Brittle Features of Device Authentication

Washington Garcia  
w.garcia@ufl.edu  
University of Florida  
Gainesville, Florida, USA

Animesh Chhotaray  
chho58@ufl.edu  
University of Florida  
Gainesville, Florida, USA

Joseph I. Choi  
chojoseph007@ufl.edu  
University of Florida  
Gainesville, Florida, USA

Suman Kalyan Adari  
sadari@ufl.edu  
University of Florida  
Gainesville, Florida, USA

Kevin R.B. Butler  
butler@ufl.edu  
University of Florida  
Gainesville, Florida, USA

Somesh Jha  
jha@cs.wisc.edu  
University of Wisconsin-Madison  
Madison, Wisconsin, USA

ABSTRACT
Authenticating a networked device relies on identifying its unique characteristics. Recent device fingerprinting proposals demonstrate that device activity, such as network traffic, can be used to extract features which identify devices using machine learning (ML). However, there has been little work examining how adversarial machine learning can compromise these schemes. In this work, we show two efficient attacks against three ML-based device authentication (MDA) systems. One of the attacks is an adaptation of an existing gradient-estimation-based attack to the MDA setting; the second uses a fuzzing-based approach. We find that the MDA systems use brittle features for device identification and hence, can be reliably fooled with only 30 to 80 failed authentication attempts. However, selecting features that are robust against adversarial attack is challenging, as indicators such as information gain are not reflective of the features that adversaries most profitably attack. We demonstrate that it is possible to defend MDA systems which rely on neural networks, and in the general case, offer targeted advice for designing more robust MDA systems.

CSCS CONCEPTS
• Security and privacy → Authentication; • Computing methodologies → Machine learning approaches.

KEYWORDS
device fingerprinting, device authentication, adversarial machine learning, traffic analysis

1 INTRODUCTION
Authentication is a classically difficult systems security problem, but is integral to the establishment of identity and ultimately trust in computer networks. In the era of the Internet of Things, which has seen an explosion in the number of wireless communication-aware devices, the problem of device authentication has become increasingly urgent. In these environments, it is advantageous to identify devices without extra modifications to their storage or communication requirements. Thus, methods adapted to leverage characteristics of the devices have been a focus of recent research (e.g., identifying a device based on the characteristics of its network traffic alone [12]). To facilitate this goal, machine learning (ML) has become a means of establishing a device’s identity [4, 11, 40].

In an ML-based device authentication (MDA) system, an underlying ML model maps a set of device signatures to a set of credentials. However, there has been little consideration to date for the robustness of the underlying models in device authentication settings, particularly against iteratively-crafted adversarial samples. Due to different requirements and threat models, the device fingerprinting literature contains many implementations and signature extraction methods that span the hardware and software stack. Likewise, there is not yet a unified standard for performing attestation of network devices using signatures. Our goal is to investigate the feasibility of model-based device fingerprinting as a medium for performing the device authentication task.

The field of adversarial machine learning (AML) has uncovered several milestone attacks against state-of-the-art ML models [18, 37, 38]. A limitation of these proposed attacks is the context in which they are built; their explicit goal is breaking classification tasks, rather than authentication tasks. There are two key differences:

1. Information returned from an authentication system is limited. In authentication tasks, class-level information is limited, and feedback from the model is essentially non-existent apart from the final authentication decision. During authentication, a user may attest by providing a credential-signature pair, but the authenticity method’s response is always either YES or NO. This is a much harder and more realistic case than assuming the system will return annotated labels—artifacts that are typically available in classification tasks [38].

2. Target information is secret and hidden. In prior ML attacks against binary classification models, an adversary may perturb the correctly-classified features of a known victim, and force it to be mislabeled [13], or generate new samples based on an already known set of victim samples [34]. However, in systems that use
2 BACKGROUND

We provide a brief overview of the enabling techniques behind device authentication, namely device fingerprinting, and adversarial machine learning.

2.1 Device Fingerprinting

Device authentication can be considered a general term for attesting a computational device’s identity against some known signature. Device authentication is primarily enabled by device fingerprinting techniques. The exact method of extracting a device’s fingerprint can vary, but the common goal is to create unique, attestable signatures that can be stored in a database and later queried for comparison. The most direct method of creating a device fingerprint is to leverage the physical manufacturing imperfections exhibited by hardware (i.e., process variation) to distinguish between devices. Bates et al. used USB enumerations between the USB controller hardware and the device operating system to accurately distinguish between devices of same make and model [4]. For wireless devices, techniques have leveraged process variation to accurately identify wireless sensor nodes [12], commodity devices [25, 28, 40], and radio network transmitters [32].

In terms of applications, device fingerprinting has become an enabling technology in the secure Internet of Things (IoT) literature. Although useful for device type identification and network-level access control [31], it has also enabled services such as quarantining [33] and watermarking [17]. Unfortunately, these applications are undermined if an adversary can reliably fool the underlying fingerprinting technique.

2.2 Adversarial Machine Learning

Most attacks against device fingerprinting tasks have been framed in terms of traditional, network-based adversaries, as categorized by Mamdouh, Elrukhshi, and Khattab [30]. Adversarial machine learning has shown that models such as artificial neural networks [46] and tree-based classifiers [27] are vulnerable to straightforward attacks.

For the authentication task, the adversary’s goal is similar to mounting an attack within a limited query model, a setting that has been investigated by Dang et al. under the binary classification task [13]. However, attacks against authentication are particularly challenging given the lack of labeled information returned; this setting is analogous to the restricted query model where no counterexample is provided, as discussed by Angluin [2]. Several prior works have investigated techniques for fooling biometric authentication using the top-k scores from the system [43], or relying on complete knowledge about the victim [34]. Although similar, these works have not considered the restricted-query binary decision setting. This setting is much more realistic, as the adversary has no knowledge about the victim, and must interact with the authentication system to know if they are successful. A tangential result in fooling authentication systems is the work by Hitaj et al. [23], which uses generative models to guess string-based passwords. Contrary to their work, our method targets the signatures emitted by hardware, and leverages design faults of model-based fingerprinting techniques, rather than the low entropy of human-generated passwords.

