SoFi: Spoofing OS Fingerprints Against Network Reconnaissance

Xu Han, Haocong Li, Wei Wang, *Member, IEEE*, Haining Wang, *Fellow, IEEE*, Xiaobo Ma, *Member, IEEE*, Shouling Ji, *Member, IEEE*, and Qiang Li

***Abstract*—Fingerprinting is a network reconnaissance technique utilized for gathering information about online computing systems, including operation systems and applications. Unfortunately, attackers typically leverage fingerprinting techniques to locate, enumerate, and subsequently target vulnerable systems, which is the first primary stage of a cyber attack. In this work, we explore the susceptibility of machine learning (ML)-based classifiers to misclassification, where a slight perturbation in the packet is included to spoof OS fingerprints. We propose SoFi (Spoof OS Fingerprints), an adversarial example generation algorithm under TCP/IP specification constraints, to create effective perturbations in a packet for deceiving an OS fingerprint. Specifically, SoFi has three major technical innovations: (1) it is the first to utilize adversarial examples to automatically perturb fingerprinting techniques; (2) it complies with constraints and integrity of network packets; (3) it achieves a high success rate in spoofing OS fingerprints. We validate the effectiveness of adversarial packets against active and passive OS fingerprints, verifying the transferability and robustness of SoFi. Comprehensive experimental results demonstrate that SoFi automatically identifies applicable and available OS fingerprint features, unlike existing tools relying on expert knowledge.**

***Index Terms*—Adversarial machine learning (ML), fingerprinting, operating system (OS).**

I. Introduction

NOWADAYS, cyberattacks have increasingly targeted critical infrastructures as attackers attempt to circumvent the deployed security measures and compromise computing systems, resulting in serious consequences, such as safety failure and service disruption. Typically, a cyberattack follows a distinct cycle with different phases characterized by network reconnaissance, enumeration, exploitation, intrusive attempts, and performing malicious tasks. As the initial stage of the cyberattack cycle, network reconnaissance involves attackers gathering relevant information about remote hosts, such as their operating systems and applications. Given that there are nearly 25 billion daily global intrusion attempts — a number that continues to rise [1] — it is important to note that almost 50% of these cyber attacks involve network reconnaissance to extract the operating system or port information from remote systems [2].

As a vital component of network reconnaissance, fingerprinting allows attackers to remotely locate, target, and potentially exploit vulnerable systems by identifying a remote host's operating system (OS). Classic scanning tools like Nmap [3] are often used by both attackers and network administrators to uncover underlying hosts and extract OS-related information. Recent studies [4], [5], [6] have proposed using machine learning algorithms to generate OS fingerprints and infer classification models. Subsequently, these fingerprinting techniques may enable attackers to locate vulnerabilities and bypass deployed security measures, thereby increasing the risk of successful intrusions.

Spoofing OS fingerprints stands as a vital and proactive line of defense against underlying cyber attacks due to the importance of OS fingerprints to attackers. In particular, OS fingerprints help attackers obtain detailed OS information and then successfully run an exploit against vulnerable computing systems/devices. There are instances when patches are not readily available or feasible to apply, leaving an OS version unpatched. By pinpointing the specific OS of a host, attackers can launch crafted attacks against target machines. Besides nullifying the reconnaissance efforts of attackers, from a privacy perspective, spoofing fingerprints can prevent the unsolicited revelation of a user's OS and application information, enhancing user privacy.

In this paper, we propose a novel approach to spoofing OS fingerprints through adversarial examples by subtly altering packet data. Inspired by recent studies indicating that the performance of learning-based detection techniques can be degraded by carefully crafted variants in the image [7] and the text [8] domains, we adapt this insight to the defense against network reconnaissance. We devise evasion techniques for rapid, preemptive endpoint defense. Our basic idea is to alter the data point during inference, induce misclassifications, and then disrupt the OS fingerprints' abilities.

Received 31 July 2024; revised 3 February 2025 and 3 April 2025; accepted 3 April 2025. Date of publication 18 April 2025; date of current version 2 May 2025. This work was supported in part by Beijing Natural Science Foundation under Grant L221014 and Grant M23019; in part by the National Natural Science Foundation of China under Grant 62272029; in part by the Systematic Major Project of China State Railway Group Company Ltd., under Grant P2023W002, Grant P2024S003, and Grant P2024W001-4; in part by the Science and Technology Research and Development Plan of China Railway Information Technology Group Company Ltd., under Grant WJZG-CKY-2023014 (2023A08) and Grant WJZG-CKY-2024040 (2024P01); in part by the Science and Technology Project of Haihe Laboratory of ITAI under Grant XCHR-20230701; and in part by Hangzhou Qianjiang Distinguished Experts Programme in 2024. The associate editor coordinating the review of this article and approving it for publication was Prof. Yanjiao Chen. *(Corresponding author: Qiang Li.)*

Xu Han, Haocong Li, and Qiang Li are with Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, China (e-mail: liqiang@bjtu.edu.cn).

Wei Wang and Xiaobo Ma are with the Key Laboratory for Intelligent Networks and Network Security, Ministry of Education, Xi'an Jiaotong University, Xi'an 710049, China.

Haining Wang is with the Department of Electrical and Computer Engineering, Virginia Tech, Arlington, VA 22203 USA.

Shouling Ji is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China.

Digital Object Identifier 10.1109/TIFS.2025.3561673

The crux of the challenge lies in the oversimplification of existing adversarial techniques [9], [10], [11], [12], [13], which merely transform one OS's feature values into another OS's, compromising packet integrity. These techniques manipulate (add, change, or remove) features within network packets, rendering them invalid. To create adversarial examples satisfying these constraints, we propose SoFI (Spoof OS Fingerprints), an adversarial example generation algorithm under TCP/IP specification constraints. Specifically, SoFI divides OS' features into three categories: mutable features, immutable features, and checked features. By constructing a substitution space for mutable features, SoFI bolsters the feasibility of adversarial examples. Moreover, for each adversarial example, it adjusts checked features to guarantee the validity of the OS fingerprints.

We implement a prototype of SoFI as a self-contained piece of software based on open-source libraries. To validate its efficacy, we systematically evaluate SoFI against a spectrum of machine learning models. These models, trained on an extensively used OS fingerprint dataset, include traditional machine-learning algorithms and deep-learning algorithms. Our evaluation results show SoFI achieves high success rates in deceiving attackers with spoofed passive/active OS fingerprints. In comparison to the baseline's performance, SoFI guarantees the integrity of TCP/IP specifications and has a higher success rate. Additionally, we validate the SoFI 's transferability where we cannot possess internal knowledge and understanding of the OS detectors. Finally, we demonstrate SoFI 's advantages from three perspectives: the comparison with existing defense tools, the comparison of manual versus automation, and the prototype deployment.

Our major contributions are summarized as follows:

- We are the first to propose that adversarial examples can easily deceive both passive and active OS fingerprints.
- We introduce SoFI, a novel approach to generate adversarial examples against OS fingerprinting techniques. Unlike existing tools that depend on expert knowledge, SoFI automatically identifies applicable and available features for deceiving OS fingerprints.
- Our results demonstrate a higher success rate than baselines, revealing that enhancing OS fingerprint robustness is a significant challenge.

Roadmap: The remainder of this paper is organized as follows. Section II provides the background and related work of OS fingerprinting techniques. Section III details the design of SoFI against OS fingerprinting techniques. Section IV describes its implementation details. Section V details experimental evaluation and ML explainability for OS fingerprints. Section VI further demonstrates the advantages of SoFI from three aspects. Section VII presents the discussion and limitations. Finally, Section VIII concludes.

II. Background and Related Work

This section first describes the preliminary knowledge of OS fingerprinting techniques. Then, we present defense strategies against OS fingerprinting.

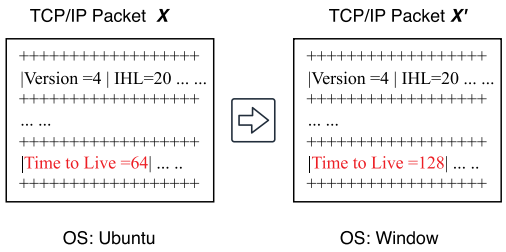

Fig. 1. Passive OS Fingerprints: (1) changing the TTL field, (2) causing a different OS class from 'Ubuntu' to 'Windows.'

A. Os Fingerprinting

In network reconnaissance, fingerprinting has been widely used for more than three decades. OS fingerprinting is to determine the operating system of a remote host on a network. The core idea is that different OS have various implementations of IETF RFCs or TCP/IP protocols, leading to different network behaviors of TCP/IP packets [14]. This technique enables attackers to enumerate and assess potential targets for cyber attacks both locally and across the Internet. Typically, there are two categories of OS fingerprinting: passive and active.

Passive OS fingerprinting is to eavesdrop or sniff network packets traveling between hosts. There are several popular and available passive OS fingerprinting tools, including SinFP3 [15], [16] and p0f [17]. In those tools, the matching rules of OS fingerprints are stored in the database to identify the OS. For passive OS fingerprinting, research works [5], [18], [19], [20], [21] use the input x to infer one of the k OS classes. Zander et al. [22] proposed the differences in time synchronization in the TCP packet header to fingerprint OS information. Shen et al. [23] extracted the spectrogram of the LoRa signal and leveraged a CNN model to generate a hybrid classifier as device fingerprints. The TCP timestamp option [5] might reveal hardware properties, and [19] utilized a sequence of inter-arrival times between network packets for detecting the OS of remote hosts. Ma et al. [24] designed a context-aware system that can fingerprint access to websites using a two-stage spatial-temporal flow correlation approach. Figure 1 depicts two OS classes in the TCP/IP packet header, where the TTL field is 64 for Ubuntu and 128 for Windows.

