You Are What You Broadcast: Identification of Mobile and IoT Devices from (Public) WiFi

Lingjing Yu, *Institute of Information Engineering, Chinese Academy of Sciences; School of Cybersecurity, University of the Chinese Academy of Sciences;* Bo Luo, *The University of Kansas;* Jun Ma, *Tsinghua University;* Zhaoyu Zhou and Qingyun Liu, *Institute of Information Engineering, Chinese Academy of Sciences*

https://www.usenix.org/conference/usenixsecurity20/presentation/yu

This paper is included in the Proceedings of the 29th USENIX Security Symposium.

August 12–14, 2020

978-1-939133-17-5

You Are What You Broadcast: Identification of Mobile and IoT Devices from (Public) WiFi

Lingjing Yu[†‡], Bo Luo[§], Jun Ma[♯♮], Zhaoyu Zhou[†‡], Qingyun Liu[†‡]
[†] *National Engineering Lab for Information Security Technologies*
Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[‡] *School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China*
[§] *EECS/ITTC, The University of Kansas, Lawrence, KS, USA*
[♯] *Tsinghua University, Beijing, China;* [♮] *Pi2star Technology, Beijing, China*
yulingjing@iie.ac.cn, bluo@ku.edu, majun_ee@tsinghua.edu.cn, {zhouzhaoyu,liuqingyun}@iie.ac.cn

Abstract

With the rapid growth of mobile devices and WiFi hotspots, security risks arise. In practice, it is critical for administrators of corporate and public wireless networks to identify the type and/or model of devices connected to the network, in order to set access/firewall rules, to check for known vulnerabilities, or to configure IDS accordingly. Mobile devices are not obligated to report their detailed identities when they join a (public) wireless network, while adversaries could easily forge device attributes. In the literature, efforts have been made to utilize features from network traffic for device identification. In this paper, we present OWL, a novel device identification mechanism for both network administrators and normal users. We first extract network traffic features from passively received broadcast and multicast (BC/MC) packets. Embedding representations are learned to model features into six independent and complementary views. We then present a new multi-view wide and deep learning (MvWDL) framework that is optimized on both generalization performance and label-view interaction performance. Meanwhile, a malicious device detection mechanism is designed to assess the inconsistencies across views in the multi-view classifier to identify anomalies. Finally, we demonstrate OWL's performance through experiments, case studies, and qualitative analysis.

1 Introduction

Over the past decade, we have observed a steady growth in the number and types of portable devices. WiFi and cellular network remain the two major options for mobile devices to connect to the Internet. Although cellular networks have improved speed and coverage, and reduced costs in recent years, WiFi still has the edge in lower cost, better support from devices, and less capacity limits. Cisco predicts that the role and coverage of WiFi will continue to expand, and WiFi traffic will account for 50% of total IP traffic by 2022. Meanwhile, the number of public WiFi hotspots will grow 4-fold globally, from 124 million (2017) to 549 million (2022) in a five-year span [11]. With the significant growth of public Wifi support and usage, security and privacy concerns naturally arise.

The administrators of corporate and public WiFi services are concerned with malicious devices connecting to their networks, which may potentially harm the platform or other users in the network, e.g., [4, 45]. The security challenges are primarily caused by the diversity of devices, potential access to critical/core services, lack of proper security management by their owners, and limited auditing capability. On the other hand, users of public WiFi also express concerns about the security of their devices, data, and personal information. However, they do not always exercise proper privacy protection while connecting to unknown networks [5, 9, 30].

For system administrators, whenever a new mobile device connects to the network, it is critical to identify its manufacturer, type, and model, so that proper security precautions could be taken, e.g., configure firewall rules accordingly, verify if known vulnerabilities are patched, or inform IDS. In practice, identifying the *type* of mobile/IoT devices is of particular interest, since devices of the same or similar types are often managed under similar access control and firewall policies. For instance, when employees connect smart tea kettles or coffee makers to the network, the corporate security policy may place them in the same group that is limited from accessing any internal resource, while smartphones are expected to be governed by completely different policies. Meanwhile, the *manufacturer*[1] attribute also provides important information in device management. The same manufacturer tends to share the design and implementation of hardware and software components across products. As a result, they often have similar vulnerabilities and are patched simultaneously. For example, the firmware vulnerability reported in CVE-2006-6292 affects Apple's Mac mini, MacBook, and MacBook Pro

L. Yu, Z. Zhou, and Q. Liu were supported in part by the Youth Innovation Promotion Association of the Chinese Academy of Sciences, and the Key Technical Talents Project of CAS (Y8YY041101); B. Luo was supported in part by NSF-1565570, NSA Science of Security (SoS) Initiative, and the Ripple University Blockchain Research Initiative.