The most similar work to ours is by Zhao et al. [49], who attack biometric authentication systems in the binary decision setting using randomly generated inputs. In contrast to their approach, our analysis is performed in the device authentication setting, using a plausible fuzzing-based attack and a gradient estimation attack.
For comparison, we discuss the result of random inputs on MDA systems and visualize the drawbacks of such an approach in Section 6.1.2 and Section 7. In addition, our analysis is grounded in approximations of the model’s decision boundary (XAI) and the model’s adversarial subspace (LID). Notably, LID helps us to quantify the acceptance region effect defined by Zhao et al. [49].

3 SECURITY MODEL OF AN MDA SYSTEM

In a non-adversarial setting, an MDA system involves several principals: an authentication server and multiple (honest) client devices. An MDA system operates in two phases.

1. **Device registration.** Each device is assumed to possess a unique device identification string $id_d$. Also, there exists some pre-defined mechanism that an authentication server can use to collect a set $Y$ of data from device $d$ and generate a signature $sig_d$, a string encoding of $Y$.

   During device registration, the authentication server uses its device interface to collect a set $T$ of $(id, sig)$ pairs for all devices. We refer to the $sig_d$ that is used during device registration as the representative signature of the device $d$. The set $T$ constitutes the training data for the underlying ML model of the MDA system. The registration phase terminates when the training (testing and validation) of the ML model stops.

2. **Device authentication.** We assume that only devices that are registered with the authentication server can query it for authentication. In the device authentication phase, the server uses the pre-defined mechanism to collect a (potentially different) set of data from the querying device $d$, and generates a new signature $sig'_d$. The trained ML model processes the $(id_d, sig'_d)$ pair to output a prediction vector. Given the prediction vector and the $(id_d, sig'_d)$ pair as inputs, the MDA system uses some deterministic checks against certain threshold values to accept/reject the device.

Let us explain the working of an MDA system in a wireless network setting. Here, the network hosts (e.g., computers) act as clients to be authenticated, which can communicate directly with a dedicated authentication server. The clients wish to use resources on the network. As a preliminary challenge step, a host sends traffic over the wireless network to the dedicated server. The server uses this traffic to detect intrinsic timing data. It can process this timing data to generate a succinct fingerprint of the wireless network host. Radhakrishnan et al. show that this method is feasible using inter-arrival times of network packets, which can identify and authenticate of network hosts [40]. During device registration, an authentication server collects representative (host id, timing data) pairs of each host. It uses the collected data to train an underlying ML model. When a registered network device comes online again after some time, it generates a new set of traffic as part of the challenge, which emits a new signature. The trained ML model processes the new timing signature and outputs a prediction. The MDA system processes the prediction, along with the claimed host id, using a hand-crafted heuristic. It returns YES if it is a match, and NO otherwise.

### 3.1 Formalizing an MDA system

Fix integer $t$ denoting the maximum number of times that any device interacts with the authentication server. Fix sets $I$ and $S$ to denote the set of all (string encodings of) valid device identifiers, and the set of all (string encodings of) valid signatures, respectively, for any given MDA system. Let $n = |I|$ denote the number of elements in set $I$. Fix $C$ as the set of functions that encode the mapping $I \times S \rightarrow \mathbb{R}^n$. We define an MDA system $AS$ over sets $(I, S, C)$ as a tuple of algorithms $(SigGen, Train, Auth)$ with the following syntax.

- **SigGen**: The randomized signature generation algorithm $SigGen : I \rightarrow S^t$ takes a device-identification string $id \in I$ and returns a list $L = (sig_1, sig_2, \ldots, sig_t)$ of device signatures, where $\forall i \in [1, t], sig_i \in S$. We assume that $L[1] = sig_1$ is the representative signature of the device and is used by $AS$ for registration; when $j > 1$, $L[j]$ is the signature that will be used by $AS$ to authenticate the device in its $j$-th attempt.

- **Train**: The randomized model training algorithm $Train : (I \times S)^n \rightarrow C$ takes a tuple of $(id, sig)$ pairs of training data and returns a trained model $C \in C$. The model $C$ takes $(id, sig)$ as inputs and returns a prediction vector $P$, where each element $P_{id}$ in $P$ denotes the prediction/confidence value that $id = id$. Since an MDA system registers all devices in the registration phase, we insist $Train$ to take $n$ tuples as input.

- **Auth**: The deterministic device authentication algorithm $Auth : I \times S \times C \rightarrow \{0, 1\}$ takes a $(id, sig)$ pair and a trained model $C$ as inputs and returns a binary decision: 0 indicating "fail" and 1 indicating "pass." The authentication server uses $Auth$ to authenticate any device $(id, sig)$ using $C$ and certain decision heuristics (that are part of the description of $Auth$). Note that $C$ will always be a fixed input to the $Auth$ algorithm.

ML models seldom have 100% accuracy. Therefore, we define $\delta$-correctness of an MDA system as opposed to absolute correctness. In words, a $\delta$-correct $AS$ is one whose $Auth$ algorithm outputs one for valid $(id, sig)$ pair with probability at least $1 - \delta$. Natural examples of $AS$ are $(id, sig)$ pairs (where sig $\notin$ SigGen(id)) for which $Auth$ outputs one with probability at least $1 - \delta$. In practice, $\delta$-correctness can be described using the standard definitions of accuracy, precision, and recall.

### 3.2 Threat model

In the adversarial setting, an adversary knows the underlying signature-generation algorithm of an MDA system $AS$ and gets only decision-level access to $Auth(\cdot, \cdot, C)$, where $C$ denotes a trained model. It is given no other information. Such an adversary is referred to as an untargeted exploratory (UE) adversary by Biggio et al. [5]. Informally, the security goal of $AS$ is to prevent UE adversaries $A$ from "fooling $AS$ by making $Auth(\cdot, \cdot, C)$ output one for some id of an honest user, and some arbitrary signature that the adversary generates (from its own signature).