Active OS fingerprinting is to carefully craft packets with different settings or flags and send them to a remote host to obtain its responses. There are several popular and available active OS fingerprinting tools, including Nmap [3], SinFP3 [15], [16], and Nessus [25]. Nmap [3] is a classic tool to detect OS versions based on the differences between TCP/IP implementations (e.g., TCP window size, max segment size, and options). Those tools use a database to store crafted packets and matching rules for identifying OS. Similarly, prior research works [4], [26] usually leverage machine learning or deep learning algorithms to infer OS classes. Caballero et a. [4] leveraged the SVM algorithm to learn features of packet headers for generating OS fingerprints. Anderson et al. [26] extracted features from TCP/IP for OS fingerprinting in the potential presence strategies. Given a set of k OS classes, the input x represents received responses from a remote host, and

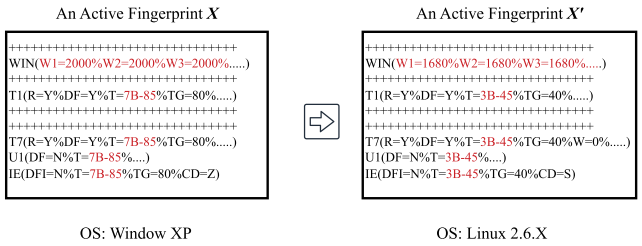

Fig. 2. Active OS Fingerprints: (1) changing the window size and TTL field, (2) causing a different OS class from 'Window XP' to 'Linux 2.6.X.'

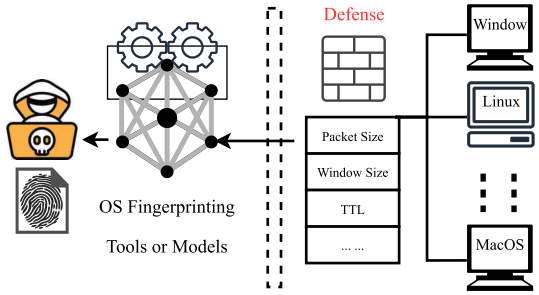

Fig. 3. The defense strategies against OS fingerprinting techniques.

the output y indicates one of the OS classes or an unknown class. Active OS fingerprinting is to learn a mapping function between the input x and the output y, denoted as $y = f(x)$. Note that active OS fingerprinting needs to send additional packets to the remote hosts (e.g., 16 probing packets by Nmap).

Figure 2 illustrates two OS classes ('Window XP' and 'Linux 2.6.X') for their active fingerprints, where each line is a test response for a particular probing packet, % is a separator between fields, W is the TCP window, S is the sequence number, and T is TTL. For example, T1 reflects the results from the first TCP probe, and IE is a test related to the two ICMP echo probes.

B. Defense Against Os Fingerprinting

The defense model thwarts OS fingerprinting to prevent attackers' OS profile collection. To achieve this objective, the defenses can be deployed on computers to deceive the attacker's OS fingerprinting tools or techniques, as shown in Figure 3. Specifically, we divide the OS fingerprinting defense models into two categories: existing tools and research works.

The basic idea of a fingerprinting defense is to change the TCP/IP packet header, e.g., modifying the total length field of the IP header [27]. In other words, we can forge or confuse the fingerprint information of remote hosts to defend against potential cyberattacks. OSfuscate [28] is to change the registry in Windows XP/Vista to deceive the Nmap and p0f. IP Personality [29] and OSfooler [30] leverage IPTables/NetFilter to modify the fixed packet headers for deceiving OS fingerprinting. IP Personality [29] is a Linux kernel patch that can simulate other OS fingerprinting information at the TCP/IP layer, thereby deceiving the fingerprinting tools, e.g., Nmap. OSfooler [30] is an open-source tool for Unix-like operating systems, and it performs fingerprint spoofing by sending specific TCP/IP packets. However, deceiving the fingerprints uses fixed position modification, which is manual and time-consuming. Finding a qualified feature in the TCP/IP packet header is arduous and incomplete, and it is challenging to keep up-to-date with the numerous new implementations and new version updates.

Other OS fingerprinting defense strategies include deploying network honeypots [31], [32] and moving target defense (MTD) [33]. HoneyNet is to mimic OS, where various types of software or hardware-based honeypots are integrated into the network. MTD dynamically changes the attack surface of a computer, disrupting the network reconnaissance and increasing the attack complexity. The disadvantage is that their deployment has a heavy-weight cost and is not well suited for placing obfuscation of OS fingerprinting. Several prior works [34], [35], [36], [37], [38] focused on web fingerprinting obfuscation. In contrast, our work specifically addresses OS fingerprinting, targeting the unique network-level characteristics derived from TCP/IP headers to identify operating systems.

Kampanakis et al. [39] proposed increasing the difficulty of OS fingerprinting by introducing time delays during the TCP handshake. Albanese et al. [40] modified four aspects of TCP/IP packets to spoof OS fingerprints: the total length field in the IP header, the sequence number in the TCP header, the packet size, and the packet fragmentation field. Rahman et al. [41] applied game theory to alter TCP/IP packet fingerprints to confuse attackers' identification efforts. In contrast, website fingerprinting involves a passive local eavesdropper deducing information from users' browser activities. Ling et al. [38] proposed a genetic-based variant to evade/obfuscate website fingerprints, where they injected dummy packets into the raw traffic as the defense strategy. Mathews et al. [42] explored today's popular website fingerprinting defenses, where hand-crafted features may still leak information.

In addition, prior works on OS fingerprinting obfuscation require either system configuration (e.g., kernel patch [29]) or manual efforts. By contrast, we propose to automatically generate adversarial examples of OS fingerprints as the defense mechanism, which is a feasible and promising pro-active defense mechanism.

III. Spoofing Os Fingerprints

SoFi*'s Goal* As we mentioned before, the OS fingerprinting can be summarized as the mapping relation $y = f(x)$, where the input x represents the TCP/IP packet, and the output y is one of the OS classes. Similarly to most evasion attack settings, our goal is to misclassify the input as a class other than the original class. During the OS detection, we craft an adversarial example x' from a legitimate sample x to spoof the mapping function $f(\cdot)$, as the following problem:

$$x' = x + \eta, \quad f(x') \neq f(x); \\ sim(x, x') \leq \epsilon \tag{1}$$

where η is a slight perturbation in the input's packet, and $sim(x, x')$ is the perceptual similarity between adversarial examples and the original ones. Here, $f(\cdot)$ belongs to a multi-class setting, and we have a set of k classes. Our SoFi focuses

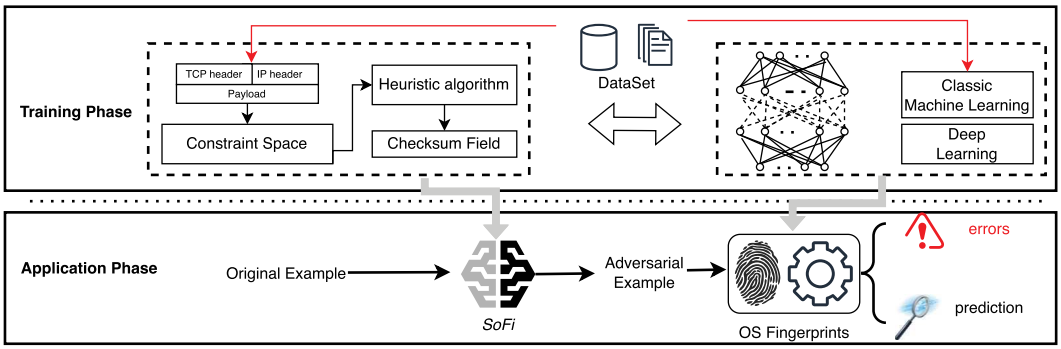

Fig. 4. The SoFi's overview. (1) In the Training Phase: we use classic ML and DL to train an OS model, utilizing the constraints of the packet header to find a suitable perturbation substitution set. (2) In the Application Phase: we use SoFi to generate adversarial examples to deceive OS fingerprinting.

on non-targeted attacks, aiming to misclassify a legitimate OS fingerprint to another class, merely inducing an incorrect prediction.

There are three design requirements for generating adversarial packets, as follows.

- **No Prior knowledge**. Unlike other obfuscating tools, there is no need for prior knowledge to find available and modifiable content for SoFi. The adversarial packet is automatically generated and suitable for updating with the numerous new OS fingerprints.
- **Constraints**. Our SoFi adds a slight perturbation in a TCP/IP packet of a remote host, which will not affect the normal usage of remote hosts. The adversarial packet can still be routable and thus transferred by the communication network rather than being dropped or blocked by the firewall or the router. For instance, an adversarial example would be dropped from the network if it has incorrect IP addresses or checksums.
- **Effectiveness**. SoFi effectively spoofs the OS recognition model, leading to incorrect prediction results. This hampers an attacker's ability to gather general OS information and achieve accurate reconnaissance results from remote hosts. A lower accuracy rate of OS fingerprints indicates superior evasion effectiveness.