[1]In the rest of the paper, we use *manufacturer* and *make* interchangeably.

products. Meanwhile, regular users also have the need to discover potentially harmful devices, such as hidden cameras or a virtual machine with spoofed identity [10, 60], when they connect to WiFi hotspots. While active reconnaissance poses the risk of being detected and denied, users have the option of passive reconnaissance, where they receive and examine broadcast/multicast (BC/MC) messages to identify other devices in the same network, and looks for potential threats.

Efficient and accurate identification of mobile devices is challenging, especially when the features are limited and often incomplete. There is no standard protocol to actively query devices for their identities. Even if there were one, devices do not have to provide faithful answers. Existing researches on IoT device identification utilize a small set of network features and were only tested on approximately 20 to 50 devices in controlled environments, e.g., [43, 64]. With relatively small feature space, scalability becomes a concern. That is, detection accuracy may drop dramatically with the increasing quantity and diversity of devices in real-world applications.

In this paper, we attempt to answer three questions: (1) When a mobile/IoT device connects to a wireless network, what protocol(s) would broadcast information that may be received by other devices connected to the same WiFi? (2) What information or features contained in the broadcast messages are unique to a device, and how could system administrators or normal users make use of such information to accurately identify the important attributes: manufacturer, type, and model, of the devices? And (3) How can we utilize subtle hints caught during device identification to discover malicious devices?

To answer these questions, we present *OWL: overhearing on WiFi for device identification*. The key idea is to utilize the unique features in network packets that are introduced by the subtle differences in the implementations of network modules on mobile/IoT devices. OWL examines and utilizes all the features that could be passively collected from broadcast and multicast protocols such as `DHCP`, `DHCPv6`, `SSDP`, `mDNS`, `LLMNR`, `BROWSER`, `NBNS`, `IGMP`, etc. Distinct features extracted from related protocols naturally form a *view*. Multi-view learning is then employed to utilize views constructed from all available protocols for device classification. With fingerprints collected from more than 30,000 mobile/IoT devices, we demonstrate outstanding performance of the proposed mechanism.

Moreover, malicious devices may attempt to forge their identities and hide their presence to avoid being identified or tracked. For instance, in our dataset, we found a virtual machine running on a laptop that claimed to be an open WiFi hotspot. We argue that it is difficult for adversarial devices to completely forge the complex set of features from the entire stack of essential network protocols. We observed that fabricated or forged devices often behave inconsistently in different views, e.g., the fake WiFi hotspot demonstrated features of a real WiFi access point on some views, while showing features of its host laptop on other views. Therefore, we further attempt to discover malicious devices by examining the inconsistency across views in the multi-view classifier.

The technical contributions of this paper are: (1) We propose a multi-view wide and deep learning model to identify mobile/IoT devices using features from BC/MC packets collected through passive reconnaissance over WiFi; (2) Through large-scale experiments, we demonstrate the performance of the proposed mechanism in identifying the manufacturer, type, and model of mobile/IoT devices; and (3) OWL is also able to effectively detect forged or fabricated devices by identifying the abnormal inconsistencies across views.

The rest of the paper is organized as follows: we define the problem in Section 2, and explain the data collection processes in Section 3. We present the OWL algorithm, followed by implementation and experiments in Sections 4 and 5. We present case studies of abnormal devices in Section 6. We discuss other important issues and review the literature in Sections 8 and 9, and finally conclude the paper.

2 Problem Statement and the Threat Model

In this section, we formally present the objectives of OWL, followed by an adversary model of abnormal devices.