A standard assumption is that an adversary owns one of the devices that is registered with the (trusted) authentication server. Let $id_A$ be the device identifier and $L_A = SigGen(id_A)$ be the list of signatures of the device that $A$ owns. Given only decision-level answers.
At a very high level, untargeted-exploratory attacks comprise two purposes. The analysis of these settings is an interesting direction for future work.

Let \( \mathcal{L} \) be sampled from \( \{ \text{sig} \} \), \( \text{sig} \) of device identifiers. The hyperparameter \( h \) is responsible for scanning the set of known \( \text{VictimScan}(\text{device identifiers}) \) to find an adversarial sample \( D \) takes as input the list \( \mathcal{L} \). SampleGen algorithms:

- Perturb
- Auth

Let \( \text{VictimScan}(\mathcal{L}) \) use \( \text{SampleGen} \) to craft adversarial samples \( \mathcal{D} \). Let \( \text{VictimScan} \) to scan and select a victim device \( (\text{id}, \text{sig}) \). The robustness of an adversarial sample-crafting algorithm is defined as the probability with which \( A \) fails to fool \( \mathcal{D} \). The failure rate of \( \mathcal{D} \) can be succinctly described in terms of the false acceptance rate, or false positive rate.

### 4 Fooling MDA Systems

Our central hypothesis is as follows: MDA systems can be reliably fooled due to the mis-characterization of adversarial capabilities. From a design standpoint, MDA systems should have built-in robustness against UE adversaries which can provide perturbed samples to the underlying fingerprinting technique. As noted previously in Section 2, recent work in device fingerprinting has considered an adversarial machine learning style of attack, which can iteratively modify a sample based on the system’s feedback. We describe a framework for such attacks in the following text.

#### 4.1 Attack Framework

Let \( \text{id}_A \) denote the identifier of the device that the adversary controls. Let \( \mathcal{L} = \text{SigGen}(\text{id}_A) \), and \( \text{Perturb} \) be some randomized procedure for generating a perturbed signature from an input signature. At a very high level, untargeted-exploratory attacks comprise two algorithms: \( \text{SampleGen} \) and \( \text{VictimScan} \). The \( \text{SampleGen} \) algorithm takes as input the list \( \mathcal{L} \) of device signatures and returns a set of adversarial samples \( \mathcal{D}_A := \{ \text{sig}_A^i : i \in [0, \sigma], \text{sig}_A^i \in \mathcal{S} \} \). The \( \text{VictimScan} \) algorithm is responsible for scanning the set of known device identifiers to find an \((\text{id}_A, \text{sig}_A^*)\) pair that fools \( \mathcal{D} \). The behavior of \( \text{VictimScan} \) can vary based on the implementation, as it determines how potential victims are sampled from the known set of device identifiers. The hyperparameter \( \sigma \) acts as an upper bound on the number of attempts. Here, \( \text{sig}_A^* \) is sampled from \( \mathcal{D}_A; \text{id}_A^* \) is sampled from \( \mathcal{L} \setminus \{ \text{id}_A \} \). Both algorithms get decision-level access to \( \text{Auth} \). See Figure 1 for a visual description of the described attack framework.

In this work, we explore two instances of \( \text{SampleGen} \): a new algorithm, \text{QuickFuzz} and Zeroth-Order Optimization (Zoo) [6]. We keep \( \text{VictimScan} \) fixed for simplicity, and describe it in the context of attack scenarios in Section 5.2. Our selection is motivated by the need for two distinct sample generators that can empirically characterize \( A \) within our framework.

#### 4.1.1 QuickFuzz

We describe \text{QuickFuzz} in detail in Algorithm 1. The intuition behind the design of \text{QuickFuzz} is to perform a discrete, uniform random walk through the signature space, using the provided input signature as a starting point. This approach is inspired by program-fuzzing attacks in the systems literature. In the outer loop of Algorithm 1, \( \text{SampleGen} \) generates several adversarial samples using the adversary’s own list of signatures as a seed; in the inner loop, it invokes a generic \text{Perturb} subroutine that can be tuned for specific fingerprinting domains. The effective goal of the inner loop is to reach a \text{NO} result, which signals that the adversary has crossed their own decision boundary, and potentially entered another user’s decision space. Thus \text{QuickFuzz} assumes the MDA system acts as a one-vs-rest classifier internally. In our implementation, \( \text{Perturb} \) offsets a small, random subset of attributes in \( \text{sig} \), each by an amount \( \delta \) that follows a uniform distribution, i.e., \( \delta \sim \mathcal{U}(a, b) \) for lower bound \( a \) and upper bound \( b \). In practice, \( b \) is an evaluation of \( f(v) \), an upper bound function controlled by parameter \( v \) in Algorithm 1. In each iteration of the inner loop, the upper bound increases by a function of \( v \), thus, acting as a search step-size parameter. The step-size is a trade-off between granularity of the search through signature space and number of queries made to \( \text{Auth} \). For any attribute value \( x_i \in \text{sig}_A \) of the associated pair \((\text{id}_A, \text{sig}_A)\), the perturbed attribute value \( x_i + \delta \) is clipped so that the resulting pair \((\text{id}_A, \text{sig}_A^*)\) is still in the domain of \( C \).

---

Figure 1: Visual description of the attack framework. Adversary \( A \) uses \( \text{SampleGen} \) to generate (multiple) adversarial samples \( \text{sig}_A \in \mathcal{S} \) and store them in a set \( \mathcal{D}_A \). It uses \( \text{VictimScan} \) to scan and select a victim device \((\text{id}, \text{sig})\), and then pairs the device with a signature \((\text{sig}^*)\) from \( \mathcal{D}_A \) to fool \( \text{Auth} \).

---

**Algorithm 1 QuickFuzz**, our fuzzing-based implementation of \( \text{SampleGen} \) for adversarial sample-crafting.

**Input**: \( \mathcal{L}, \sigma \), an arbitrary upper bound on number of adversarial samples to create, and algorithm \( \text{Auth} \), an interface available for querying \( \mathcal{D} \).