Overview. Figure 4 depicts the overview of SoFi. In the offline stage, we employ both classic machine learning and deep learning algorithms to construct a multi-classifier, as OS fingerprints f_θ. These models are treated as complete black boxes; we do not access the model's architecture, parameters θ, or loss function $J(f_\theta(x), y)$. Furthermore, we cannot query the model for confidence scores of the inputs. During the evasion stage, we use one classifier as the surrogate model while others serve as targets. SoFi has access to the surrogate's architecture for manipulation. It modifies TCP/IP packet headers to trigger misclassification in OS detection, segmenting the header fields into mutable, immutable, and checked categories. SoFi employs the weighted fast gradient sign method (W-FGSM) for gradient loss calculation, determining the influence of each field on model sensitivity. For mutable fields, SoFi generates a substitution space within TCP/IP protocol limits. Using the W-FGSM and this substitution space, we propose a heuristic algorithm for adding perturbations to the packet header. Upon generating the adversarial example, we update checked fields to ensure the integrity and validity of the packet. The following provides a detailed breakdown of these modules, including field partition, constrained space generation, and the heuristic algorithm.

TABLE I
FIELD PARTITION IN THE TCP/IP PACKET HEADER WITH CORRESPONDING WEIGHTS

Category	Field	Weight
Immutable Region	ip.version, ip.id, ip.flags.rb, ip.flags.mf, ip.frag_offset, ip.proto, ip.src, ip.dst, ip.dsfield, tcp.srcport, tcp.dstport, tcp.seq, tcp.ack, tcp.len	w_0
Mutable Region	ip.flags.df, tcp.flags.ack, tcp.window_size tcp.flags.fin, ip.ttl, tcp.flags.reset, tcp.flags.push, tcp.flags.urg, tcp.flags.syn,	w_2/w_1
Checked Region	ip.checksum, ip.len, tcp.checksum, ip.hdr_len, tcp.hdr_len	w_0

A. Field Partition

The TCP/IP packet header comprises a variety of fields, with each subject to specific protocol requirements. Consequently, we categorize these fields into three regions: immutable, mutable, and checked. To spoof the fingerprinting technique, we would add a slight perturbation η into those fields under the region constraints. Table I lists these three categories pertaining to the TCP/IP packet header fields.

Immutable Region has inherent TCP/IP packet header fields that cannot be altered without compromising packet validity and integrity, e.g., 'tcp.seq' and 'tcp.ack' fields. Alterations to such fields, like 'ip.src' and 'ip.dst' that represent host network addresses, would directly impair the packet's transmission function.

Mutable Region contains changeable fields (e.g., the time to live, flag, or window size) in the TCP/IP packet header, allowing for legal perturbations through operations such as addition, deletion, and replacement. Legally changing these fields may deceive OS fingerprinting without interference with the regular usage of remote hosts. For example, the 'ip.ttl' field defaults to 128 in the Windows system.

In the mutable region, some fields are independent, and some fields are dependent on others. Randomly changing those fields also breaks the availability and integrity of the TCP/IP

packets. Hence, we would construct a constrained substitution space of each dependent field to represent their dependent constraints in the mutable region.

Checked Region has the fields that are determined by multiple fields or all fields in the TCP/IP packet header. For example, the length of an IP packet is calculated from the sum of the length of the header of the IP packet and the length of the TCP packet. Those checked fields are used to ensure the validity and integrity of TCP/IP packets. Typically, checked fields cannot be perturbed by any slight change for deceiving OS fingerprinting. A slight perturbation in the TCP/IP packet header would affect those checked fields. Therefore, we must modify the corresponding checked fields after generating a qualified adversarial example.

W-FGSM. We propose the weighted fast gradient sign method (W-FGSM) to identify perturbations for approximatively minimizing the OS fingerprints' loss functions. The FGSM proposed by Goodfellow et al. [9] can linearize the learning-based model's cost function around the input to be perturbed and select a perturbation by differentiating this cost function with respect to the input itself. The distinction in W-FGSM is that we combine the loss gradient method with the weight to calculate a change propensity score for each field in the TCP/IP packet header. The W-FGSM equation is defined as follows,

$$T(f_i, y) = w_i * |\partial_{f_i} J(x, F, y)|_2, \tag{2}$$

where x is a TCP/IP packet, y is the class label of OS, J is the loss function of the model F. Each field f_i is assigned a weight w_i corresponding to its region, with w_i being part of the set w_0, w_1, w_2. The respective values of w_0, w_1, and w_2 are outlined in Table I. Our approach, SOFI, computes the L_2 norm of the loss gradient as a representation of the gradient method's value. With w_0 equal to 0, adversarial examples cannot alter the field. Conversely, w_1 and w_2, set to 1 and 2, respectively, denote fields where perturbations for adversarial example generation are permissible. If a field is independent, we assign it with the weight w_2; otherwise, with the weight w_1. Overall, SOFI first changes independent fields and then changes dependent fields. The reason is that independent fields are not affected by other fields, and dependent fields have multiple relationships in places, leading to larger perceptual differences.

SOFI calculates each field's loss gradient to determine its impact on the model's output. A larger loss gradient, obtained through backpropagation, signifies a field that significantly influences the input's classification. To enhance the efficiency of adversarial example generation, SOFI prioritizes fields with a high score in generating OS fingerprint adversarial examples. Notably, our W-FGSM can be applied in black-box attacks by using a substitute model approximating the targeted model, where crafted adversarial examples can also misclassify the original model.

B. Constrained Space Generation

The mutable region allows valid perturbations that comply with TCP/IP protocol constraints. Typically, there are three requirements for the perturbation for each field: data type, data scope, and dependent constraints. For the data type and scope, we utilize the TCP/IP specifications to obtain the format and semantics of each field. Note that this manual effort is necessary to generate legal adversarial examples of OS fingerprints. For dependent constraints, SOFI automatically constructs a constrained space of each field.

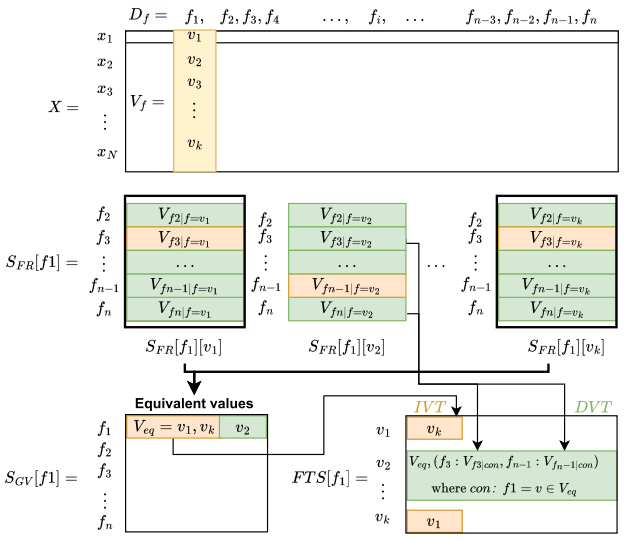

Fig. 5. The constraint relationship generation: X is the dataset, D_f is the feature set, S_{FR} is the constrained value set, S_{GV} is the equal feasible value set, and FTS is the constaint set for the feature.

Constraint Representation: As previously noted, some mutable fields are independent, while others are dependent, being constrained by or impacting other fields due to their conditional relationships. For instance, the 'tcp.flags.push' field must be set to 1 when 'tcp.len' (the length of the TCP packet payload) exceeds 0. Changing dependent fields should comply with their data scope and constraints from other fields; otherwise, it would violate the integrity of TCP/IP packets.

For constrained relations, we use the symbol *Re* to represent the conditional relationships among those fields, indicating that the value of one field is constrained by the values of other fields. We use the equation to represent *Re* as follows,

$$Re(f_i) \leftarrow \{(f_i^1) \wedge \ldots \wedge (f_i^k)\},$$

where $f_i^1, \ldots f_i^k$ are fields that have the conditional relationship with the field f_i. It presents the field's value as legal when its conditional fields are under those constraints. For instance, the equation $Re(syn = 1) \leftarrow (fin = 0) \wedge (reset = 0)$ exemplifies that when 'tcp.flags.syn' is 1 (indicating a request or confirmation to establish a connection), the values of 'tcp.flags.fin' and 'tcp.flags.reset' must be 0. Any deviation from this pattern may result in it being blocked or dropped by the router.

Constraint Generation: For each field, SOFI automatically generates its constraints ($Re(f_i)$) by mining the OS fingerprint dataset. A field refers to raw data elements, while a feature is a derived attribute used for fingerprinting. Figure 5 depicts the constraint relationship generation process: (1) we first generate a constrained value set S_{FR} from the dataset; (2) we divide S_{FR} into equivalence classes S_{GV} whenever two feature values perform a similar role within the OS class; (3)

we construct the constraint set *FTS* for the feature, utilizing both the equivalence class S_{GV} and the constrained value set S_{FR}.

Given a dataset $X = \{x_1, x_2, \ldots, x_N\}$, $D_f = \{f_1, f_2, \ldots, f_n\}$ denotes the feature set, where each feature f has a feasible value set $V_f = \{v_1, v_2, \ldots, v_m\}$. The feature constraint ($Re(f_i)$) can be represented as a set of tuples, where each tuple contains the constrained feature f' and its dependent fields, denoted as S_{FR}

$$S_{FR}[f][v] = \{(f', V_{f'|f=v}) \mid f' \in D_f \setminus \{f\}\} \tag{3}$$

where $V_{f'}$ is the value set for a feature f', and $V_{f'|f=v}$ denotes the subset of feasible values for f' when the feature f has a value v.