Device Identification. The primary goal of OWL is to identify devices on a WiFi network through packets they broadcast/multicast (BC/MC). Formally, device identification is a classification problem: given a set of labeled samples $\{(D_i, l_i)\}$, find a classifier $c : D \to L$, which assigns a label $l_x = c(D_x)$ to a new sample D_x. In OWL, D_i is a device represented by features extracted from BC/MC packets. Devices are identified at three granularity levels: {manufacturer}, e.g., “amazon”; {manufacturer-type}, e.g., “amazon-kindle”; {manufacturer-type-model}, e.g., “amazon-kindle-v2.0”. Last, we design OWL to only rely on unencrypted passive traffic that could be sniffed without any special privilege.

Abnormal Device Detection. It is beneficial to the administrators/users if OWL could tell if a device appears abnormal, besides labeling it. Therefore, another objective of OWL is to identify devices whose BC/MC traffic appears to deviate from known benign patterns. This abnormal sample could be a previously unknown device, or a fabricated/forged device. Formally, function $d : D \to \{\text{“benign”}, \text{“malicious”}\}$ is designed to assign a label $d(D_i)$ for each new device D_i. Initially, d is only trained with benign samples. When new malicious samples are confirmed, they are used to re-train d to improve the detection accuracy for future samples of this type.

Assumptions and Adversary Model. We assume that OWL could connect to the to-be-measured WiFi network–the network is open, or the WiFi security key is known. This is true for network administrators who measure their own networks. This is also true for users who attempt to detect suspicious devices when they connect to public WiFi. We also assume that the network infrastructure we connect to is benign, so that they faithfully forward/route packets as defined by the protocols, and OWL is able to collect those packets. Finally, we start with a clean model in the first task, where adversaries are not considered. Hence, we assume that the overwhelming majority of the devices in the training dataset are benign.

In the task of abnormal device detection, we employ a simple threat model as follows: the adversaries attempt to connect (unauthorized) devices to (public) wireless networks. The abnormal/malicious device could be: (1) devices that do not forge their own identities (so that they are unaltered, genuine devices), however, they are forbidden in the network, such as hidden cameras; (2) devices that attempt to hide their true identities. This includes fabricated or altered devices that connect to the network with malicious purposes, such as fake access points or DHCP servers, spoofed IoT device identities [60, 65]. This also includes devices that are counterfeit or forged at manufacturing, such as the fake Apple TVs we discovered (please see Section 6). This threat model only applies to the second task of the OWL approach.

3 Data Collection and Feature Extraction

3.1 Data Collection and Initial Analysis

Data was collected through a fully passive approach from three types of WiFi networks: (1) Open (unencrypted) public networks at coffee shops, restaurants, retail stores, some airports, etc. We directly connected to the hotspots without providing any credentials. (2) Open public WiFi with captive portals at airports, hotels, corporate guest networks, etc. We connected to these networks but did not provide information on landing pages. Hence, we were usually blocked from accessing the Internet, but we were able to sniff BC/MC packets. (3) Secure WiFi networks, including organization networks, home WiFi, and some public WiFi. We only collected data from networks that we were granted access to, such as university networks and retail stores that give passwords to customers. We connected the sniffing laptop to the networks, and employed `Wireshark` or `tcpdump` to download all BC/MC messages. The process was completely passive and non-intrusive. We did not turn on promiscuous or monitor mode. We did not actively send any message or make any spoofing attempt. The packets were all in plaintext and were also accessible to any other user on the same network.