**Output**: Set of adversarial samples, \( \mathcal{D}_A \).

\[
\begin{align*}
\mathcal{D}_A &\leftarrow \emptyset \\
R &\leftarrow \text{YES} \\
v &\leftarrow 0 \triangleright \text{(Distortion parameter)} \\
\text{while} \ |\mathcal{D}_A| < \sigma \text{ do} & \\
\quad \text{sig} &\leftarrow \text{random_sample}(\mathcal{L}_A) \\
\quad > \text{(Loop until we meet sufficient distortion for a NO result.)} \\
\quad \text{while} \ R = \text{YES} & \\
\quad \quad \text{increment}(v) \\
\quad \quad \text{sig}_A &\leftarrow \text{Perturb}(\text{sig}, v) \\
\quad \quad R &\leftarrow \text{Auth}(\text{id}_A, \text{sig}_A) \\
\quad \text{end while} & \\
\quad \text{append}(\mathcal{D}_A, \text{sig}_A^*) & \\
\text{end while} & \text{return} \mathcal{D}_A
\end{align*}
\]
In practice, \( a \) and \( f(u) \) are initiated by experimenting with the adversary’s own signatures, since the adversary simply notes how many queries it takes to achieve a NO result. The function \( f(u) \) can be defined separately for each fingerprinting domain, as each may rely on different ranges of feature attribute values. These are provided in subsequent sections.

4.1.2 Zoo. Given only query access to a function, Zeroth Order Optimization (ZOO) is a method for performing optimization of that function [35]. Recently, Chen et al. [7] demonstrated attacks on ML models using ZOO. Unlike transfer-based attacks which approximate the victim model with a separate substitute model, a ZOO attack approximates the victim model’s gradient directions through a series of specially-crafted queries. Most zeroth-order attacks can leverage probabilities returned from the model, which are known as score-level attacks. HopSkipJumpAttack [6], which is the variant we use in our experiments, only requires the top-1 label prediction from the model to be successful. This variant of ZOO attack is known as a decision-level attack, since it only requires the model decision to approximate the best gradient direction. Thus, it is compatible with our hard-label authentication threat model. Specifically, we leverage the untargeted version presented by Chen et al. to act as a walk through the signature space, guided by the approximated gradient information. We denote the HopSkipJumpAttack in our discussion as Zoo. We use the Cleverhans adversarial machine learning library [36] as the reference implementation of HopSkipJumpAttack. This implementation was originally for image classification adversaries; we modify it to operate over the domain of Auth instead.

In our experiments, for simplicity, VictimScan will select the victim devices in a deterministic fashion. With the default Cleverhans hyperparameters, Zoo requires several thousand queries to find YES samples. Likewise, it is necessary to empirically tune the hyperparameters over many queries. Due to the complexity of Zoo and different authentication system characteristics, it took between 500 to 3000 queries to initialize hyperparameters that made Zoo competitive. This is a downside of using gradient estimation techniques for the authentication setting. We do not count these queries in later tallies in order to make the comparison withQuickFuzz consistent.

4.2 Metrics

We define two metrics to help us explain the effectiveness of the QuickFuzz and Zoo attacks against an MDA system that is defined over sets \( (I, S, C) \).

4.2.1 False-positive rate. First let us define a run as a process where each user in the system attempts to attack every other user, one user at a time. We define the false-positive rate \( \alpha \) of a run as \( \alpha = \frac{r}{n^2-n} \), where \( r \leq (n^2-n) \), denotes the number of successful attacks across all attacks in the run. Intuitively, this is equivalent to taking the average of an \( n \times n \) adversary-victim Boolean matrix (1 denotes successful attack, and 0 for failure), but ignoring the diagonal values. In our experiments, we average \( \alpha \) over three runs. In each run, we swap a (fresh) victim’s device with the adversary’s device and run the attack. This effectively gives us the average efficiency of an attack.

4.2.2 Distortion. Fix some \( \operatorname{sig}_j \in S \), and some randomized Perturb procedure. Let \( \operatorname{sig}_j^* = \operatorname{Perturb}(\operatorname{sig}_j) \). We define the distortion \( \epsilon_j \) of the \( j \)-th sample pair \( (\operatorname{sig}_j, \operatorname{sig}_j^*) \) as \( \epsilon_j = \frac{||\operatorname{sig}_j - \operatorname{sig}_j^*||_2}{||\operatorname{sig}_j||_2} \). We also define the average distortion \( \bar{\epsilon} \) across multiple (say, \( p \)) runs of Perturb as \( \bar{\epsilon} = \frac{1}{p} \sum_{j=1}^{p} \epsilon_j \).

5 SETUP

Our experiments are motivated by the following two research questions related to the central hypothesis:

1. What properties of an MDA system \( AS \) are “exploitable” by an adversary in our attack framework? In particular, do SigGen, Train and Auth — algorithms that define \( AS \) — expose attack surfaces that an adversary can access?
2. To what extent are these properties exploitable? For example, how many queries suffice to fool \( AS \)?

We answer these two questions by attacking three systems: USB-Fingerprinting [4], GTID [40], and WDTF [11]. Each of these three systems provide fingerprinting facilities that can form the foundation of an MDA system. Notably, every system has been recently published at academic security venues. We selected these three systems as: (a) each tackles a significantly different fingerprinting domain, (b) the description of each system was clearly specified, and (c) raw data was available for all three systems, in contrast to deployed authentication systems which use inaccessible models and data.

5.1 Target MDA systems

We explain each MDA system by describing its three component algorithms: SigGen, Train and Auth. Specific implementation details are available in Section A.1 of the Appendix.

