*i.e.,* current state). It is important to note that malicious packets (pertinent to the intended network attack) get added to the benign traffic of the victim device. Hence, a more effective strategy is needed that considers the current state of the variable network.

A well-known method to track network traffic and create a feature vector for the victim device is poisoning the ARP tables (discussed in §III) of local IoT devices and default gateway. Therefore, AdIoTack can sit in the middle of these entities by forwarding traffic between them. Fig. 4 illustrates the monitoring phase before launching the attack. AdIoTack maintains a record (feature vector I_c) of the traffic features in the current epoch. Given I_c, AdIoTack would find feasible adversarial recipes towards the end of each epoch. A feasible recipe has to be an upper bound to vector I_c (Fig. 5 shows it in a 2D space). If the victim displays network activities (I_c) exceeding a threshold ($f < \tau$) specified by a recipe, then I_c cannot be projected to that recipe anymore simply because we cannot reduce the amount of traffic that already has been exchanged by the device but we can increase it. Among all feasible recipes, AdIoTack selects the closest adversarial instance to I_c, minimizing the overhead packets to be injected. This strategy requires AdIoTack to be aware of the inference model's timing cycles, *i.e.,* when it is called for prediction. In our evaluation (§V), we will explain how AdIoTack has the chance to execute successful attacks even without syncing with cycles of the inference model.

In this paper, we use linear search to find feasible recipes. Though we do not encounter a challenge in terms of the time complexity of the linear search in our evaluation, there might be situations where a simple linear search becomes inefficient. To achieve certain response time, one may attempt to optimize the search process and/or prune the generated recipes. Both of these objectives are beyond the scope of this paper.

We use an example in Table I to better clarify our method for executing a volumetric TCP SYN reflection attack on Google Chromecast. The top two rows indicate lower-bound and upper-bound feature values specified by the adversarial recipe for launching a successful attack. Note that the symbol "*" indicates unbounded features (no lower/upper limits). The third row shows the current feature vector of Google Chromecast over the last 60 seconds. The bottom row shows the adversarial instance comprising the intended volumetric attack traffic (bold cells) reflected by Google Chromecast and overhead packets (underlined cells) injected by AdIoTack.

TABLE I. AN EXAMPLE OF A FEASIBLE SYN REFLECTION ATTACK ON GOOGLE CHROMECAST.

	↓DNS pkt	↓DNS byte	↑DNS pkt	↑DNS byte	↓NTP pkt	↓NTP byte	↑NTP pkt	↑NTP byte	↑SSDP pkt	↑SSDP byte	↓LAN pkt	↓LAN byte	↓WAN pkt	↓WAN byte	↑WAN pkt	↑WAN byte
Condition ($>$)	2	119	2	194	*	45	*	*	7	4,353	94	9,303	88	45,568	10	5,650
Condition ($\leq$)	*	1,666	*	*	*	*	1	*	9	4,553	148	13,797	*	*	*	*
Current state (I_c)	0	0	0	0	0	0	0	0	0	0	0	0	46	125,857	51	5,752
Adversarial instance	3	120	3	195	1	46	0	0	8	4,354	95	9,304	**1,000**	**74,000**	**1,000**	**74,000**

TABLE II. SPOOFED METADATA FOR EACH FEATURE.

Feature	src MAC	dst MAC	src IP	dst IP	src Port	dst Port
↓ DNS	GW	VIC	*	VIC	53	*
↑ DNS	VIC	GW	VIC	*	*	53
↓ NTP	GW	VIC	*	VIC	123	*
↑ NTP	VIC	GW	VIC	*	*	123
↑ SSDP	VIC	*	VIC	*	*	1900
↓ LAN	*	VIC	LAN IP	VIC	*	*
↓ WAN	GW	VIC	WAN IP	VIC	*	*
↑ WAN	VIC	GW	VIC	WAN IP	*	*

To successfully launch the adversarial volumetric attack, AdIoTack needs to inject certain spoofed packets (the overhead traffic specified by the closest recipe) on behalf of the victim, so that the inference model does not detect any deviation in the behavior of the victim IoT device. Table II summarizes certain packet metadata fields to be modified along with their spoofed value for a given set of traffic features in the specific model we studied. GW and VIC stand for gateway and victim, respectively. WAN IP and LAN IP respectively refer to external (public) and internal (private) IP address. Also, "*" highlights a wildcard value for the respective field.

Practical Challenges of Network-Based Attacks: We discuss a few practical challenges to overcome for successful execution of adversarial attack instances in what follows: (i) in order to intercept bidirectional traffic exchanged between the victim and its local network, AdIoTack needs to employ Gratuitous ARP replies to send a broadcast spoofed ARP reply (on behalf of the victim device) to all local devices. That way, all local traffic destined to the victim device is forwarded to AdIoTack; and, (ii) there might be cases whereby AdIoTack and victim may not share the same network interface (*e.g.,* one is wired and the other one is wireless). In that case, AdIoTack will have to send the spoofed traffic via the default gateway, which will disrupt the MAC-learning tables (wireless vs. wired) of the default gateway's interfaces due to the spoofed source MAC address in those packets (Table II) – this can lead to mis-forwarding of other packets to/from the victim (wireless LAN instead of wired LAN). Disrupting the default gateway's MAC tables results in dropping future packets with the same MAC address, this time as the destination field. To avoid this, AdIoTack sends a specific `ICMP` packet with destination broadcast MAC address (`ff:ff:ff:ff:ff:ff`) and victim's IP address. The victim's reply will revert the MAC tables to their correct state.

TABLE III. THREE ENSEMBLE MODELS PERFORMANCE ON THE TESTING DATASET.

Model	Accuracy	False positive
Random Forest	96%	9%
Gradient Boosting	91%	0.1%
AdaBoost	99%	5%

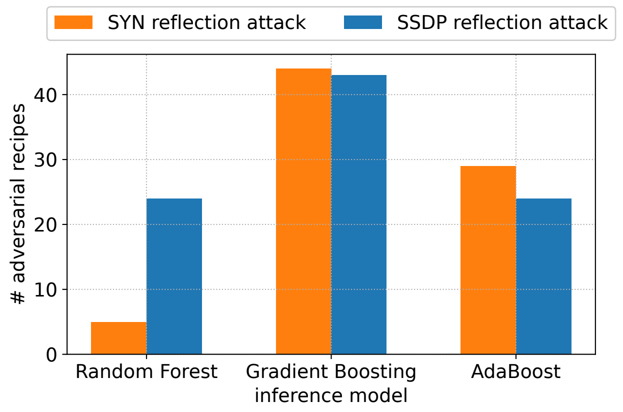


Fig. 6. Number of generated adversarial recipes for three ensemble models for attack impact of 1000 packets.