With the help of our collaborators, we collected wireless network traffic from seven countries: US, Portugal, Sweden, Norway, Japan, Korea, and China. From January 2019 to July 2019, we collected data from 176 WiFi networks, among which 12 networks disabled BC/MC. Each data collection session lasted approximately 20 to 30 minutes. The WiFi networks we sniffed were very diverse in terms of ownership, including university, airport and hotel WiFi, restaurant, retail store, and volunteers' household WiFi. In total, we collected BC/MC packets from 31,850 distinct devices, which were identified by MAC addresses. Figure 1 (a) shows the distribution of WiFi networks (allowed BC/MC) and devices. The number of devices per network is higher in Korea and China, mostly due to higher population density. In particular, we collected data from an airport in Korea and a student dorm in China, which contributed large volumes of devices. We statistically analyzed the collected data and found the following:
1. In total, we have identified 275 distinct protocols in the data. Note that we treat `UDP` packets to different ports as distinct protocols. Figure 1 (b) shows the distribution of the top 10 most frequently used protocols, led by `ARP`, `ICMPv6` and `mDNS`.
2. 69.5% of devices sent BC/MC packets using more than 2 protocols and 46.1% of devices sent BC/MC packets using more than 3 protocols. Intuitively, the more protocols devices use for broadcasting, the more information they leak. 51.9% of the devices sent `mDNS` packets, which may convey semi-identifiable attributes of the devices. Application layer protocols like `DHCP`, `SSDP` and `LLMNR` are also wildly used.
3. Protocol popularity appears to be consistent across countries, with a few exceptions. For instance, `mDNS` is the most frequently used BC/MC protocol in the US, Japan, and Sweden, but is ranked lower in the other countries. This is explained by the fact that these countries have higher density of Apple devices[2], which intensively use `mDNS` to discover services in the network. Meanwhile, `Dropbox LAN Sync Discovery (DLSD)` is not found in China, because DLSD is a proprietary protocol of Dropbox, which is blocked in China.
4. Some protocols are only used by one type of devices. For instance, the `KINK` protocol is only found in packets sent from Samsung TVs. This observation implies two perspectives: (1) the proprietary protocols are good identifiers of hardware/software manufacturers; (2) when a proprietary protocol appears in the traffic generated by a third-party device (identified from other network traffic features), such device should be further investigated–it could be a spoofed device.
5. In the initial analysis, we employ Apriori [56] to statistically examine the patterns of BC/MC protocols used in each type of devices, and show some examples in Table 1. For each device, the protocols are ranked by the frequency of captured packets. We can observe that each device family may have its distinct frequency pattern of protocols. Different products from the same manufacturer may show the same/similar pattern of protocols, e.g., several `DLink` devices demonstrate identical patterns of protocols. Most likely, such devices share the same hardware and software in their WiFi component.

The initial analysis suggests the possibility of using features extracted from BC/MC packets to identify the make, type, and model of the devices. The complexity of the patterns also implies that it could be very challenging for adversaries to perfectly spoof the network features of other devices.

3.2 Ethical Considerations

We collected data through a completely passive approach. We did not turn on promiscuous mode. That means, we were

[2]According to OS market share by country reported by `https://gs.statcounter.com/os-market-share/`

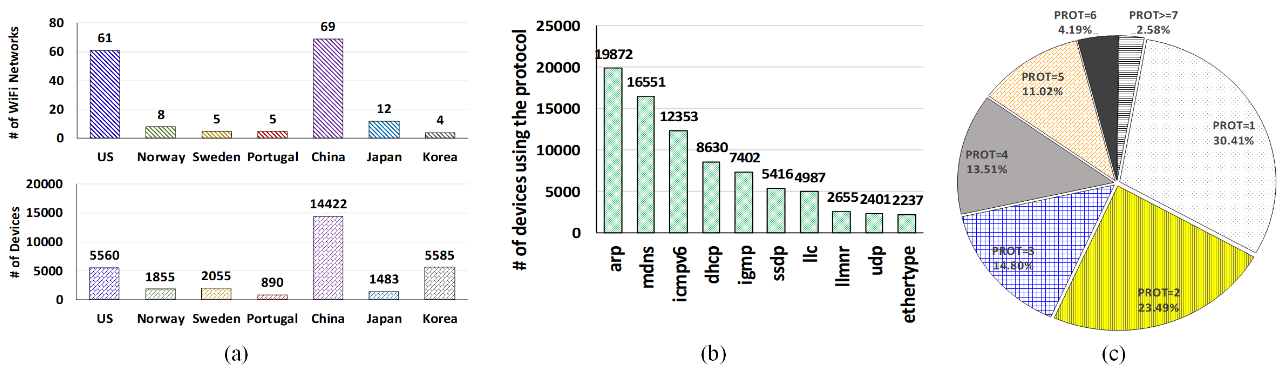


Figure 1: Statistics of collected data: (a) distribution of sniffed WiFi networks and devices in 7 countries; (b) the top 10 most frequently used BC/MC protocols in the dataset; and (c) the distribution of number of protocols used in devices.