5.1.1 USB-Fingerprinting [4]. In this MDA system (denoted USB-F), SigGen uses specific USB-enumeration timings of a computer under test to generate device signatures. The Train algorithm trains the underlying ML model (Random Forest) in a target vs. outlier fashion. It creates a new model for every device that registers on the system, and balances outlier classes with respect to every possible device in the system. Train also performs over-sampling (with replacement) until the number of target samples matches the number of outlier samples. The Auth algorithm uses majority voting over multiple signatures to make a pass/fail decision.

5.1.2 GTID [40]. In this MDA system, SigGen uses inter-arrival times of network (TCP) packets to generate device signatures, for both device authentication and device-type authentication. The Train algorithm uses Artificial Neural Networks (ANNs) as the underlying ML model. It uses an ensemble approach by training one ANN for device identification, and another ANN for device-type identification. Although tested as a fingerprinting technique, GTID is advertised by the authors as a potential authentication system [40]. Thus, we defined Auth as follows. A device passes authentication only if the GTID heuristics (Algorithm 1 from Radhakrishnan et al. [40]) output a predicted device ID and device type that matches the user’s claimed device id and device type.

5.1.3 WDTF [11]. In this MDA system, SigGen uses probe-request traffic of IEEE 802.11 wireless devices to generate device signatures.
### Table 1: Lower and upper bounds of the uniform distribution used by QuickFuzz’s Perturb implementation, as described in Section 4.1.1.

<table>
<thead>
<tr>
<th>System</th>
<th>( \delta \sim \mathcal{U}(a, f(v)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTID</td>
<td>( \delta \sim \mathcal{U}(0, 2 \times (v + 1)) )</td>
</tr>
<tr>
<td>USB-F</td>
<td>( \delta \sim \mathcal{U}(10^{-25}, 10^{-20+\varepsilon}) )</td>
</tr>
<tr>
<td>WDTF</td>
<td>( \delta \sim \mathcal{U}(10^{-15}, 10^{-17+\varepsilon}) )</td>
</tr>
</tbody>
</table>

The authors use a customized statistical model based on some distance function \( \kappa \) (that is parameterized using the representative device signature of all devices) to instantiate \( \text{Train} \). The \( \text{Auth} \) algorithm uses \( \kappa \) to compute a list of distance-values between the signature of the device under test and the representative signature of all registered devices. If none of the distance values are less than some pre-determined threshold value, then \( \text{Auth} \) returns 0; otherwise, it returns 1.

We provide the function \( f(v) \) used in each QuickFuzz attack of the three systems in Table 1. These values were found empirically with knowledge of \( \text{Sup}(\text{SigGen}) \), using between 5 to 20 queries to find a reasonable step-size before achieving NO from \( \text{Auth} \). These queries do not have to be performed by the adversary in a single session, so they are not included in later query tallies.

#### 5.2 Attack Scenarios

Each of the described systems are evaluated under different attack scenarios — described using SampleGen and VictimScan — to answer our core research questions. Each scenario uses the metrics from Section 4.2. Note that SampleGen gets \( L_A = \text{SigGen}(\text{id}_A) \) as input and returns the adversarial-sample set \( D_A \). Regardless of the attack scenario, we always measure against the same unseen test set to give a fair comparison. In addition to our metrics, we rely on the notion of accuracy and recall defined by the scikit-learn library, and calculate them in attack scenarios by including any found adversarial samples (with replacement) into the test set during evaluation.

**5.2.1 Baseline.** In the baseline case, SampleGen initializes \( D_A \) with all signatures \( \text{sig}_A \) in the list \( L_A \). Likewise, Perturb is simply an identity function. The VictimScan fixes an arbitrary set \( V \subseteq I \) of victim devices, and queries \( \text{Auth} \) by sampling (with replacement) id from \( V \) and sig from \( D_A \). In our evaluations, the baseline case will be used to establish the lower bound on the system’s robustness.

**5.2.2 Random.** Let \( \min(L_A), \max(L_A) \in \mathbb{R} \) denote the minimum and maximum signature values in \( L_A \). Let \( m \) be some positive integer. In the random test case, SampleGen samples \( m \) signatures uniformly at random from the range \( \min(L_A), \max(L_A) \) and stores them in set \( D_A \). The VictimScan algorithm is same as that of the baseline case. Although random samples may fool the underlying model, they can be easily detected by the MDA system using adversarial training [46]. As we show in later sections, the random strategy is also not viable for maximizing \( \alpha \).

**5.2.3 Greedy and exploratory.** In both the greedy and exploratory cases: the SampleGen algorithm can be instantiated with either QuickFuzz or Zoo; the VictimScan algorithm is the same as that of the baseline case. The difference between greedy and exploratory cases is that in the latter, the number of queries that the adversary can make to \( \text{Auth} \) is bounded by a pre-determined threshold value that is less than the total number \( N \) of possible queries. \( (N = |D| \times |I|) \). On the other hand, in the greedy case, the adversary is allowed to make \( N \) queries to \( \text{Auth} \). Moreover, in the exploratory case, we are interested in computing the number of victim devices that the adversary can masquerade. Hence, the adversary tries other victim devices until it exhausts its query budget. That means, even when its first query to \( \text{Auth} \) is successful, it continues attempting to masquerade other victim devices. In the greedy case, the adversary stops as soon as \( \text{Auth} \) returns one.

#### 5.3 Attack post-mortem

In this section, we discuss three metrics that will be useful to carry out a post-mortem analysis of any attack that is captured by our attack framework. We allow the post-mortem analyst to have full access to all three algorithms — \( \text{SigGen}, \text{Train}, \) and \( \text{Auth} \) — that define an MDA system.