Given a S_{FR}, we further segment the value set of each feature into equivalence classes according to the constraint relations. Two values are considered equivalent if they exhibit identical constraint patterns, i.e., $S_{FR}[f][v_i] = S_{FR}[f][v_j]$. We denote an equivalence class of a feature using $S_{GV}[f]$ as follows:

$$S_{GV}[f] = \bigcup_{V_{\text{eq}} \subseteq V_f} \left\{ \left\{ l | l V_{\text{eq}} \begin{cases} l \bigcup_{v \in V_{\text{eq}}} V_{\text{eq}} \equiv V_f, \\ \wedge \left(\forall v', v^* \in V_{\text{eq}}, S_{FR}[f][v'] = S_{FR}[f][v^*]\right) \\ \wedge \left(\forall V_{\text{eq}1}, V_{\text{eq}2} \in S_{GV}[f], V_{\text{eq}1} \cap V_{\text{eq}2} = \emptyset\right) \end{cases} \right\} \right. \tag{4}$$

In this formulation, each V_{eq} in $S_{GV}[f]$ represents an equivalence class derived from S_{FR}. Furthermore, all V_{eq} are disjoint subsets, collectively covering all feasible subsets.

After that, we generate a feature transition set (FTS), including an independent transition set (IVT) and a dependent transition set (DVT). FTS is the candidate set to represent the transition pathway for any feature value (independent or dependent). Specifically, we use the equation 4 to generate IVT as follows.

$$IVT[f][v] = \begin{cases} V_{\text{eq}} \setminus \{v\} & \text{if } v \in V_{eq}, |V_{\text{eq}}| > 1 \\ \emptyset & \text{otherwise.} \end{cases} \tag{5}$$

$IVT[f]$ refers to the set of feature values under f that can independently change without impacting other features; DVT represents the transition relationships between v and other subsets, denoted as:

$$DVT[f][v] = \begin{cases} \{(v_i, f' \rightarrow V_{f'|f=v})\} & \text{if } v \in V_{eq}, \\ & \wedge |V_{eq}| = 1, v_i \notin V_{eq} \\ \varnothing & \text{otherwise.} \end{cases} \tag{6}$$

Here, $DVT[f][v]$ represents the dependent transition set for a value v, mapping the feature f' and its potential values $V_{f'|f=v}$, given the constraints of v. The notation $DVT[f]$ refers to feature value f with constraits of other feature values. The process to generate a FTS for all features, including both independent and dependent transitions, is described in this section using Equations 5 and 6.

Algorithm 1 SoFi: Generating Adversarial Examples for OS Fingerprints

Input : Model F, original example x, set of independent fields S_{If}, set of dependent fields S_{Df}, set of field definitions S_{Def}, set of constraints FTS

1 $x' \leftarrow x$;
2 **while** $(S_{If} \neq \emptyset$ *or* $S_{Df} \neq \emptyset)$ **do**
3 **if** $S_{If} \neq \emptyset$ **then**
4 $f \leftarrow Sort(S_{If})$ *by Equ. 2* ▷ Pick top 1
5 $x'[f] \leftarrow$ random$(S_{Def}[f])$;
6 S_{If}.delete(f);
7 **else**
8 **if** $S_{Df} \neq \emptyset$ **then**
9 $f \leftarrow Sort(S_{Df})$ *by Equ. 2* ▷ Pick top 1
10 $(v, S_{cons}) = FTS[f]$.get_substitution();
11 $x'[f] \leftarrow v$; S_{Df}.delete(f);
12 **for** $f' \in S_{cons}$ **do**
13 $x'[f'] \leftarrow S_{cons}[f']$; S_{Df}.delete(f');
14 **if** $F(x) \neq F(x')$ **then**
15 $x_{adv} \leftarrow$ UpdateCheck(x, x');
16 break;
17 **return** x_{adv}

C. Heuristic Algorithm

As aforementioned, existing adversarial example generation approaches [9], [10], [11], [12], [13] are not designed for crafting perturbations towards OS fingerprints. If we directly apply those approaches to generate adversarial examples, the integrity and validity of TCP/IP packets would be broken. Hence, we combine the W-FGSM and the field's substitution space to propose a heuristic algorithm for adding legal perturbations into the fields of packet headers. We use the independent set I_f to store all independent variable fields that are not affected by other fields, e.g., the 'ip.ttl' field. We use the dependent set D_f for all dependent fields, where *FTS* stores all legal values and conditional fields. Note that *FTS* is labeled as NULL when a field belongs to independent fields.

Algorithm 1 depicts how to generate adversarial examples of OS fingerprints under TCP/IP constraints. The algorithm's input is the target model F, original packet x, independent field set I_f, dependent set D_f, and constraint set *FTS*. Given a packet x, we first ensures that at least one of the sets I_f or D_f contains elements before proceeding with modifications. We perturb I_f by selecting the field with the highest impact according to W-FGSM (Equation 2), and then randomly choose a valid replacement from S_{Def}, the set of legal substitutions for specific fields for this top-ranked independent field (lines 1-6). Subsequent to updating I_f, we utilize W-FGSM to determine the most impactful field in D_f (line 9). Once identified, we employ *FTS* to find and apply the minimal necessary modification to this field (lines 10-13). Upon creating an adversarial example, we utilize the target model to determine whether the manipulated example successfully misleads the classification model. The overall goal is to induce misclassification by subtly altering the packet at inference time.

Updating Checked Fields: Perturbations in TCP/IP packets should guarantee the integrity and availability of checked fields. Even with a very minor modification, the checked fields would differ from their original values. For every adversarial example, SOFI would update its checked fields in order to ensure the legitimacy of OS fingerprints. Specifically, we first update the header lengths of TCP/IP packets and the total length of a packet. Then, we update the values of the TCP checksum field and the IP checksum field. The checksum updating is as follows: (1) we assign 0 to the TCP checksum; (2) we divide the string (the header and data parts of the TCP segment) into groups of 16 bits (2 bytes); (3) we calculate the sum of every group as a 16-bit result; (4) we calculate the sum of complement code on each group; and (5) the result is a new checksum.

IV. IMPLEMENTATION

In this section, we have implemented ML-based and DL-based OS classifiers to represent OS fingerprinting techniques. We have implemented SOFI to generate adversarial TCP/IP packets for deceiving the OS fingerprinting.

OS Fingerprinting Based on Classic Machine Learning: There are many fields in the TCP/IP packet header, where some fields are distinguishable, and others are irrelevant for OS fingerprints. TCP/IP packets have limited representational capacity, so classic machine learning cannot learn complex dependencies between features of the input data. To build a multi-classifier, we have conducted feature engineering to pick meaningful information about the input representation. We use the Chi^2 Test [43] approach to select features for OS fingerprints, where all features are ranked in descending order. The value ranges of fields in the TCP/IP packet header vary greatly, affecting the stability of the OS fingerprint. We employ data standardization to bring these values to a consistent scale. This standardization involves transforming the original data through the following formula:

$$z = (x - \mu)/\sigma$$

where z is the standardized data, x represents the original data, μ is the mean of the original data, and σ represents the standard deviation of the original data. Different variables' value ranges and distributions are converted to the same scale. Specifically, we write a Python script to extract field values of the packet header and take the Chi^2 test as the statistical metric to select fields for classic ML-based OS fingerprinting.

We use the open-source scikit-learn [44] to implement four classic OS classifiers, including KNN (K-Nearest Neighbors), SVM (Support Vector Machine), RF (Random Forest), and DT (Decision Tree). During the training, the KNN algorithm uses 5 as the number of samples. Our SVM algorithm utilizes a radial basis function (RBF) as its kernel, with a penalty coefficient set to 1.0. We use 10 as the number of decision trees and the entropy as the quality function for the RF algorithm. The DT algorithm uses 2 as the number of samples, 1 as the minimum number of leaf nodes, and an unlimited maximum tree depth.

OS Fingerprinting Based on Deep Learning: Unlike classic OS fingerprinting, there is no need to conduct feature engineering to extract the underlying statistical patterns between the input data and the output. Deep Learning algorithms use multiple layers of neural networks to extract feature representations of the input data. In each layer, the neuron unit performs a nonlinear transformation on the input and output to other neuron units in the next layer after calculation through a series of mathematical operations and activation functions. Every layer can be represented as follows,

$$h_l = f_l(W_l h_{(l-1)} + b_l),$$

where h_l represents the output of neurons in layer L, f_l is the activation function, W_l and b_l are the weights and biases of this layer. Thus, we do not use feature engineering or data standardization techniques for deep learning algorithms.

We use the PyTorch [45] library to implement three neural network structures for OS fingerprints, including Deep Neural Network (DNN), Recurrent Neural Network (RNN), and Convolutional Neural Network (CNN). The CNN consists of three layers: convolutional layers, pooling layers, and fully connected layers. The DNN contains three fully connected layers. The RNN uses two Long Short-Term Memory Networks (LSTM) and a fully connected layer. During the training, the loss function of the DL-based OS classifier is Cross Entropy Loss, and the optimizer uses the Adam optimization algorithm.

SOFI *Implementation:* We implement a prototype of SOFI as a self-contained piece of software based on open-source libraries. The W-FGSM and heuristic algorithm in SOFI are pipelined via custom Python scripts. We deploy the SOFI prototype system on a Ubuntu 20.04.3 LTS server powered by four Intel Xeon Gold 6230 CPUs, 125GB RAM, and four 24GB NVIDIA GPUs.

V. EVALUATION

In this section, we first present the experimental settings of SOFI. Then, we evaluate the performance of SOFI and compare it with the baseline approaches. We use a surrogate OS classifier to demonstrate the SOFI's transferability. We compare SOFI with a variety of existing OS fingerprinting defense tools. Further, we shed light on the explainability of OS fingerprints.

A. Experimental Settings

Dataset: We use two datasets to validate SOFI's performance, including one passive OS fingerprinting dataset and one active OS fingerprinting dataset. The passive dataset is from the CIC-IDS (Canadian Institute for Cybersecurity Intrusion Detection System) [46], which contains 48,962 packets and covers eight OS fingerprints with corresponding versions. Each sample in the passive dataset is a TCP/IP packet. The active dataset is from the open-source repository [47], which contains 264,852 packets and covers seven OS fingerprints. Each sample in the active dataset has several packets used to calculate the feature of active OS fingerprints, the same as the NMAP [3] OS dataset.