Table 1: Examples of broadcast/multicast protocol frequency patterns of mobile/IoT devices.

device-type	protocol frequency pattern	device-type	protocol frequency pattern
apple-phone	`ARP,mDNS,DHCP,ICMPv6,LLC,IGMP`	apple-smartspeaker	`ARP,ICMPv6,mDNS`
dlink-siren	`ARP,mDNS,DHCP,ICMPv6,IGMP`	hikvision-camera	`ADWIN_CONFIG,SSDP,IGMP`
dlink-watersensor	`ARP,mDNS,DHCP,ICMPv6,IGMP`	lg-tv	`ARP,mDNS,ICMPv6,SSDP,IGMP`
edimax-camera	`ARP,mDNS,DHCP,SSDP,IGMP`	sumung-tv	`ARP,UDP_15600,UDP_8001,IGMP`
microsoft-gameconsole	`mDNS,LLMNR,ICMPv6,DHCPv6,IPv6,SSDP,IGMP`	xiaomi-humidifier	`ARP,mDNS`

the legitimate and intended receivers of the BC/MC packets. These packets were also received by all other computers in the same subnet. We did not eavesdrop on any unicast packet. We did not attempt to send anything (e.g., ARP requests). To our best knowledge, the data collection process did not violate any networks' Terms and Conditions that were presented to the users. None of the T&Cs mentioned BC/MC traffic or network monitoring. Some forbid activities that may impact the security or usability of the network, while we did not impact the network. Some information in our data set may be considered somewhat sensitive. We discuss them here:

1. MAC. MAC addresses are unique identifiers of devices (not users). Recent research showed that users are vulnerable to MAC tracking attacks [14]. Such privacy risk does not apply in our data: (1) we only briefly collected data from sites that are very sparsely scattered globally. The probability to re-encounter the same MAC is extremely low. (2) We only retained the top six hexadecimal digits of MAC addresses. They cannot be used as unique identifiers of devices.

2. Device Name. Some devices (e.g., iOS devices) allow users to configure device names, and adopt them in several protocols, such as mDNS and DHCP [3]. Users may name the device with their own name (e.g., Alice's iPhone). We observed individuals' names in approximately 7% of the devices. The majority of them were first names, and many were fake names. In data pre-processing, we removed all names and analogues.

Besides MACs and (some) names, we do not have any identifier or personal information in the data. We did not collect any opinion, behavioral information, sensor data, demographic attribute, or other sensitive information. It is extremely difficult, if not impossible, to associate the collected data with offline identities. We did not make any attempt to discover personal information or to track any user. The data collection and analysis process did not introduce any risk to any user. The information we collected was technical data that was received by a large audience (anyone in the subnet).

We discussed the project and data collection process with the IRB of the *National Engineering Lab for Info. Sec. Tech.* at CAS. They determined that our project was not human subject research, and it did not need a full IRB review. The Human Research Protection Program at the *University of Kansas* reviewed our written memo and agreed with the decision.

3.3 Identifiers and Feature Extraction

We extracted three categories of features from the sniffed BC/MC packets: (1) the identifiers are (almost) unique to each make/type/model of the devices, i.e., they can be employed to uniquely identify devices when they are available. (2) The main features are robust discriminators that can be combined to collectively provide enough information to distinguish devices. (3) auxiliary features are collected through actively querying devices. We only use them in evaluation.

1. Identifiers. Examples of protocols/fields that may carry device identification attributions are listed in Table 2, roughly ordered by their popularity and robustness (i.e., the unlikelihood to be altered). MAC prefix is available on every device and it could be utilized to infer the manufacturer of a device [40]. We retain the top six hexadecimal digits of MAC addresses in the MAC prefix feature, e.g., string "80:e6:50" is extracted from MAC "80:e6:50:19:54:4e". However, MAC prefix may only indicate the manufacturer of the WiFi module on some devices, not the device manufacturer. Next, `Host Name` in `DHCP`, `answer names` in `mDNS` response messages are

[3]Although Android allows users to set device names, the user-defined names are only used as hotspot and Bluetooth names, while DeviceName in mDNS and DHCP are manufacturer-defined strings that cannot be changed.