**5.3.1 Local Intrinsic Dimensionality — exploiting expected dimensionality of \( \text{SigGen} \).** The authors of each fingerprinting system apply expert knowledge of their respective domains to design classifiers that can overcome the variance of emitted signatures. However, they do not consider the curse of dimensionality, which states that as dimensionality of a data set increases, the effectiveness of distance-based measurements (generally) decreases. The intrinsic dimensionality (ID) of a data set quantifies the relationship between the dimensionality of a data set and the effect on distance-based measurements [24]. This notion extends to the local continuous intrinsic dimension (LCID) around a point in the set using nearest-neighbor distances. An approximation of the LCID value is known as the Local Intrinsic Dimensionality (LID). Recently, Amsaleg et al. showed that LID values can be used to explain successful adversarial attacks against ML models [1]. More precisely, they showed that as LID increases, the amount of perturbation needed to move into an adversarial region decreases. In our post-mortem analysis, we study the unperturbed signature samples and the adversarial samples that are generated using different instances (QuickFuzz and Zoo) of \( \text{SigGen} \). At a high level, LID quantifies the adversarial subspace within each model. This is analogous to the acceptance region effect showcased by Zhao et al. in biometric authentication systems [49].

**5.3.2 Feature Attribution — exploiting attack surface of \( \text{Train} \).** Given the heterogeneity of the underlying ML models that the MDA systems use, we must abstract away much of the model-specific behavior already covered in the literature [18, 27, 46]. Although the MDA systems seem incomparable on the surface, they share the common assumption that devices are unique due to hardware-manufacturing imperfections. These imperfections are thought to be captured by the signature-generation algorithm, \( \text{SigGen} \).

Our analysis of features of the \((\text{id}, \text{sig})\) pairs is enabled by recent work in the Explainable AI (XAI) literature [15, 20, 41]. The main

\[1https://scikit-learn.org/stable/modules/model_evaluation.html\]
We examine the effect of leveraging XAI is homogenizing the discussion between heterogeneous methods and data, by not biasing analysis towards any particular model architecture’s “naïve” explainability (i.e., decision trees and hand-crafted heuristics). This motivates our use of Local Interpretable Model-agnostic Explanations (LIME), an XAI technique proposed by Ribeiro, Singh, and Guestrin [41]. LIME is a good match for our setting as: (a) it is model agnostic (only requires score-level access to the trained model) and (b) it can generate explanations on a per-adversarial sample basis.

Given an ML model, LIME generates a linear approximation of the model and allows computation of attribution scores for each feature in the original model. In the MDA setting, we use the adversarial signatures — generated by QuickFuzz and Zoo — as inputs to LIME to get a linear approximation of the trained MDA system’s model (i.e., the output of Train). Based on the attribution scores that LIME outputs, we analyze the features that make an MDA system susceptible to AML-style attacks. It is worth noting that recent works have attempted to subvert the explanation ability of XAI methods, namely back-propagation-based interpreters [14, 48] and LIME [44]. In our setting, the post-mortem analysis uses LIME honestly.

6 RESULTS

We organize our discussion using the research questions posed in Section 5 as a road map.

6.1 Attack Effectiveness

We examine the effect of SampleGen instantiations combined with the defined strategies.

6.1.1 Performance Impact. In Tables 2, 3, and 4, we compare against the published results, our baseline, and attack scenarios for USB-F, GTID, and WDTF, respectively. For USB-F in Table 2, we find that our implementation exceeds the published results in the regular device identification scenario described by Bates et al. [4]. When under the Random attack described in Section 5.2.2, the system admits no adversaries (89% accuracy, 0% recall). However, the accuracy and recall diminish reliably as any given instance of $A$ expands their search. A greedy adversary manages to reduce accuracy down to 80% using QuickFuzz, lower than the 93% with Zoo. However, as the query budget is relaxed, Zoo ultimately outperforms QuickFuzz in terms of accuracy. In the worst case, the 500-query adversary with Zoo will reduce accuracy to 59% and recall to 52%. This contrasts with the case of GTID in Table 3, as QuickFuzz is generally more successful regardless of query budget. We note that our implementation of GTID had very close performance to the published result of 99% accuracy, with only some false negatives (83% recall versus published 94% recall). We observed that during successful attacks, the device id ANN was sufficiently confident in adversarial samples to bypass the UNKNOWN path of Algorithm 1 from Radhakrishnan et al. [40]. On the surface, this appears to reflect similar vulnerabilities in multi-modal biometric authentication [26, 42]. Essentially, the adversary instance $A$ can exploit failure modes of the Auth algorithm within the GTID MDA system. Interestingly, Zoo is more successful in all adversarial cases of WDTF in Table 4. In fact, accuracy for WDTF fell to 25%, accompanied by a recall score of 0% in the worst case. Even the greedy adversary case is successful against WDTF, with an accuracy of 69%. Despite the three MDA systems relying on different pattern recognition techniques, each was susceptible to the tested SampleGen instances.

6.1.2 Attack Distribution & Queries. We delve further by examining the spread of damage to the MDA systems when under attack. This is visualized for each system in Figures 2, 3, and 4. The access matrices allow us to make high level observations about the SampleGen instances. The top and bottom rows of the access matrices correspond to the QuickFuzz and Zoo SampleGen instances, respectively. Each access matrix represents the average over three random seeds of running an attack scenario, where each adversary...

---

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuickFuzz</td>
<td>94-99%</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Performance comparison of USB-F scenarios. Measurements are averaged over three runs.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuickFuzz</td>
<td>99%</td>
<td>94%</td>
</tr>
</tbody>
</table>

Table 3: Performance comparison of GTID scenarios. Measurements are averaged over three runs.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuickFuzz</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 4: Performance comparison of WDTF scenarios. Measurements are averaged over three runs.
instance with identifier id_i occupies a row, and each column represents attempts on victim id_o. Color intensity corresponds to success rate.

Notably, some victims tend to be more susceptible than others, and likewise, some adversaries tend to be stronger than others. This behavior differs depending on the exact attack used. For instance, QuickFuzz on USB-F in Figure 2 allowed increasingly higher access to victims zero, three, and seven as queries were increased. This behavior amplifies with Zoo, allowing increased access to additional victims two, four, and six. False positive rate δ is notably higher using Zoo, peaking at 40% in the worst case, despite requiring more queries than QuickFuzz. A similar pattern can be observed with the random attack on GTID, shown in Figure 3. Victim two is more susceptible to the random attack than any other principal. Unlike with USB-F, QuickFuzz was generally more successful against GTID, even requiring less queries, and spreading the attack impact more evenly. Even with 100 queries, the QuickFuzz attack manages to fool GTID with a δ of 12%. Zoo manages to outperform QuickFuzz when used on WDTF, as shown in Figure 4. WDTF is the most susceptible system, reaching a δ of 67% in the worst case. Overall, test of the two attacks against the three systems show that a trade-off exists between number of queries and success. The more complex Zoo attack is not necessarily more successful in every case, as it sometimes requires more queries or fails to find failure modes of the MDA system.