Table II lists the distribution of each OS and the number of packets per category. The passive dataset is imbalanced, where Ubuntu server 12 is the class with a small proportion of

TABLE II
TWO DATASETS: ONE PASSIVE AND ONE ACTIVE OS FINGERPRINTS

Passive OS	No. of Packets.	Active OS	No. of Packets
Ubuntu 14.4	6,558	Linux	142,548
Ubuntu 16.4	410	Windows	67,128
Ubuntu server 12	63	iOS	17,232
Win 10	23,059	macOS	16,160
Win 7	6,636	BSD	14,278
Win 8.1	9,323	Solaris	4,926
Win Vista	1,493	Android	2,580
macOS	1,420		

packets, and Win 10 is the class with a half dataset. The active dataset is also imbalanced, where Android has the lowest number of packets. We use the ratio (8:2) to divide the training and test sets for evaluating OS classifiers.

Metric: To evaluate the performance of adversarial example generation, we use three metrics to measure the effectiveness and performance of SOFI. (1) *Precision/Recall/F1* are statistics indicators to measure the performance of OS fingerprinting. (2) *Success Rate (SR)* is to measure the degree of deceiving OS fingerprinting. SR is equal to the proportion of samples that make the OS fingerprints misclassified to other classes. The higher the SR, the more effective the generated adversarial examples. (3) *Constraint Requirement (CR)* indicates whether an adversarial example satisfies TCP/IP specifications.

B. SoFi 's Performance

Performance of OS Fingerprinting: We measure the precision and recall of passive OS fingerprinting determined by KNN, SVM, RF, DT, CNN, DNN, and RNN. The performance of active OS fingerprints is also evaluated, demonstrating promising results for recognizing OS information. We observe that both ML and DL-based OS classifiers demonstrate comparable performance with precision and recall soaring to around 97%-99%. The performance of ML-based OS classifiers might be attributed to the feature engineering that provides the CHi-Squared test and data standardization to pick up features. The experimental results show that active fingerprints achieve promising performance for recognizing OS information, e.g., close to 96% F1-score. One interesting finding is that despite the dataset's imbalance, for instance, Ubuntu server 12 having only 63 packets, the performance of OS classifiers does not degrade. This might be due to the efficient handling of imbalanced data or the robustness of the algorithms in handling such variations.

Performance of Adversarial Fingerprints: SOFI generates adversarial examples at inference time in order to induce a misclassification for OS fingerprinting. We employ three Deep Learning (DL) OS models as the target models, which SOFI attempts to deceive through adversarial examples generated from original packets. We use four metrics to measure the performance degradation of OS models, including F1-score, accuracy, ∇, and SR. Note that ∇ represents the difference in the model's accuracy between inputs of original packets and those of adversarial examples. The higher the ∇, the larger the performance degradation of the model. (1) *Passive OS fingerprints*. Table III lists the SOFI's performance over passive OS fingerprints: DNN, CNN, and RNN. We observe that the generated adversarial examples can greatly reduce the classification performance of the model. On average, the accuracy of the model can be reduced to about 10%, and the ∇ value achieves nearly 90%. In addition, their F1 scores also decrease below 0.2. We can see that the accuracy rate of the RNN model has the largest descent, from 97% to 6.23%, followed by the DNN model and CNN model. (2) *Active OS fingerprints*. Table IV lists the SOFI's performance over active OS fingerprints: DNN, CNN, and RNN. Specifically, the model's F1-score decreases from 96% to 0.0%, and the accuracy decreases from 95.8% to 0.0%. Their ∇ values are equal to 1.00. Overall, adversarial examples of active OS fingerprints achieve 96.2% SR. We verify that those generated adversarial examples comply with the TCP/IP specifications. All adversarial examples are legal, including their checked fields and constrained fields.

TABLE III
THE SOFI 'S PERFORMANCE ON PASSIVE OS FINGERPRINTS: THE F1 SCORE, ACCURACY, ∇, AND SR

Target Model	F1	Accuracy	∇	SR
DNN	0.2820	0.1608	0.8092	83.92%
RNN	0.1231	0.0623	0.9077	93.77%
CNN	0.1501	0.0812	0.8888	91.880%

TABLE IV
THE SOFI 'S PERFORMANCE ON ACTIVE OS FINGERPRINTS: THE F1 SCORE, ACCURACY, ∇, AND SR

Target Model	F1	Accuracy	∇	SR
DNN	0.0000	0.0000	1.0000	96.19%
RNN	0.0000	0.0000	1.0000	96.30%
CNN	0.0000	0.0000	1.0000	96.23%

Comparison: Specifically, we implement five baseline methods on the passive dataset:

1) *FGSM* [9] linearizes the model's cost function around the input to select perturbations by differentiating the cost function with respect to the input itself, setting the perturbation amplitude at 0.1.
2) *BIM* [13] generates adversarial examples by iteratively adding small perturbations to the input data, with 100 iterations and a perturbation amplitude of 0.1.
3) *C&W* [11] minimizes the cost function through an optimization algorithm, with a learning rate of 0.01 and L_∞ as the distance metric.
4) *DeepFool* [10] employs an iterative optimization algorithm with a linear approach, using 100 iterations and a numerical stability parameter of 10^{-6}.
5) *JSMA* [12] uses a Jacobian matrix to evaluate the model's output sensitivity to each input, acting as a

TABLE V
PASSIVE OS FINGERPRINTS: THE COMPARISION PERFORMANCE BETWEEN SOFI AND 5 BASELINES

	SR	CR	Violation Scope	Violation Constraint
FGSM [9]	52.61%	No	100%	100%
BIM [13]	57.09%	No	100%	100%
DeepFool [10]	94.79%	No	100%	100%
C&W(L_∞) [11]	82.52%	No	100%	100%
JSMA [12]	48.91%	No	100%	100%
SOFI	91.71%	Yes	0.00%	0.00%

TABLE VI
PASSIVE OS FINGERPRINTS: THE COMPARISION PERFORMANCE BETWEEN SOFI AND 5 BASELINES

	SR	CR	Violation Scope	Violation Constraint
FGSM [9]	14.32%	No	100%	100%
BIM [13]	1.47%	No	100%	100%
DeepFool [10]	87.80%	No	100%	100%
C&W(L_2) [11]	93.27%	No	100%	100%
SOFI	96.19%	Yes	0.00%	0.00%

saliency map, with a maximum perturbation rate of 0.1 per iteration.

Adversarial examples are crafted to exploit the features most sensitive to classification outcomes. The active OS fingerprint contains a large feature dimension (233), and the passive OS fingerprint contains a small dimension (32). We set the maximum perturbation rate at 0.2. During implementation, the batch size is set to 128, and all fields can be perturbed by all baselines.

Table V lists the comparison results between SOFI and five baseline approaches for passive OS fingerprints. We find that SOFI and DeepFool both achieve a higher SR than the other four baseline methods: DeepFool has 94.79%, and our generated adversarial examples have a 91.71% success rate for spoofing OS fingerprinting. The SR of the C&W method is 82.52%, which is lower than that of SOFI. The strict conditions of the L_∞ norm result in fewer fields perturbed by the C&W method. The SR of FGSM and BIM are relatively low, 52.61% and 57.09%, respectively. JSMA has the lowest SR value nearly 48.91%, because the features in the Jacobian matrix are constrained by other features. These results show that the OS fingerprinting model is vulnerable to adversarial examples.

Table VI lists the comparison results between SOFI and five baseline approaches for active OS fingerprints. SOFI achieves a higher SR than the other five baseline methods, with a success rate of 96.19%. C&W(L_2) has the second highest SR, with a success rate of 93.27%, and DeepFool has 87.8% SR. The other 3 baselines show a lower SR.

All five baselines violate the requirements of the TCP/IP protocol, producing invalid or illicit adversarial examples. We conduct an in-depth analysis of the violations present in adversarial examples, distinguishing between data scope violations, where the perturbation falls outside the field's value range, and constraint violations, where the perturbation conflicts with the values of other fields. Ensuring TCP/IP compliance is a fundamental requirement for adversarial example generation methods in the network domain, as non-compliance results in invalid packets that are dropped by network devices. SOFI distinguishes itself by achieving compliance while maintaining a high obfuscation success rate, addressing the limitations of prior methods that produce invalid packets due to domain-specific constraints.

TABLE VII
SOFI 'S TRANSFERABILITY OVER DIFFERENT TARGET MODELS: THE SUBSTITUTION MODEL IS RNN

	F1	Accuracy	▽		F1	Accuracy	▽
DNN	0.18	0.10	0.88	SVM	0.00	0.00	0.96
CNN	0.20	0.11	0.87	RF	0.00	0.00	0.99
KNN	0.00	0.00	0.97	DT	0.06	0.03	0.96

TABLE VIII
SOFI 'S TRANSFERABILITY ON OS FINGERPRINTS: THE TRAINING DATASET IS UNKNOWN WITH RNN

	Acc., Pre.	SR	Change Rate
Data Augmentation	0.97, 0.98		
SOFI	0.00, 0.01	97.5%	0.04
RNN Transfer	0.06, 0.12	91.9%	0.04

C. SoFi 's Transferability

To further assess SOFI's efficacy, we validate its transferability under the black-box setting with various ML algorithms. In the black-box setting, the target model (e.g., the architecture, the loss function, or parameters) is unknown to SOFI, which presents a realistic situation where an OS fingerprinting method is often inaccessible. The goal is to generate adversarial examples via the surrogate model that lead to misclassifications in the original target models.