V. EVALUATION RESULTS

This section evaluates the performance of AdIoTack in its two phases, namely offline adversarial learning, and online adversarial execution. We begin by training three well-known decision tree ensemble models namely Random Forest, Gradient Boosting, and AdaBoost classifiers.

A. The Inference Model

Inspired by [43], we consider flow-level features from network traffic, periodically computed over one-minute windows. That way, no need for inspecting packet payloads. Traffic features include statistics (packet and byte counts) of ↓`DNS`, ↑`DNS`, ↓`NTP`, ↑`NTP`, ↑`SSDP`, ↓`LAN`, ↓`WAN`, and ↑`WAN`; where ↑ indicates upstream traffic from each IoT device, and ↓ highlights downstream traffic to each IoT device. This means a total of 16 features. For training and testing, we use two datasets collected from our testbed (comprising nine IoT devices) during Feb-May 2020. Our full dataset merely contains 954,384 benign instances with no trace of attacks (malicious instances). We use data from February, March, and the first half of April for training (i.e., 668,415 instances) and the second half of April and May for testing (i.e., 285,969 instances). Training based on "only benign" instances helps the model tighten and purify its acceptable boundaries to devices' expected behaviors (a finite set) without crippling the model by instances of certain known attacks (which can be infinite in practice). It has been shown

TABLE IV. EXPECTED CLASSIFICATION SCORE (μ AND σ) OF THE RANDOM FOREST MODEL OBTAINED FROM TRAINING.

	Benign	
IoT class	μ	σ
Amazon Echo	0.94	0.10
Belkin motion	0.96	0.06
Belkin switch	0.86	0.18
Chromecast	0.90	0.16
Hue bulb	0.99	0.01
LiFX bulb	0.72	0.15
Netatmo camera	0.88	0.14
Samsung camera	1.00	0.002
TP-Link switch	0.86	0.10

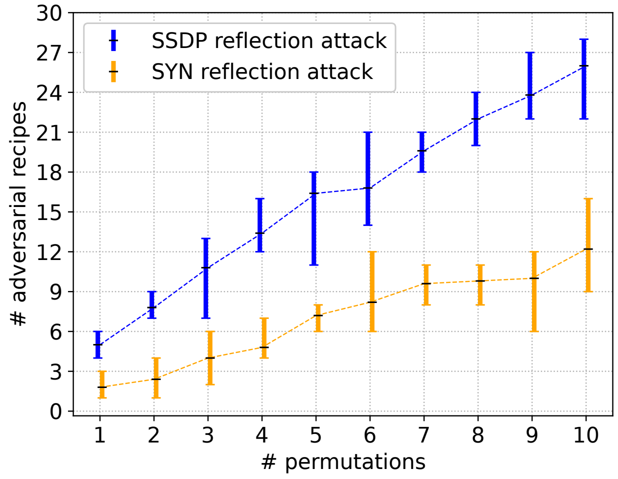


Fig. 7. Number (min, max, avg) of generated adversarial recipes for different permutation counts for attack impact of 1000 packets.

[47] that a model trained on known attacks will fall short in detecting unknown ("zero-day") attacks.

We use the Python Scikit Learn library to train multi-class classifiers by three representative ensemble decision-tree algorithms, namely Random Forest, Gradient Boosting, and AdaBoost, to predict device class label. These models provide us with a classification score used as a baseline threshold for detecting misbehavior. Table III shows the accuracy and false positive rate of each model. For detecting misbehaviors, we use classification scores' μ and σ for each class of IoT device (shown in Table IV for Random Forest model) obtained from the training dataset in such a way that if for a given instance the model's classification score for its predicted label is below $\mu - \sigma$ (of that label), the instance is considered as malicious. Table IV shows the Random Forest model's classification score for each class of IoT devices in our testbed obtained from the training dataset. We use μ and σ values obtained from the training for detecting IoT devices' misbehavior in such a way that if for a given instance the model's classification score for its predicted label is below $\mu - \sigma$ (of that label), the instance is considered as malicious.

B. AdIoTack Offline Instances

In this part, we evaluate the performance of AdIoTack in offline adversarial learning with a view to launch two well-known network-based attacks, namely TCP SYN and SSDP reflection attacks.

Let us begin by showing that our offline learning method

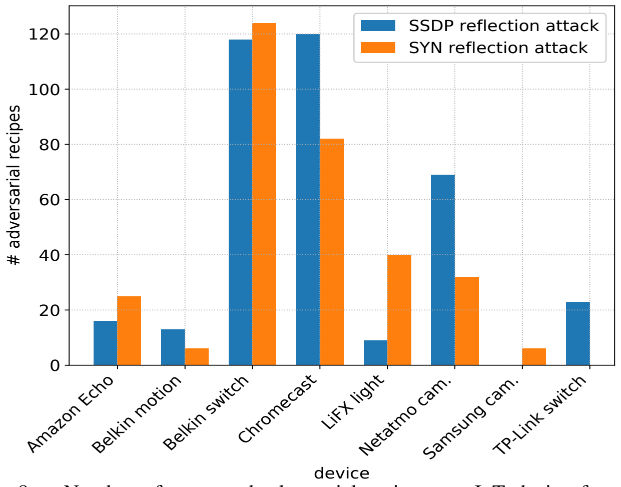


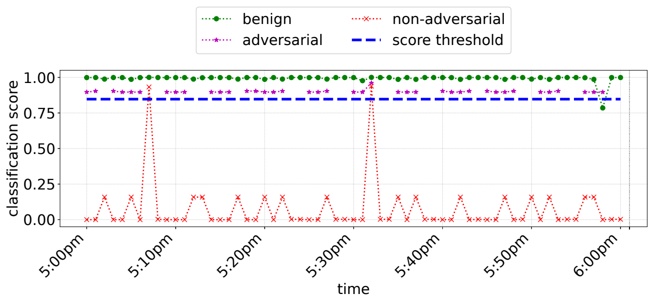
Fig. 8. Number of generated adversarial recipes per IoT device for two representative attacks (result of a 20-permutation run).

is generalizable by applying it to the three popular ensemble models. Fig. 6 indicates the number of adversarial recipes for SSDP and SYN reflection attacks per inference model. Random Forest seems to be more robust to adversarial attacks compared to the other two models, corroborating with the observations in [24]. Because of its relative robustness, we choose Random Forest for the rest of the experimental evaluation in this paper.