6.2 Feature Attribution Analysis

As described in Section 5.3, we aggregate positive feature attribution weights across every victim in each attack scenario, to visualize the features that were most important in influencing a decision. For ease of comparison and visualization, we select up to eight victims and their perturbed samples from the Exploratory A scenario (q = 300), and query LIME to return only the top-eight features attributing to decisions.

6.2.1 USB-F. Figure 5 shows feature attribution weights for up to eight victims under the USB-F method with QuickFuzz on top, and Zoo on the bottom. We see that the attribution weights returned by LIME vary based on the attack. Samples generated with QuickFuzz offered lower weights compared Zoo. In the case of Zoo, we can make some high level observation. Mainly, any arbitrary adversarial sample only relied on a handful of features, according to LIME. For example, victim ‘vatta’ was subverted with feature subset \( \{X_{12}, X_{18}, X_{23}, X_{24}\} \). In fact, this pattern is not unique to ‘vatta.’ Adversarial samples needed no more than five of the features to align with the victim to be successful. Of interest is that features...
\( \{x_26, x_27, x_28\} \) were never used. Since attacked features vary between victims, it is not clear that domain knowledge would help secure the model.

6.2.2 GTID. The feature attribution for GTID across victims is shown in Figure 6. In either case, we notice that attributions across victims is clustered relatively close to the center. Since each attack had similar scaling of attribution weights, we can now see that specific features vary between attacks. For example with Zoo, ‘DevIT2, iPhone3G’ was subtverted with features in the center of the signature, while features in the periphery were used with QuickFuzz. This behavior is seen with other victims, notably ‘DevIT1, iPhone3G’ and ‘DevIF2, iPhone4G’. Some overlap occurs with ‘DevIF1, iPhone4G’ as features were selected towards the center. Radhakrishnan et al. select the start and end points of the signature histogram to fit the peak of tested traffic (see Footnote 3 [40]). It is possible that the model attends disproportionately to the central peak of a typical signature, rather than the periphery. Apart from this spatial correlation between original sample and feature attributions, there is little relation between the perturbed features and the packet arrival-time semantics. Likewise, only a handful of features are chosen in either attack, as shown prior with USB-F. We conclude that the detection of these brittle features during design would be non-obvious.

6.2.3 WDTF. Feature attribution weights for the four victims under WDTF were very similar to USB-F and GTID, and are omitted for brevity. We observed that the adversary only needs a handful of important features for the attack to succeed. This was true regardless of the considered victim. Likewise, features also varied between each attack, so connection back to domain semantics would not be helpful.

6.3 Local Intrinsic Dimensionality (LID)

In the previous section we saw that component-wise feature analysis is difficult, as each instance of SampleGen exploited different features. We abstract away features and instead focus on analyzing the adversarial subspace of inputs to each MDA, as described in Section 5.3. Figures 7 and 8 show the LID values of randomly selected normal (blue) and adversarial (red) SampleGen signatures from the USB-F and GTID MDA systems, respectively. For brevity, we omit results from WDTF, as they mirrored those of GTID. As in the analysis performed by Ma et al. [29], we use min-max normalization to obtain LID scores in the range \([0.0, 1.0]\).

Our results are consistent with the findings of Ma et al., which is that adversarial regions in the signature space can be characterized by equal, or higher, LID scores than normal data regions. This effect is primarily noticeable with QuickFuzz for both USB-F and GTID, and Zoo on GTID. We interpret this as follows. Although expert knowledge can arrive at intuitive features that are useful for classification, they contain a (local) intrinsic dimensionality that increases as points move away from the original SigGen output.
space. It suffices to move in random directions to find local submanifolds with high complexity, as evidenced by the success of QuickFuzz. Zoo on USB-F had mixed results, with LID fluctuating between normal and adversarial levels, while Zoo on GTID behaves as expected. One possible explanation is that Zoo unintentionally crafts samples with lower LID during gradient approximation.

6.4 Attack Distortion Characteristics

We empirically evaluate the average distortion induced by each SampleGen instance of QuickFuzz and Zoo in Table 5. Generally, the QuickFuzz instances produced high values of $\varepsilon$ with high variability. Zoo tends to have lower distortion on USB-F and GTID with lower variation, but increases with WDTF. We interpret this as a by-product of the methods. WDTF exhibited high false-positives to random samples earlier in Section 6.1, whereas USB-F did not. We essentially have two extreme cases of robustness to random uniform noise, with USB-F responding positively, GTID performing moderately, and WDTF responding negatively. In this sense, a random walk should induce more noise with USB-F than the guided search of Zoo. WDTF only requires 2.30% distortion on average with random walk, but needs 93.8% with Zoo. This indicates our SampleGen algorithms cover two distinct strategies. The strategy to choose depends on the data Train was instantiated with.

<table>
<thead>
<tr>
<th>SampleGen</th>
<th>USB-F</th>
<th>GTID</th>
<th>WDTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuickFuzz</td>
<td>169±529</td>
<td>255±474</td>
<td>2.30±2.29</td>
</tr>
<tr>
<td>ZOO</td>
<td>8.36±768</td>
<td>66.3±81.1</td>
<td>93.8±53.4</td>
</tr>
</tbody>
</table>

Table 5: Average distortion $\varepsilon$ induced by each SampleGen instance of QuickFuzz and Zoo.