We leverage an RNN model as a surrogate to approximate the other 6 target models. We use the same training dataset to learn the RNN model. Table VII lists the performance degradation of OS fingerprinting by the other 6 models. We observe that adversarial examples from the RNN model can still deceive the OS fingerprints learned by other models. F1-score and accuracy of those models are greatly reduced, where the average ▽ is closely 0.94. For classic ML-based OS fingerprinting, the F1-score and accuracy of KNN, SVM, and RF drop into zero by our adversarial examples. DT-based OS fingerprinting has a certain resistance towards adversarial examples: the pruning operation in DT might eliminate noises or adversarial examples. We find that DL-based fingerprints are slightly less affected than ML-based fingerprinting. The reason is that extracting high-level features from the data might make the model more robust to small perturbations in the input.

Further, we evaluate the SOFI's transferability on the active and passive OS fingerprints with a different dataset from the training data. Specifically, we leverage the data augmentation

TABLE IX
SoFi 's Robustness: The Adversarial Training of OS Fingerprints With DNN Model

	Acc.	Pre.	SR	Change Rate
Original training	0.96	0.967		
SoFi	0.00	0.00	96.2%	0.18
Adversarial training	0.96	0.97		
SoFi	0.00	0.00	96.4%	0.64

to generate new data from the training data. Given the original 264,852 packets in seven OS fingerprints, there are 26,494 new samples as the training data. The surrogate model has also adopted an RNN algorithm, and the target model is DNN. Table VIII lists the SoFi's performance. SoFi achieves a high SR of 97.5% for the active dataset, and the change rate is 0.04. When we use RNN as the target model of DNN, the performance is 91.9% SR for adversarial fingerprints. Overall, SoFi can still generate adversarial examples to obfuscate OS fingerprints when we have little knowledge of the training dataset. These results indicate that SoFi has transferability across models and datasets.

D. SoFi 's Robustness

We use the adversarial training to validate the SoFi's robustness. Adversarial training is a general method that uses adversarial examples as the supplement of the training data [9]. Specifically, generated adversarial examples are added to the original training dataset. We only keep successful samples and remove failed samples. The training data set is divided into the same ratio of 8:2 for learning OS classifiers.

Table IX lists the SoFi's performance with/without the adversarial training data. We observe that the performance of the OS classifier does not obtain any improvement with adversarial training, where its accuracy remains at a similar level as the original OS classifier. The plausible reason is that adversarial examples do not bring more context and semantic information than the original training data. We observe that SoFi can still generate adversarial examples with a high SR towards both classifiers with/without adversarial training. The improvement lies in the change rate between original and adversarial examples, which increases from 0.18 to 0.64. A high change rate indicates a large cost of SoFi. Note that adversarial training is expensive due to the iterative generation of adversarial examples during the training. In short, adversarial training with adversarial examples cannot improve OS fingerprint performance but increase fingerprints' robustness.

VI. Automation and Deployment

In this section, we demonstrate SoFi's benefits from two aspects: the comparison with existing defense tools, and the comparison of manual versus automation.

A. Comparison With Defense Tools

We compare SoFi with the existing defense tools, including a random approach and OSfooler [28]. The manual approach is to select a TCP/IP header field and change its value based on expert experience. OSfooler [28] leverages the OS fingerprints in the Nmap database to change the response packets. It picks up a different OS fingerprint in the Nmap database and modifies the packet based on the selected OS fingerprint. Those approaches rely on expert knowledge and manual efforts. Table X lists the overall performance comparison between SoFi and other approaches. SoFi has a 91.71% SR in deceiving OS fingerprinting, while Manual has only 44.91% SR and OSfooler has 92.13% SR. Generated adversarial packets have zero violation constraint, compared with 86.62% violation constraint in Manual and 22.59% violation constraint in OSfooler. One advantage is that SoFi can automatically generate adversarial packets without any manual efforts and professional knowledge. We observe that SoFi archives a promising performance for misleading today's OS fingerprinting.

TABLE X
The Comparison Performance Between SoFi and 2 Baselines

	SR	CR	Avg. Perb. (%)	Violation (%)
Manual	44.91%	No	4.99	22.59%
OSfooler [11]	92.13%	No	1.00	86.62%
SoFi	91.71%	Yes	1.00	0.00%

TABLE XI
Comparison Between SoFi and Exsint Tools and Works

Defense	Properties			
	MT	OC	DE	FS
Ipersonality [29]	○	○	○	○
Honeyd [48]	◑	●	●	○
OSfuscate[28]	○	○	○	○
OSfooler [30]	○	○	○	○
MTD [33]	●	○	◑	○
TMorph [49]	○	○	◑	○
SoFi	●	○	●	●

Maintenance(MT) - ● Actively ◑ Occasionally ○ No Recent
OS Compatibility(OC) - ● OS-Independent ○ OS-Dependent
Deployment Effort(DE) - ● Easy ◑ Moderate ○ Challenging
Feature Selection(FS) - ● Automatic ○ Manual

Then, we compare SoFi with existing tools and research works. Table XI lists a qualitative comparison between SoFi and other approaches. Note that there is a compatibility issue between SoFi and those works, e.g., OSfuscate works in Windows XP/Vista. Thus, we only use four metrics to represent their advantages and disadvantages, including MT, OC, DE, and FS. Overall, SoFi can automatically find modifiable features for adversarial packets, adapting to the new and emerging OS fingerprinting techniques.

B. Manual Vs. Automation

SoFi automatically finds applicable and available features to deceive OS fingerprints. By contrast, the manual approach leverages expert knowledge to identify which feature impacts

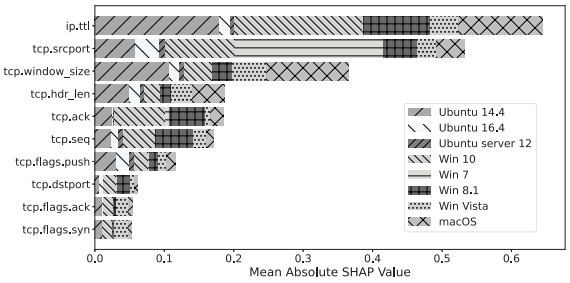

Fig. 6. The distribution of features' SHAP values contribution to the OS fingerprints.

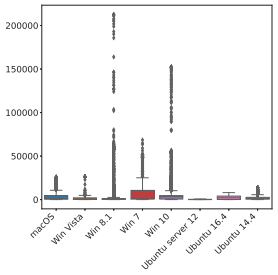

Fig. 7. The distribution of TCP SEQ of different OS fingerprints.

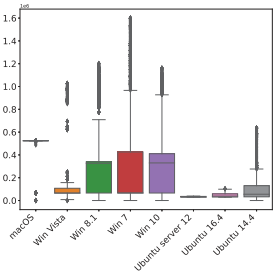

Fig. 8. The window size of different OS fingerprints.

OS fingerprints. We use SHAP [50] to represent a manual approach for obfuscating OS fingerprints, which manually seeks important features. SHAP leverages the cooperative game theory concept to interpret the model predictions. SHAP provides insight into the importance of features in classifier decisions and the direction of that influence. Specifically, SHAP explains the final value of a prediction by attributing a value to each feature based on its contribution to OS fingerprints, as follows:

$$g(x) = \theta_0 + \sum_{j=1}^{M} \theta_j \cdot f_j,$$

where x is the packet, f_j is the jth field, and θ_j is the contribution of feature f_j to the OS fingerprint.

Figure 6 depicts the distribution of features' SHAP values based on the contribution to the OS fingerprints. We only pick the top 10 features' SHAP values for OS fingerprints. Note that the sum of SHAP values across all features of a given sample equals the model's logit output, convertible to a probability via logistic transformation. Our analysis reveals that the TTL field, source port, and window size are the top 3 influential features in the OS fingerprinting model. We observe that a TCP/IP packet feature plays varying roles in affecting the model's classification decision. For instance, the TTL field significantly impacts Ubuntu 14.4, Win 8.1, Win 10, and macOS predictions but is less influential for other OS classes. We explain that each feature's SHAP value approximates the confidence of the decision boundary. From a defensive perspective, the SHAP value of a feature offers a model-agnostic insight for researchers. Meanwhile, from an attacker's viewpoint, the SHAP value of a feature can guide the creation of perturbations, thereby affecting the generalizability of adversarial examples. Our approach automatically identifies features like window size, which is consistent with the SHAP analysis results.

We provide the feature analysis of various fields from the TCP/IP packet header. Our analysis reveals distinct feature distributions among different OS versions. Figure 7 illustrates a relatively flat and stable 'tcp.seq' distribution across OS fingerprints, with Windows 10 and 8 displaying the most significant fluctuation. By contrast, other OS fingerprints cannot be distinguished based on the 'tcp.seq' feature. Figure 8 shows similar window size distributions between Windows 7 and 8, with Ubuntu and MacOS maintaining narrow and fixed ranges, respectively. It is evident that the window size achieves better distinguishability among OS fingerprints than the 'tcp.seq'. Note that the feature analysis also provides insights for spoofing OS fingerprints. SOFI can automatically find those features for generating adversarial packets, which is more efficient than manual efforts.