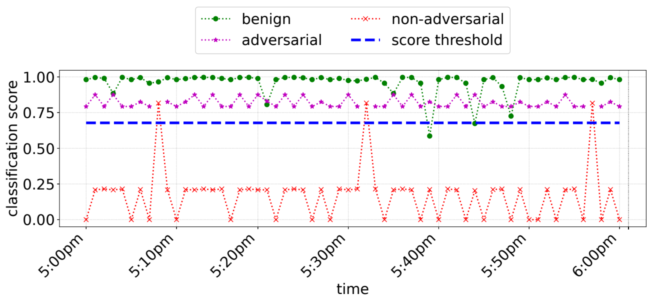
We now quantify the impact of the sequence order of trees in generating adversarial recipes. We define permutation as a random shuffle of the trees inside the ensemble model, which we need to iterate through them in the same order as the permutation suggests. We show that various permutations can result in different adversarial recipes. Fig. 7 shows the min, max, and average number of unique recipes for different permutation counts by running the algorithm five times for each permutation value. For example, if the permutation count is three, we run the offline learning with three different permutations for each device over five rounds and count the number of unique recipes at each round. The plot shows an overall increasing trend which means as we try different permutations, there is a high chance of finding new recipes. Also, the plot suggests that SSDP reflection can generate more recipes than SYN reflection, in which the increasing trend seems to become smoother after three permutations. The average time taken for generating adversarial recipes per permutation is about 1.5 minutes for SSDP reflection and 2.2 minutes for SYN reflection attack. This is probably because four features (packet and byte count of flows $\downarrow$WAN and $\uparrow$WAN) are affected by SYN reflection attack, whereas only two features (packet and byte count for $\uparrow$SSDP) are affected by the SSDP reflection attack.

Fig. 8 illustrates the number of generated recipes for each IoT device in our testbed by running the offline learning algorithm with 20 permutations. Belkin switch and Chromecast in both types the attacks. Samsung smart camera has no recipe for SSDP reflection, TP-Link has no recipe for SYN reflection attack, and Hue lightbulb has no recipe for both of the attacks, indicating tight constraints for those classes.

Before experimenting with our adversarial recipes in an operational network of IoT devices, we evaluate the performance of our inference model and efficacy of offline adversarial instances by replaying various unseen datasets in three scenarios,



(a) Amazon Echo.



(b) Belkin switch.

Fig. 9. Time trace of model performance against replayed instances of two representative IoT devices: (a) Amazon Echo, and (b) Belkin switch, in three scenarios: benign (green), synthetic adversarial SYN reflection attack (purple) and synthetic non-adversarial SYN reflection attack (red).

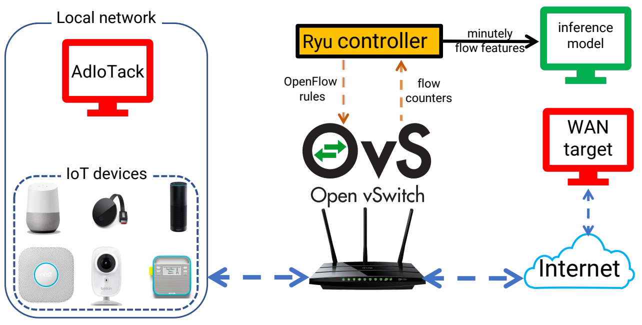


Fig. 10. Prototype of AdIoTack.

namely: (1) a purely benign dataset is expected to receive high classification scores from the model, (2) syntactically changed features of the benign instances (to represent non-adversarial SYN reflection attack) are expected to receive low scores from the model (highlighting a detection), and (3) synthetically changed features of the benign instances (to represent adversarial SYN reflection attack) are expected to receive high scores from the model (highlighting a miss).

Fig. 9 shows the results for two representative IoT devices over an hour. As expected, almost all benign instances are classified with a high score (*i.e.,* above the threshold shown by dashed blue line), highlighting the expected performance under normal situations (benign traffic). The dotted red curve shows the model's score for non-adversarial attack instances, which the model detects a majority of them (receiving scores below the threshold). Finally, the purple curve highlights the model's inability to detect adversarial attacks (*i.e.,* giving scores above the threshold, meaning benign). In some epochs like at 5:03pm for Amazon Echo and 5:08pm for Belkin switch, there is no recipe in the feasible region to project the current instance, thus missing data points.

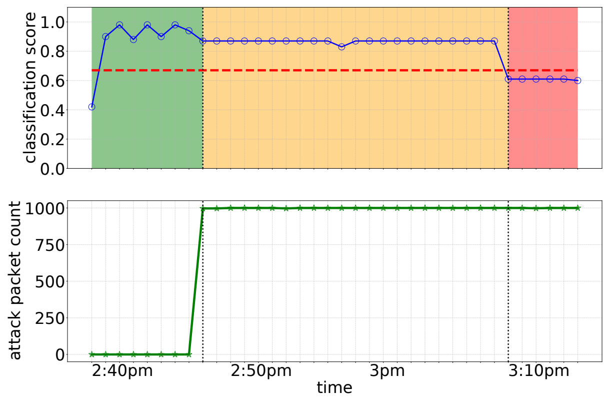
C. AdIoTack Online Evaluation

To demonstrate adversarial attacks in a realistic scenario, we deploy our inference model in a system that can classify traffic in real-time. It receives a stream of flow-based telemetry and predicts traffic features of individual connected devices every minute. To collect real-time telemetry, we use an SDN-enabled home gateway in our setup, shown in Fig. 10. We installed Open Virtual Switch (OVS) on the gateway of our IoT network. Ryu, an open-source SDN controller, is installed on another machine that communicates with this OVS. The Ryu controller initializes the setup by sending a fixed set of flow rules specific to each IoT device on the network, of which the inference model needs the corresponding flow counters as features (features described in §V-A). Every minute, the model receives flow counters from the controller and classifies individual devices. The model also detects unexpected behavior by way of classification score (explained in §V-A).