7 MITIGATIONS

Our analysis of MDA adversaries in the previous section poses an interesting challenge. Although XAI can tell us which features are brittle, the exact features tend to vary between instances of SampleGen. Abstracting away features and focusing on the adversarial manifold enables the use of LID. Although encouraging, LID was more stable with Zoo than with QuickFuzz. Thus we attempt to protect an existing MDA system without relying on knowledge of the domain or assumptions of the underlying manifold. In this section, we rely on state-of-the-art results obtained by randomized smoothing [10]. Cohen et al. show that randomized smoothing can efficiently certify robustness given a fixed parameter $\sigma$ which bounds the size of distortion an adversary induces.

We examine the effect of the two SampleGen instances when Train is modified according to the randomized smoothing technique. Notably, this defense is viewed as modifying Train’s output, due to training with Gaussian data augmentation. We apply the randomized smoothing technique to GTID to obtain the smooth-GTID MDA system. In practice, we remove the heuristics proposed by the original authors of GTID in favor of those presented by Cohen et al. due to effects noted in Section 6.1.1. In this instance, randomized smoothing necessitates modification of Auth, although it may not be necessary in all cases.

Based on guidance from the work by Cohen et al. [10], we select $\sigma = 0.5$ for the main experiment, shown for each instance of SampleGen in Figure 9. Experimentally, we observed similar results regardless of the choice of $\sigma$. The randomized smoothing nullifies any false positives from the Zoo instance, despite losing some coverage in the form of false negatives. With the QuickFuzz instance, some false positives are admitted, although they are rare compared to the prior cases in Section 6.1. We interpret the success of QuickFuzz as a symptom of decreased accuracy, due to the noisy instantiation of Train, and the neural network’s lack of capacity for such noise. We can infer that choosing a more expressive model is the key to truly certifying GTID. Given the removal of the original GTID heuristics, we can further conclude that existing domain-specific heuristics have little to offer in terms of adversarial robustness.

With the smooth-GTID system, we revisit our original post-mortem analysis using LID, albeit using only the QuickFuzz instance of SampleGen, as smooth-GTID thwarts the Zoo attack. The LID scores for the smooth-GTID attack signatures is shown in Figure 10. A main takeaway is that LID scores are either similar to their normal counterparts, or very high. When comparing against Figure 8, there is no visible trend in terms of maximum or minimum LID scores. We do note that signatures with high ‘normal’ LID score had a high ‘adv’ LID score, and vice versa for low scoring signatures.

Due to the small sample size, it is difficult to make general claims, but these initial results could suggest that the signature space inhibits high LID as a result of the randomized smoothing mechanism, at the expense of raising the baseline LID score of normal samples.
8 DISCUSSION

From our experiments, we uncover attack surfaces in the algorithms of our formalized MDA system. Each algorithm can be exploited due to different assumptions that are made during design of an MDA system.

In Section 6.2, we saw that QUICKFUZZ and ZOO only needed a handful of features to fool the MDA systems under test. Since $A$ is essentially blind, SampleGen allows advancing blindly in input space, through selective fuzzing in the case of QUICKFUZZ, or with guided heuristics as in ZOO, until a correct combination of brittle features is met. LID analysis in Section 6.3 shows that sub-manifolds exist in the signature space with high complexity. Since signatures are the output of SigGen, we can infer that design of SigGen is to blame. This may explain why the adversary does not need many queries. In this sense, we notice some similarity to single-pixel attacks in the image domain [45]. However, we also consider the following. Although all features may contribute towards a learning task, the Train algorithm conditions the model to attend to certain features. As described by Goodfellow et al. [19], an adversary needs to only find the feature values which the model aligns most with. Despite being less data-driven, MDA systems are equally vulnerable to this phenomenon.

Apart from SigGen and Train, we showed that previously proposed heuristics for designing Auth are not secure. Although the decision process can benefit from hand-crafted features, they do not imply robustness against adversaries. The feature attribution analysis showed that target features vary between victims, and even between attacks. Thus, it is not practical to use domain-specific heuristics for defending models. Instead, one must consider underlying properties of adversarial samples. LID is an encouraging first step, as it can abstract away knowledge of specific features. We took this abstraction further, and showed that designers of MDA systems can essentially ignore domain knowledge, by applying end-to-end style defenses such as randomized smoothing. However, such techniques rely on sufficiently expressive models. We summarize by recommending designers focus on applying end-to-end style defenses, which shifts the security challenge to one of model selection, rather than heuristics crafting.

9 RELATED WORK

The progression of attacks in the Adversarial Machine Learning (AML) space often focus on particular applications of machine learning systems. Papernot et al. provide a recent survey of the general deep learning attack landscape, including a high-level view of different threat models and adversary goals [39]. Due to the constrained, hard-label feedback of learning systems in-the-wild, limited-information attacks are incredibly valuable, despite tending to be less powerful. However, such attacks are less prevalent in the literature. Conceptually, the problem can be framed in the restricted query model outlined by Angluin [2]. Biggio briefly considers a concept learning attack against a signature authentication system [5], although in the absence of counting adversarial queries. These iterative attacks evolved into camouflage-style attacks which intend to cloak the adversary in ‘fashionably’-crafted accessories or clothing that can fool the authentication system [16, 43]. Recent years have seen the state-of-the-art in limited information attacks, which rely on zeroth-order optimization (ZOO) methods to approximate the victim model’s gradient information [7]. The most recent ZOO attacks rely only on the top-1 decision from the victim model [6, 8, 9].

10 CONCLUSION

Although machine learning is a powerful tool for performing device authentication, previous works failed to consider the susceptibility of such systems to AML-style attacks. We demonstrate new restricted-query attacks against device authentication that are successful regardless of the underlying machine learning model. With the help of XAI techniques, we discover that the features used in device authentication are often brittle, and selective perturbation of certain features can be highly effective at breaking device authentication systems.

ACKNOWLEDGMENTS

The authors would like to acknowledge Dave Tian and Scott Clouse for their helpful comments on early revisions of the paper. This work was supported by AFOSR FA9550-19-1-0169, NSF CNS-1815883, and NSF CNS-1562485.

REFERENCES


null