TABLE XII
PACKET HEADER FIELDS BEFORE AND AFTER MODIFICATION

Field	Original	Adversarial
	Ubuntu 23.04	Windows
Immutable Region		
Mutable Region		
ip.flags.df	1	0
tcp.flags.ack	1	1
tcp.window_size	65160	(65535/8192/4128)
tcp.flags.fin	0	0
ip.ttl	64	(128 / 255)
tcp.flags.reset	0	0
tcp.flags.push	0	1
tcp.flags.urg	0	0
tcp.flags.syn	1	1
Checked Region		

C. Real-World Deployment

We deploy adversarial OS fingerprints to validate SOFI effectiveness. Our original OSes are Ubuntu 20.04 and Ubuntu 23.04, and the target OSes are Windows Vista and Windows. In the passive dataset, we have 1,581 pairs of original and adversarial OS fingerprints, abbreviated as (Ubuntu, Win Vista); in the active dataset, we have 8,133 pairs (Ubuntu, Win). We obtain packet header differences for those pairs, detailed in Table XII.

Figure 9 depicts the implementation of adversarial OS fingerprints in a real-world scenario. Leveraging existing tools or libraries to change TCP/IP packet headers, we strategically modify packet header fields such as IP TTL, TCP window

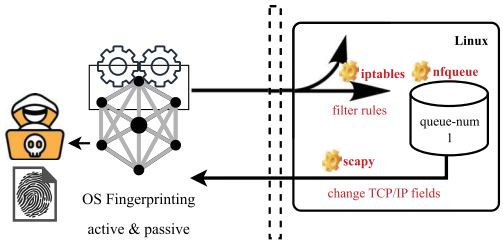

Fig. 9. The deployment for adversarial OS fingerprints.

size, and TCP flags. Specifically, the iptable tool configures the kernel's firewall to divert TCP SYN packets into a designated queue,[1] and the NetfilterQueue tool accesses the packet queue, where we can manipulate packets in user-space.[2] After that, we leverage the Scapy tool to modify packet header fields, including TTL, window size, and sequence numbers. We use the Tshark tool to capture the packet flows and identify whether adversarial packets can spoof OS fingerprints. Our deployment for adversarial OS fingerprint only uses Python scripts to change the packet headers, which is efficient and effective.

VII. Discussion and Limitation

Compatibility Limitation: One concern is that SoFI has a compatibility issue with existing fingerprinting defense tools [28], [29], [30]. Changing the TCP/IP packet header must involve OS kernel operations, leading to the comparability limitation. In fact, those fingerprinting defense tools [28], [29], [30] are also not compatible with each other; for instance, OSfuscate works in Windows XP/Vista, and OSfooler works in Ubuntu. We have deployed adversarial OS fingerprints in the Ubuntu system with several open-source libraries or tools for validating SoFI's effectiveness.

Usage: SoFI acts as the guideline to help fingerprinting defense tools find available modification places in the TCP/IP packets. Prior works [27], [40], [51] used expert knowledge to defeat OS fingerprinting. A straightforward method is to revise the TTL field in the IP header, which depends on the specific OS implementation. Note that this manual process is arduous and incomplete, as it relies on only one tool to guess the modification operation. When attackers deploy new tools or techniques, the existing defense approaches may not be effective against a wide range of new OS fingerprinting methods. By contrast, SoFI only relies on a target model for generating adversarial packets and provides the guideline information to existing defense tools/approaches.

Targeted Obfuscation: We use a multi-classifier to represent the OS fingerprinting and focus on non-targeted evasion. The targeted evasion misclassifies the OS fingerprinting into a specific class, and the non-targeted evasion merely induces an incorrect prediction. For example, when the target machine runs an Ubuntu 12.4, the targeted evasion would deceive attackers into an Ubuntu 16.2 version. Targeted approaches can mislead attackers and potentially force them to adopt ineffective strategies. Now, SoFI adopts an obfuscation approach, aiming to make OS fingerprints unidentifiable rather than mimicking a specific OS. In our future work, we will extend SoFI into the targeted evasion.

Model Limitation: First, we are free to access the training data of OS fingerprints due to its public availability. Second, we explore SoFI's effectiveness with limited access to the target model, which is referred to as model transferability. SoFI can access the architecture and loss function of the surrogate model. SoFI with the surrogate model still achieves a high success rate in deceiving the target model. Third, our SoFI is evaluated for these adversarial examples against both classic machine learning and deep learning algorithms. In our future work, we will add existing fingerprinting tools as the surrogate model for deceiving attackers.

Coverage: One concern is that our fingerprinting model suffers a coverage issue because there are a variety of OS fingerprinting tools and techniques (see Section II). It is impossible to find a single model that satisfies all of them. Another concern is that the experimental dataset only covers a limited number of OS versions, not representing the diversity of OS versions in practice. In future work, we will collect a comprehensive dataset to generate adversarial OS fingerprints, including OS variants and new versions.

Additional Features: SoFI currently does not support some features used in advanced fingerprinting, such as TCP flag combinations (e.g., FIN, XMAS), ICMP responses, and fragmentation. These features are also utilized by OS fingerprinting tools like Nmap and p0f. Modifying these features requires specialized knowledge and manual effort, as users must know which features to adjust and what values to change. Additionally, altering these features in practice can be challenging—such as manipulating ACK or FIN numbers in host communications. As a result, SoFI focuses on leveraging available and modifiable features for OS fingerprinting.

IPv6 Extension: So far, SoFI does not support the OS fingerprints to IPv6. IPv6 offers a number of distinct and new features for OS fingerprints. IPv6 packets have different fields to IPv6, like the Hop-by-Hop Options or Routing Header. In contrast, some features between IPv6 and IPv4 are similar, such as certain TCP flags, time-to-live (TTL) values, or window sizes. To extend SoFI to IPv6, several things need to be done: (1) manually identifying features in IPv6 packets, (2) collecting the IPv6 dataset, (3) dividing features into 3 categories (Table I), and (4) finding perturbations in the IPv6 packets for OS fingerprints.

Long-Term Viability: One concern is that SoFI may not be viable in the long term due to the rapid evolution of OS fingerprinting techniques and the increasing complexity of network environments. Specifically, SoFI excels in static network environments, it may face challenges when confronted with dynamic network conditions or adaptive fingerprinting systems. The modification is that we put the new dataset into SoFI to generate adversarial OS fingerprints. In our future work, we will extend SoFI for advanced algorithms to incorporate tool-specific adaptations or optimize obfuscation strategies for specific reconnaissance scenarios.

[1] iptables -A INPUT -p tcp –syn -j NFQUEUE –queue-num 1.
[2] nfqueue.bind(1, process_packet).

VIII. Conclusion

As fingerprinting techniques play an important role in network reconnaissance, we propose a novel approach called SoFi to spoof OS fingerprints as a proactive defense measure. The core of SoFi is an adversarial example generation algorithm. Uniquely designed, SoFi works within TCP/IP specification constraints and introduces effective perturbations in packet data to deceive OS fingerprints classifiers. Our research underscores these classifiers' vulnerability to evasion and reveals how subtle perturbations can successfully spoof OS fingerprints. Our results demonstrate the effectiveness of adversarial examples and assess the impact of model transferability on attackers. Additionally, we employ ML explainability to pinpoint potent features, enhancing the robustness of OS fingerprints.

References

[1] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, Dec. 2019.

[2] F. Cremer et al., "Cyber risk and cybersecurity: A systematic review of data," *Geneva Papers Risk Insurance-Issues Pract.*, vol. 47, no. 3, pp. 698–736, 2022.

[3] Nmap.(1997). *Network Security Scanner Tool*. [Online]. Available: http://nmap.org/projects/iptables/index.html

[4] J. Caballero, S. Venkataraman, P. Poosankam, M. G. Kang, D. Song, and A. Blum, "FiG: Automatic fingerprint generation," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, San Diego, CA, USA, Jan. 2007.

[5] Z. Shamsi, D. B. H. Cline, and D. Loguinov, "Faulds: A non-parametric iterative classifier for internet-wide OS fingerprinting," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2339–2352, Oct. 2021.

[6] J. Varmarken, J. Al Aaraj, R. Trimananda, and A. Markopoulou, "FingerprinTV: Fingerprinting smart TV apps," *Proc. Privacy Enhancing Technol.*, vol. 2022, no. 3, pp. 606–629, Jul. 2022.

[7] C. Szegedy et al., "Intriguing properties of neural networks," Presented at the 2nd Int. Conf. Learn. Represent. (ICLR), Banff, Canada, Banff, AB, Canada, Apr. 2014.

[8] N. Papernot, P. McDaniel, A. Swami, and R. Harang, "Crafting adversarial input sequences for recurrent neural networks," in *Proc. MILCOM IEEE Mil. Commun. Conf.*, Nov. 2016, pp. 49–54.

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

[10] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A simple and accurate method to fool deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2574–2582.

[11] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 39–57.

[12] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Eur. Symp. Secur. Privacy*, Mar. 2016, pp. 372–387.

[13] J. Wang, "Adversarial examples in physical world," in *Proc. IJCAI*, 2021, pp. 4925–4926.

[14] D. E. Comer and J. C. Lin, "Probing TCP implementations," in *Proc. Usenix Summer*, Jun. 1994, p. 17.

[15] P. Auffret, "SinFP, unification of active and passive operating system fingerprinting," *J. Comput. Virol.*, vol. 6, no. 3, pp. 197–205, Aug. 2010.

[16] B. Proxy. (2013). *SINFP3: A Passive and Active Os Fingerprinting Tool*. [Online]. Available: https://metacpan.org/dist/Net-SinFP3/view/bin/sinfp3.pl

[17] M. Zalewski. (2013). *P0F: A Passive TCP/IP Stack Fingerprinting Tool*. [Online]. Available: http://lcamtuf.coredump.cx/p0f3/

[18] C. Sarraute and J. Burroni, "Using neural networks to improve classical operating system fingerprinting techniques," 2010, *arXiv:1006.1918*.

[19] Z. Shamsi, A. Nandwani, D. Leonard, and D. Loguinov, "Hershel: Single-packet OS fingerprinting," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2196–2209, Aug. 2016.