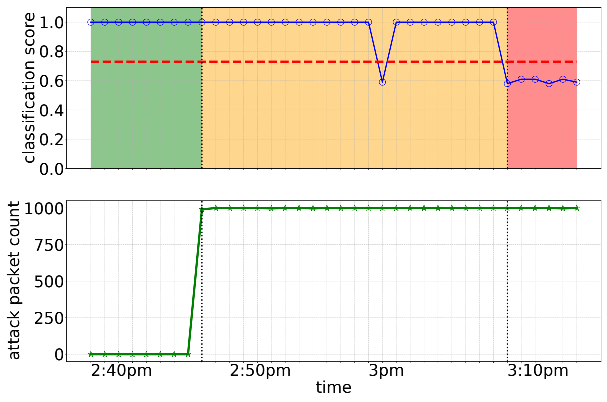
Fig. 11 shows the performance of AdIoTack in the online execution phase for three representative devices with SYN reflection attack. Our entire evaluation lasted for 35 minutes with three stages: between 2:40pm and 2:48pm (highlighted in green), no attack was executed to observe the behavior of our inference model in normal operation; between 2:48pm and 3:10pm (highlighted in orange), we performed adversarial SYN reflection attacks in each epoch; lastly, between 3:10pm and 3:15pm (highlighted in red), we performed non-adversarial SYN reflection attack with the same impact of the adversarial ones launched in the middle stage. The upper subplots show the inference model's classification score at the end of each epoch. The score threshold of each class is shown by a constant dashed red line. The lower subplots show the number of attack packets received by the ultimate target machine (victim server) residing on the WAN interface of the gateway (outside the local IoT network).

It can be seen that the inference model displays an acceptable performance under normal operation (*i.e.,* pure benign traffic, without any attack). Moving to the orange regions, we observe that all adversarial reflection attacks are successful (classification scores above the threshold), except during an epoch (at 3:01 pm) for Chromecast in Fig. 11(b) – this sudden drop in the score was due to a short network disruption that caused packet loss in the traffic sent by AdIoTack. Lastly, focusing on the red regions, the model gives scores lower than the expected threshold to non-adversarial instances across the three representative devices.

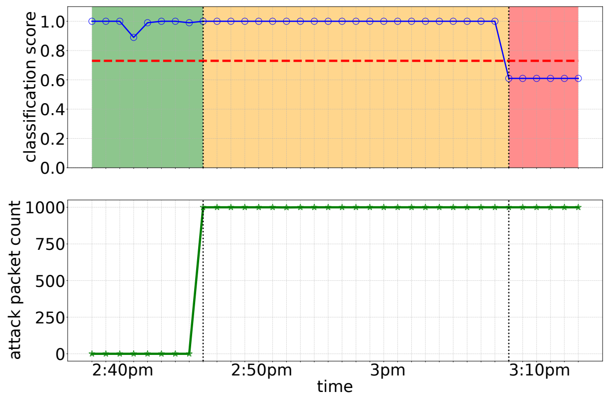
During the online evaluation, we found that it is impossible to execute any adversarial attack on three devices, namely Samsung smart camera, Philips Hue lightbulb, and Amazon Echo. For Samsung smart camera, the reason is that no feasible adversarial recipe was found for I_c (device's current feature vector) to be projected on as I_c during our experiment violated the upper bound of the rule on $\uparrow$`SSDP` feature. For the other two



(a) Belkin switch.



(b) Chromecast.



(c) Netatmo camera.

Fig. 11. Time trace of model performance and attack traffic on three IoT devices in three scenarios: normal operation (green), adversarial attack (orange) and non-adversarial attack (red).

devices, the reason relies on the network aspect. Philips Hue lightbulb responds to the corrective ping message sourced by AdIoTack. However, the response does not revert the gateway's MAC table; probably because Philips Hue is not a standalone device and relies on a separate bridge for communication. Amazon Echo did not respond to the ping message, hence the disrupted MAC table issue stopped us from launching the attack.

During this evaluation, we assumed that AdIoTack knows the inference cycles of the model *i.e.,* when the model fetches the flow counters from the SDN controller and classifies the traffic. This is a valid assumption as we already declared our work to be a white-box approach. Still, we further evaluate AdIoTack online execution in scenarios where inference cycles are unknown. Fig. 12 illustrates this evaluation for Belkin motion sensor and Chromecast. We launched adversarial attacks on these devices with four different timings. Time-shift of T means that the attacker is T seconds ahead of the inference model. We need to note that the time window in this evaluation is 60 seconds meaning that the model becomes online every 60 seconds.

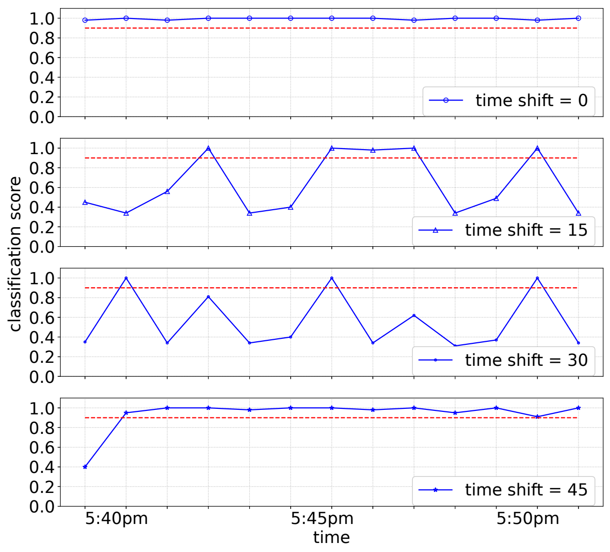
Unsurprisingly, Fig. 12 shows that adversarial attacks with the time shift of zero are 100% successful in bypassing the model. Fig. 12 shows that for Chromecast, time-shift almost has no effect on the success of adversarial attacks; however, for Belkin motion sensor, it has significant effects. The results suggest that a time shift of 30 seconds leads to the minimum success rate (23%), but when it gets closer to the inference model's time cycle, *i.e.,* time shift of 45 seconds, the success rate significantly rises (92%). The key takeaway is that regardless of when AdIoTack launches the attack, there is always a chance to bypass the model.