[20] M. Laštovička, M. Husák, P. Velan, T. Jirsík, and P. Čeleda, "Passive operating system fingerprinting revisited: Evaluation and current challenges," *Comput. Netw.*, vol. 229, Jun. 2023, Art. no. 109782.

[21] D. H. Hagos, A. Yazidi, Ø. Kure, and P. E. Engelstad, "A machine-learning-based tool for passive OS fingerprinting with TCP variant as a novel feature," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3534–3553, Mar. 2021.

[22] S. Zander and S. J. Murdoch, "An improved clock-skew measurement technique for revealing hidden services," in *Proc. 17th Conf. Secur. Symp.*, Berkeley, CA, USA, Jul. 2008, pp. 211–225. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496711.1496726

[23] G. Shen, J. Zhang, A. Marshall, L. Peng, and X. Wang, "Radio frequency fingerprint identification for LoRa using spectrogram and CNN," in *Proc. IEEE INFOCOM*, May 2021, pp. 1–10.

[24] X. Ma et al., "Context-aware website fingerprinting over encrypted proxies," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2021, pp. 1–10.

[25] Tenable.(2017). *Nessus Vulnerability Scanner*. [Online]. Available: https://www.tenable.com/products/nessus

[26] B. Anderson and D. McGrew, "OS fingerprinting: New techniques and a study of information gain and obfuscation," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2017, pp. 1–9.

[27] M. Smart, G. R. Malan, and F. Jahanian, "Defeating TCP/IP stack fingerprinting," in *Proc. 9th USENIX Secur. Symp. (USENIX Secur.)*, 2000.

[28] A. Crenshaw. (2008). *Change Your Windows Os Tcp/ip Fingerprint To Confuse P0f, Networkminer, Ettercap, Nmap, and Other Os Detection Tools*. [Online]. Available: http://www.irongeek.com/i.php?page=security/osfuscate-change-your-windows-os-tcp-ip-fingerprint-to-confuse-p0f-networkminer-ettercap-nmap-and-other-os-detection-tools

[29] I. Personality. (2013). *The Emulation of Other OSes at the Network Level*. [Online]. Available: https://sourceforge.net/projects/ippersonality/

[30] (2019). *Preventing Remote Active/passive OS Fingerprinting By Tools*. [Online]. Available: https://github.com/segofensiva/OSfooler-ng

[31] N. Provos, "Honeyd-a virtual honeypot daemon," in *Proc. 10th DFN-CERT Workshop, Hamburg, Germany*, vol. 2, 2003, p. 4.

[32] Honeypot Website.(2015). *The Honeynet Project*. [Online]. Available: https://www.honeynet.org/

[33] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati, "A survey of moving target defenses for network security," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1909–1941, 3rd Quart., 2020.

[34] M. Nasr, A. Bahramali, and A. Houmansadr, "Defeating DNN-based traffic analysis systems in real-time with blind adversarial perturbations," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 2705–2722.

[35] S. Shan, A. N. Bhagoji, H. Zheng, and B. Y. Zhao, "Patch-based defenses against web fingerprinting attacks," in *Proc. 14th ACM Workshop Artif. Intell. Secur.*, Nov. 2021, pp. 97–109.

[36] L. Qiao, B. Wu, S. Yin, H. Li, W. Yuan, and X. Luo, "Resisting DNN-based website fingerprinting attacks enhanced by adversarial training," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 5375–5386, 2023.

[37] B. Hayden, T. Walsh, and A. Barton, "Defending against deep learning-based traffic fingerprinting attacks with adversarial examples," *ACM Trans. Privacy Secur.*, vol. 28, no. 1, pp. 1–23, Feb. 2025.

[38] Z. Ling, G. Xiao, W. Wu, X. Gu, M. Yang, and X. Fu, "Towards an efficient defense against deep learning based website fingerprinting," in *Proc. IEEE Conf. Comput. Commun.*, May 2022, pp. 310–319.

[39] P. Kampanakis, H. Perros, and T. Beyene, "SDN-based solutions for moving target defense network protection," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw.*, Jun. 2014, pp. 1–6.

[40] M. Albanese, E. Battista, and S. Jajodia, "A deception based approach for defeating OS and service fingerprinting," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Sep. 2015, pp. 317–325.

[41] M. A. Rahman, M. G. M. M. Hasan, M. H. Manshaei, and E. Al-Shaer, "A game-theoretic analysis to defend against remote operating system fingerprinting," *J. Inf. Secur. Appl.*, vol. 52, Jun. 2020, Art. no. 102456.

[42] N. Mathews, J. K. Holland, S. E. Oh, M. S. Rahman, N. Hopper, and M. Wright, "SoK: A critical evaluation of efficient website fingerprinting defenses," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2023, pp. 969–986.

[43] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer, 2006.

[44] Scikit-learn. (2007). *A Free Software Machine Learning Library for the Python Programming Language*. [Online]. Available: http://scikit-learn.org/stable/index.html

[45] Pytorch.(2018). *An Open Source Machine Learning Framework that Accelerates the Path From Research Prototyping To Production Deployment*. [Online]. Available: https://pytorch.org/

[46] CIC-IDS. (2017). *Canadian Institute for Cybersecurity Intrusion Detection System*. [Online]. Available: https://www.unb.ca/cic/datasets/ids-2017.html
[47] R. P. Jove. (2015). *Applying Artificial Intelligence To Os Fingerprinting Nmap*. [Online]. Available: https://github.com/rubenpjove/osfingerprintingia
[48] C. Valli, "Honeyd-a OS fingerprinting artifice," in *Proc. 1st Austral. Comput., Network Inf. Forensics Conf.*, Perth, WA, Australia, Nov. 2003.
[49] Z. Xu, H. Khan, and R. Muresan, "TMorph: A traffic morphing framework to test network defenses against adversarial attacks," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2022, pp. 18–23.
[50] S. Lundberg and S. Lee, "A unified approach to interpreting model predictions," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2017.
[51] D. B. Berrueta, "A practical approach for defeating nmap OS-fingerprinting," *Retrieved March*, vol. 12, p. 2009, Jan. 2003.

Xu Han received the bachelor's degree from Beijing University of Chemical Technology in 2018 and the Ph.D. degree from Beijing Jiaotong University, China, in 2024. Her research interests lie in trustworthy and interpretable Al technologies for cybersecurity applications.

Haocong Li received the master's degree from the School of Computer Science and Technology, Beijing Jiaotong University, in 2024. She is currently with China Everbright Bank. Her research interests include the IoT security and software supply chain security.

Wei Wang (Member, IEEE) received the Ph.D. degree from Xi'an Jiaotong University in 2006. He is currently a Full Professor with the School of Computer Science and Technology, Beijing Jiaotong University, China. He was a Post-Doctoral Researcher with the University of Trento, Italy, from 2005 to 2006. He was a Post-Doctoral Researcher with TELECOM Bretagne and with INRIA, France, from 2007 to 2008. He was also an European ERCIM Fellow with Norwegian University of Science and Technology (NTNU), Norway, and with the Interdisciplinary Centre for Security, Reliability, and Trust (SnT), University of Luxembourg, from 2009 to 2011. He has authored or co-authored over 100 peer-reviewed articles in various journals and international conferences, including IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, ACM CCS, AAAI, Ubicomp, and IEEE INFOCOM. His recent research interests lie in data security and privacy-preserving computation. He has received the ACM CCS 2023 Distinguished Paper Award. He is an Elsevier "Highly Cited Chinese Researchers." He is the Vice Chair of ACM SIGSAC China. He is an Associate Editor of IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING and an Editorial Board Member of *Computers & Security* and of *Frontiers of Computer Science*.

Haining Wang (Fellow, IEEE) received the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, MI, USA, in 2003. Currently, he is a Professor with the Department of Electrical and Computer Engineering, Virginia Tech, USA. His current research interests include security, networking systems, and cloud computing.

Xiaobo Ma (Member, IEEE) received the Ph.D. degree in control science and engineering from Xi'an Jiaotong University, Xi'an, China, in 2014. He is currently a Professor with the MOE Key Laboratory for Intelligent Networks and Network Security, Faculty of Electronic and Information Engineering, Xi'an Jiaotong University. He was a Post-Doctoral Research Fellow with The Hong Kong Polytechnic University in 2015. He is also a Tang Scholar. His research interests include internet measurement and cyber security.

Shouling Ji (Member, IEEE) received the B.S. (Hons.) and M.S. degrees in computer science from Heilongjiang University, the Ph.D. degree in electrical and computer engineering from Georgia Institute of Technology, and the Ph.D. degree in computer science from Georgia State University. He is currently a Qiushi Distinguished Professor with the College of Computer Science and Technology, Zhejiang University, and an Adjunct Research Faculty Member of the School of Electrical and Computer Engineering, Georgia Institute of Technology. His current research interests include data-driven security and privacy, AI security and software, and system security. He is a member of ACM, a Senior Member of CCF, and was the Membership Chair of the IEEE Student Branch with Georgia State University (2012–2013). He was a Research Intern with the IBM T. J. Watson Research Center. He was a recipient of the 2012 Chinese Government Award for Outstanding Self-Financed Students Abroad and ten Best/Outstanding Paper Awards, including ACM CCS 2021.

Qiang Li received the Ph.D. degree in computer science from the University of Chinese Academy of Sciences in 2015. Currently, he is an Associate Professor with the School of Computer and Information Technology, Beijing Jiaotong University, China. His research interests revolve around the Internet of Things, networking systems, network measurement, machine learning for security, and mobile computing.