VI. Refining Resilience of Decision Trees Against Adversarial Volumetric Attacks

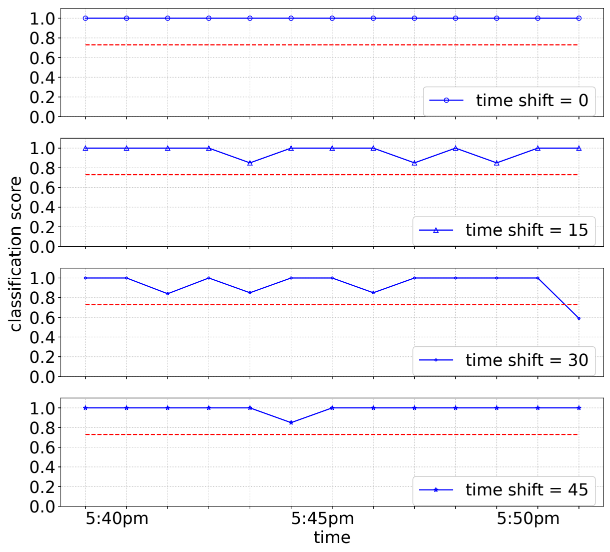
We have so far highlighted the vulnerability of decision tree-based models against systematically crafted adversarial attacks. In this section, we develop a method called *patching* to refine the model post-training, making decision trees robust against adversarial volumetric attacks.

Decision trees trained purely by benign traffic instances are inherently vulnerable (given their structure) to volumetric attacks. Fig. 13 illustrates an illustrative decision tree with four leaves and three decision nodes on three features. For each leaf, we define *bounded* and *unbounded* feature sets. For a given leaf l, its bounded feature set includes feature f, if inside the path from the tree's root to l, there is at least one decision node on f which must take the left branch (*i.e.,* $f \leq \tau$) to reach l. Otherwise, if there is no such a decision node for f, we consider f as an unbounded feature for l. For example, in Fig. 13, consider the leaf with label '`C`'. To reach this leaf from the root, the following conditions must be met: $f_1 > 100$ and $f_3 \leq 70$. Thus if an instance has $f_1 = +\infty$, $f_2 = +\infty$, and $f_3 = 70$, it still reaches the leaf with no issue as there is no upper boundary check for f_1 and there is no upper or lower boundary check for f_2 at all.

In the literature, there exist methods [9], [6] for developing

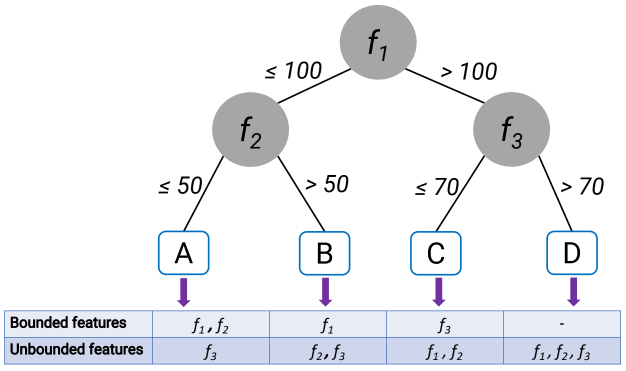


(a) Belkin motion sensor.



(b) Chromecast.

Fig. 12. Effect of time-shift for two IoT devices.



Bounded features	f_1, f_2	f_1	f_3	-
Unbounded features	f_3	f_2, f_3	f_1, f_2	f_1, f_2, f_3

Fig. 13. Bounded and unbounded features for each leaf in a decision tree.

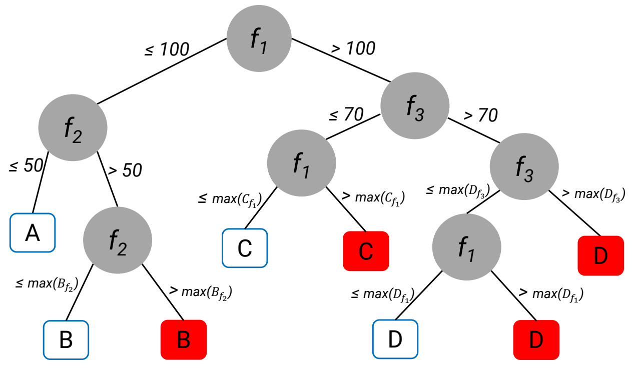


Fig. 14. Essential patching performed on individual leaves of the vulnerable decision tree from Fig. 13.

robust decision trees against adversarial attacks. The current techniques manipulate the training process of decision trees by changing the original model to make sure if an adversarial attack happens, the model still gives the correct output label which is desirable in image classification which requires a given set of adversarial instances to be provided before refining the model *i.e.,* the model still might be vulnerable to another set of adversarial instances. That way, they sacrifice the prediction accuracy for benign instances, in order to make decision trees robust in adversarial scenarios. Also, given their primary focus is on image recognition, they do not have a notion of volumetric attacks, and thus, unbounded features remain loose. Our method, however, focuses on a different problem (*i.e.,* volumetric cyberattacks), and aims to refine a trained model. Therefore, it does not interfere with the training process and does not require re-training. Our method receives a trained decision tree-based model and then patches the unbounded features with no dependency on any adversarial recipes. If the training dataset gives a correct representation of devices' traffic characteristics by at least capturing the maximum traffic volume for each class correctly, the patched model will have the exact same accuracy/false positive over benign instances as the original model. However, if there are data instances in the testing dataset that have higher traffic features' value than the limits which were captured from the training dataset, our method would flag them an as malicious (*i.e.,* higher false positives than the original unpatched model).

We define a patch for feature f on leaf l as follows:

$$patch_{l,f} = f \leq max(l_f)$$

This patch can be seen as a new decision node added just before reaching the leaf, that checks the upper bound of f for l. The upper bound is calculated based on the training dataset used for training the original model. As an example, based on our training dataset, Amazon Echo sends a maximum of four DNS packets every minute; thus, $max(Amazon\ Echo_{\uparrow DNS\ pkt}) = 4$. Any instance x reaching leaf l, which $x_f > max(l_f)$, goes to a new leaf that yields x as a malicious instance.

There are two noteworthy points: (1) Because IoT devices send/receive a low volume of traffic and have repetitive behavioral patterns, individual devices' maximum traffic volume can be captured [46]. However, this could be challenging for personal computers and smartphones which their traffic volume highly depends on users' activities. (2) Our method reduces the impact of adversarial volumetric attacks significantly. However, low-impact attacks (*i.e.,* within the normal behavior range of IoT devices) can still bypass the model.

There are two scenarios where feature f can become

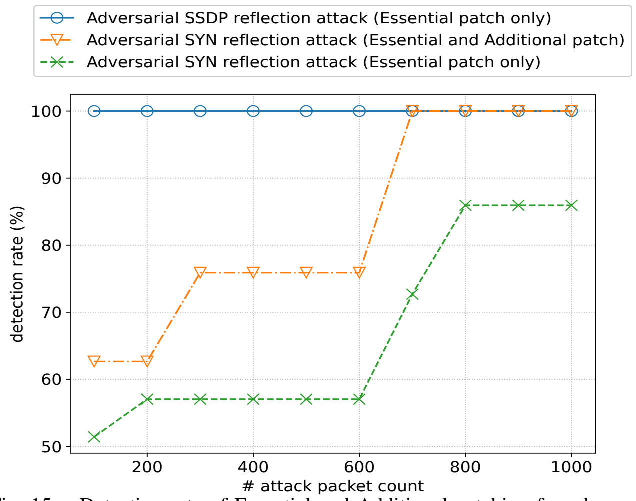


Fig. 15. Detection rate of Essential and Additional patching for adversarial SYN and SSDP reflection attacks.

unbounded for leaf l: (1) When there is a decision node on f, but its right branch (*i.e.,* $f > \tau$) is taken on the path from the tree's root to l (*e.g.,* f_1 for leaf '`C`' in Fig. 13); and (2) When there is no decision node on f through the path at all (*e.g.,* f_2 for leaf '`C`' in Fig. 13). We develop two patching techniques to address these two scenarios. *Essential Patching* solves the first problem by traversing a given vulnerable decision tree and patching leaves with unbounded features through paths from the root. Fig. 14 shows the result of essential patching performed on the tree from Fig. 13. Adversarial instances will be routed to the red leaves, which results in true detection of the attack.

Although essential patching is necessary to fix some unbounded features, it does not cover the second scenario where some features do not even exist on some paths. For example, leaf "`A`" in Fig. 14, still can be targeted with volumetric attack on $f_3 = +\infty$. In fact, some adversarial attacks can be captured solely with essential patching (*e.g.,* SSDP reflection attack in Fig. 15). To overcome this problem, we define *Additional Patching* which patches all leaves for a given feature. This way, regardless of the trees' structure, we make sure all leaves are patched against adversarial volumetric attacks on the desired features. Additional patching can be done in different ways; for example, one may choose to patch all leaves for all possible features to create maximum robustness, making the model complex and slow. Another one may use expert knowledge to select specific vulnerable features to limit the complexity of the model while making it robust against attacks over those features that are more likely to be targeted by attackers. After applying the essential patching to our original model, we quantify the accuracy and false-positive over the testing dataset and they are 96% and 0.09%, respectively, which are exactly the same as the original model.

Fig. 15 shows the performance of our patching methods over adversarial SSDP and SYN reflection attacks with a range of attack impacts (100 to 1,000 packet count). Note that these attack instances are created based on adversarial recipes, bypassing the unpatched model, as shown in the previous section. Essential patching is sufficient to detect all SSDP reflection attacks while detecting 86% of the high-impact SYN reflection attacks. Having an additional patch over $\uparrow$ `WAN` packet count feature increases the detection rate for low, medium, and high impact attacks; and detects all attacks with an impact greater than 636 packets. The average impact of SYN reflection attacks that can still bypass the model is 117 packets across our IoT devices, with the maximum impact of 636 packets for Chromecast. Compared to the vulnerable model, which has no limit over the attack packet count (*i.e.,* unlimited impact), this impact shows a significant improvement in terms of robustness.

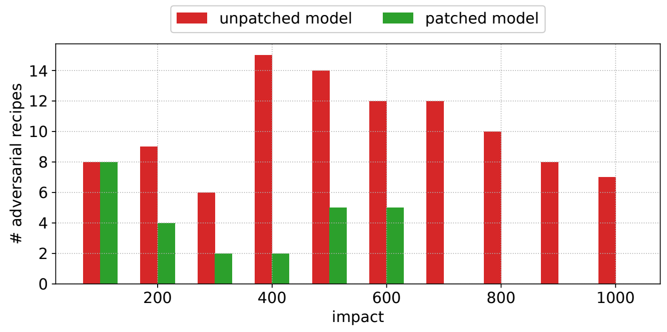


Fig. 16. Number of recipes generated for SYN reflection attack before and after patching the model.

Finally, we validate the efficacy of our patched model by re-running the adversarial offline learning function on the refined model. Fig. 16 compares the number of generated recipes in the patched model versus the original (unpatched) model. The patched model (green bars) has become more robust against SYN reflection attacks with any impact above 200 attack packets and the improvement becomes more evident as the impact increases. More importantly, no recipe exists for volumetric attacks with an impact above 600 packets.

A. Impact of Hyper-Parameters

Decision tree ensemble models come with two key hyper-parameters, namely the number of trees and the maximum depth of each tree. These parameters can be tuned to gain the best accuracy results. One may wonder about their role in the model resilience against adversarial attacks. It seems more decision trees or deeper trees can introduce additional constraints to the process of generating adversarial recipes, thus, potentially limiting those recipes. Let us investigate the impact of these two parameters on adversarial recipes. We first train six different models with 50, 100, 150, 200, 250, and 300 trees (all share a depth of 10) and thereafter generate adversarial recipes for each of these models. Similarly, we train seven different models (with 100 trees), varying the depth from 8 to 20 in steps of 2 units.

Our experiments revealed that the number of decision trees has almost no significant impact on adversarial recipes. In other words, incorporating more trees in an ensemble model does not necessarily make the model robust against adversarial attacks. However, increasing the depth of trees would noticeably limit the recipes. For example, for SSDP reflection attacks, the number of recipes generated for the seven models are 37, 33, 9, 4, 2, 1, and 0 as the max depth increases from 8 to 20, respectively. We observe similar patterns in SYN reflection attacks.

Different impacts of these two hyper-parameters could be attributed to the "bootstrap" technique used by ensemble algorithms. This technique helps to decrease the variance of model accuracy for unseen data (testing phase) by training each decision tree with a random sub-sample of the training dataset [5]. Therefore, in a given decision tree, as it gets deeper, new constraints apply to the same sub-sample space, resulting in smaller recipe regions, increasing the chance for inconsistency with other constraints in the recipe. Moving to a new decision tree, on the other hand, is built on a different sub-sample of the training dataset compared to other trees. Therefore, its constraints are likely to be on a different scope, more likely not to violate the ones found in other trees.

VII. Conclusion

In this paper, we developed *AdIoTack*, a systematic way of quantifying and refining the resilience of decision tree ensemble models against data-driven adversarial attacks. We first developed a white-box algorithm that automatically generates recipes of volumetric network-based attacks that can bypass the inference model unnoticed. Our algorithm takes the intended attack on a victim class and the model as inputs. We developed a systematic method to successfully launch the intended attack on real networks. We next prototyped AdIoTack and validated its efficacy on a real testbed of real IoT devices monitored by a trained Random-Forest model. We demonstrated how the model detects all non-adversarial volumetric attacks on IoT devices while missing many adversarial ones. Finally, we developed systematic methods to patch loopholes in trained decision tree ensemble models. We demonstrated how our refined model detects 92% of adversarial volumetric attacks.

References

[1] 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*, Dallas, USA, Jul 2019.

[2] J. Anand, A. Sivanathan, A. Hamza, and H. H. Gharakheili, "PARVP: Passively Assessing Risk of Vulnerable Passwords for HTTP Authentication in Networked Cameras," in *Proc. ACM Workshop on Descriptive Approaches to IoT Security, Network, and Application Configuration (DAI-SNAC)*, Virtual Event, Germany, Dec 2021.

[3] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar, "The Security of Machine Learning," *Springer Machine Learning*, vol. 81, p. 121–148, May 2010.

[4] L. Breiman, "Random Forests," *Springer Machine learning*, vol. 45, no. 1, pp. 5–32, Oct 2001.

[5] ——, "Bagging Predictors," *Springer Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[6] S. Calzavara, C. Lucchese, G. Tolomei, S. A. Abebe, and S. Orlando, "Treant: Training Evasion-aware Decision Trees," *Data Mining and Knowledge Discovery*, vol. 34, no. 5, pp. 1390–1420, 2020.

[7] G. Caminero, M. L. Martín, and B. Carro, "Adversarial Environment Reinforcement Learning Algorithm for Intrusion Detection," *Computer Networks*, vol. 159, pp. 96–109, Aug 2019.

[8] N. Carlini and D. Wagner, "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods," in *Proc. ACM AISec*, Dallas, USA, Nov 2017.

[9] H. Chen, H. Zhang, D. Boning, and C. Hsieh, "Robust Decision Trees Against Adversarial Examples," in *Proc. ICML*, Long Beach, CA, USA, Jun 2019.

[10] M. Cheng, T. Le, P.-Y. Chen, J. Yi, H. Zhang, and C.-J. Hsieh, "Query-Efficient Hard-label Black-box Attack:An Optimization-based Approach," in *Proc. of ICLR*, Vancouver, Canada, Apr 2018.

[11] F. Croce and M. Hein, "Reliable Evaluation of Adversarial Robustness with an Ensemble of Diverse Parameter-free Attacks," *arXiv e-prints*, p. arXiv:2003.01690, Mar 2020.

[12] CSO, "University attacked by its own vending machines, smart light bulbs & 5,000 IoT devices," Feb 2017. [Online]. Available: https://www.csoonline.com/article/3168763/university-attacked-by-its-own-vending-machines-smart-light-bulbs-and-5-000-iot-devices.html

[13] Cyberedge, "Cyberthreat Defense Report," https://cyber-edge.com/wp-content/uploads/2020/03/CyberEdge-2020-CDR-Report-v1.0.pdf, 2020.

[14] G. S. Dhillon, K. Azizzadenesheli, Z. C. Lipton, J. Bernstein, J. Kossaifi, A. Khanna, and A. Anandkumar, "Stochastic Activation Pruning for Robust Adversarial Defense," *arXiv e-prints*, p. arXiv:1803.01442, Mar 2018.

[15] Y. Ding, L. Wang, H. Zhang, J. Yi, D. Fan, and B. Gong, "Defending Against Adversarial Attacks Using Random Forests," in *Proc. CVPR*, Long Beach, CA, USA, Jun 2019.

[16] R. Doshi, N. Apthorpe, and N. Feamster, "Machine Learning DDoS Detection for Consumer Internet of Things Devices," in *Proc. IEEE S&P Workshops*, San Francisco, CA, USA, May 2018.

[17] A. Ferdowsi and W. Saad, "Generative Adversarial Networks for Distributed Intrusion Detection in the Internet of Things," in *Proc. IEEE GLOBECOM*, Waikoloa, USA, Dec 2019.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *arXiv e-prints*, p. arXiv:1412.6572, Dec 2014.

[19] A. Hamza, H. Habibi Gharakheili, T. A. Benson, and V. Sivaraman, "Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity," in *Proc. ACM SOSR*, CA, USA, Apr 2019.

[20] A. Hamza, D. Ranathunga, H. Habibi Gharakheili, T. A. Benson, M. Roughan, and V. Sivaraman, "Verifying and Monitoring IoTs Network Behavior using MUD Profiles," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2020.

[21] O. Ibitoye, O. Shafiq, and A. Matrawy, "Analyzing Adversarial Attacks Against Deep Learning for Intrusion Detection in IoT Networks," in *Proc. IEEE GLOBECOM*, Waikoloa, USA, Dec 2019.

[22] IBM X-Force Research, "The weaponization of IoT devices," 2017. [Online]. Available: https://www.ibm.com/downloads/cas/6MLEALKV

[23] A. Kantchelian, J. D. Tygar, and A. D. Joseph, "Evasion and Hardening of Tree Ensemble Classifiers," in *Proc. ICML*, NY, USA, Jun 2016.

[24] Z. Katzir and Y. Elovici, "Quantifying the Resilience of Machine Learning Classifiers Used for Cyber Security," *Expert Systems with Applications*, vol. 92, pp. 419–429, Feb 2018.

[25] T. M. Khoshgoftaar, M. Golawala, and J. V. Hulse, "An Empirical Study of Learning from Imbalanced Data Using Random Forest," in *Proc. IEEE ICTAI*, Patras, Greece, Oct 2007.

[26] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Hell of a Handshake: Abusing TCP for Reflective Amplification DDoS Attacks," in *Proc. USENIX WOOT*, San Diego, USA, Aug. 2014.

[27] M. Labs, "McAfee Labs 2019 Threats Predictions Report," https://www.mcafee.com/blogs/other-blogs/mcafee-labs/mcafee-labs-2019-threats-predictions/, 2018.

[28] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[29] T. S. Lim, W. Y. Loh, and Y. S. Shih, "A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms," *Springer Machine Learning*, vol. 40, no. 3, pp. 203–228, 2000.

[30] M. Lyu, D. Sherratt, A. Sivanathan, H. H. Gharakheili, A. Radford, and V. Sivaraman, "Quantifying the Reflective DDoS Attack Capability of Household IoT Devices," in *Proc. ACM WiSec*, Boston, USA, Jul 2017.

[31] C. Magazine, "New Vulnerability Allows DDoS Attack and Data Exfiltration on Billions of Devices," Jun 2020. [Online]. Available: https://www.cpomagazine.com/cyber-security/new-vulnerability-allows-ddos-attack-and-data-exfiltration-on-billions-of-devices/

[32] H. Mahmudul, M. M. Islam, M. I. I. Zarif, and M. M. A. Hashem, "Attack and Anomaly Detection in IoT Sensors in IoT Sites Using Machine Learning Approaches," *Elsevier Internet of Things*, vol. 7, p. 100059, Sep 2019.

[33] McAfee, "Grand Theft Data." [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-data-exfiltration.pdf

[34] M. Miettinen, M. S, I. Hafeez, T. Frassetto, N. Asokan, A. R. Sadeghi, and S. Tarkoma, "IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT," in *Proc. IEEE ICDCS*, Atlanta, USA, June 2017.

[35] NETSCOUT Security, "A Deeper Look at IoT Weaponization," 2020. [Online]. Available: https://bit.ly/3pr3NJT

[36] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A. Sadeghi, "DÏoT: A Federated Self-learning Anomaly Detection System for IoT," in *Proc. IEEE ICDCS*, Dallas, USA, Jul 2019.

[37] Nokia, "Threat Intelligence Report 2020," *Computer Fraud & Security*, vol. 2020, no. 11, 2020.

[38] OPTIV, "Attempted Florida Water Supply Tampering Underscores IoT/OT Security Challenges," Feb 2021. [Online]. Available: https://www.optiv.com/explore-optiv-insights/blog/attempted-florida-water-supply-tampering-underscores-iotot-security

[39] Palo Alto Networks, "Unit42 IoT Threat Report," Apr 2020. [Online]. Available: https://start.paloaltonetworks.com/unit-42-iot-threat-report

[40] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," in *Proc. IEEE EuroS&P*, Saarbrücken, Germany, Mar 2016.

[41] A. Pashamokhtari, N. Okui, Y. Miyake, M. Nakahara, and H. Habibi Gharakheili, "Inferring Connected IoT Devices from IPFIX Records in Residential ISP Networks," in *Proc. IEEE LCN*, Virtual Event, Canada, Oct 2021.

[42] Y. E. Sagduyu, Y. Shi, and T. Erpek, "IoT Network Security from the Perspective of Adversarial Deep Learning," in *Proc. IEEE SECON*, Boston, USA, Jun 2019.

[43] A. Sivanathan, H. H. Gharakheili, and V. Sivaraman, "Managing IoT Cyber-Security using Programmable Telemetry and Machine Learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 60–74, 2020.

[44] A. Sivanathan, H. Habibi Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," *IEEE TMC*, vol. 18, no. 8, pp. 1745–1759, Aug 2019.

[45] A. Sivanathan, H. Habibi Gharakheili, and V. Sivaraman, "Detecting Behavioral Change of IoT Devices using Clustering-Based Network Traffic Modeling," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7295–7309, Aug 2020.

[46] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and Classifying IoT Traffic in Smart Cities and Campuses," in *Proc. IEEE INFOCOM Workshops*, Atlanta, USA, May 2017.

[47] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *Proc. IEEE S&P*, Oakland, USA, May 2010.

[48] M. Taghavi and M. Shoaran, "Hardware Complexity Analysis of Deep Neural Networks and Decision Tree Ensembles for Real-time Neural Data Classification," in *Proc. IEEE NER*, San Francisco, USA, Mar 2019.

[49] Verimatrix, "IoT security for today's connected world." [Online]. Available: https://www.verimatrix.com/markets/internet-of-things

[50] P. Vähäkainu, M. Lehto, and A. Kariluoto, "IoT-based Adversarial Attack's Effect on Cloud Data Platform Service in Smart Building's Context," in *Proc. ICCWS*, Norfolk, USA, Mar 2020.

[51] H. Wang and C. Yu, "A Direct Approach to Robust Deep Learning Using Adversarial Networks," *arXiv e-prints*, p. arXiv:1905.09591, May 2019.

[52] C. Xie, Y. Wu, L. Maaten, A. L. Yuille, and K. He, "Feature Denoising for Improving Adversarial Robustness," in *Proc. IEEE CVPR*, CA, USA, Jun 2019.

[53] ZDNet, "Ransomware attack halts production at IoT maker Sierra Wireless," Mar 2021. [Online]. Available: https://www.zdnet.com/article/ransomware-attack-halts-production-at-iot-maker-sierra-wireless/

[54] C. Zhang, H. Zhang, and C.-J. Hsieh, "An Efficient Adversarial Attack for Tree Ensembles," in *Proc. NeurIPS*, Virtual, Dec 2020.