Table 2: Examples of data fields that may contain identifiers.

priority	Protocol	Fields
1	–	MAC prefix
2	DHCP	Option12 (HostName)
3	DHCP	Option60 (VendorClass)
4	DHCP	Option77 (ModuleName)
5	DHCPv6	Option39 (ClientFQDN)
6	MDNS	answer names in response messages
7	SSDP.MSEARCH	user-agent
8	SSDP.MSEARCH	X-AV-Client-Info
9	LLMNR	query name
10	BROWSER	query name
11	NBNS	query name
12	UDP	device name

also meant to contain device names. The other fields listed in Table 2 may contain keywords, such as "hp_printer_mfp-m227fdw", which can be used to directly identify devices.

Although identifiers are unique and informative, we can not solely rely on them in identifying mobile/IoT devices in practice. First, they are simply unavailable in the majority of the devices. In our data set, approximately 30% of the devices contain additional identification information beyond MAC addresses. Moreover, adversaries or even benign users may tamper with the identifiers listed in Table 2. For instance, a user may change the name of her iPhone in phone settings, e.g., to "Alice's New Toy". This name will now appear in the `HostName` field of the `DHCP` requests from this phone.

2. Main Features from BC/MC Packets. We categorize our main network features into two types: (A) key-value pair features and (B) pseudo natural language features.

2A. Key-value Pairs. A key-value pair feature has a distinctive name and a corresponding value, which is categorical, numerical, or textual. We treat features from `DHCP`, `DHCPv6`, `SSDP`, `LLMNR`, `BROWSER` and `NBNS` protocols as key-value pairs.

Each `DHCP option code` is regarded as the key, with the option value as the value. For instance, in a `DHCP Request` packet, the value of `Option 57` (Maximum Message Size) is 1500. We extract the feature as: "57:1500". For `DHCP Option 55` (Parameter Request List), the value is the sequence of all request option codes in a string. Besides, we also generate a special feature with "dhcp-key-seq" as the key, the sequence of all option codes in the `DHCP` message as the value.

Messages from protocols such as `SSDP`, `LLMNR`, `BROWSER` and `NBNS` are composed of key-value pairs intrinsically, so that they are directly extracted. For the `SSDP` protocol, we also generate a key "ssdp-key-seq" with the sequence of keys in `SSDP` as its value, which is similar to "dhcp-key-seq" feature.

IP addresses do not provide device identity information. However, the use of IPv4 and IPv6 and the port numbers are informative. IPs are transferred to strings ("IPv4" or "IPv6"), while port numbers are preserved. For example, attribute value "239.255.255.250:1900" is converted to string "IPv4:1900". Last, MAC prefixes are retained as key-value pairs.

2B. Pseudo Natural Language Features. Besides the key-value pair features, several protocols also include unstructured textual content in their messages: (1) `mDNS` payloads, (2) broadcast/multicast protocol sequence, and (3) payloads of `UDP` packets that are not resolved to a specific protocol.

We treat the resource records (RRs), authoritative-nameservers, and additional records in `mDNS` messages as pseudo-natural-language features. We concatenate the values from all the fields in an RR (`RR name`, `RR type`, `cache-flush`, etc) into a string, in which fields are separated by '|', so that each field will be processed as a token in feature embedding. Especially, the string values of `RR name` field and `domain name` in `RDATA` field are divided into multiple tokens by replacing all the occurrences of the '.' character with the separator '|'. Each `TXT` value in `data` field is treated as a token as well. On average, there are 216 tokens in each RR. We group all the RR strings into the `mDNS` feature of a device.

Next, we concatenate the protocol sequence used by each device to an unstructured text string, and use it as a pseudo-natural-language feature. We also observed that some `UDP` messages, which are not resolved to a specific protocol, contain information of devices in the form of unstructured plain text. Hence, we also extract all textual data from the payloads of such BC/MC UDP messages as pseudo natural language features. Last, we would like to note that all IP and MAC addresses in pseudo natural language features are also transformed in the same way as key-value pair features.

3. Auxiliary Features. Devices may advertise their services by multicasting a `SSDP notify` message with the device description URL provided in the "LOCATION" field. The URL points to a device description file in XML format that contains identifiable information of the device. For instance:

```
<friendlyName>DELL-PC: dell:</friendlyName>
```

Following the URL to download the file is considered active reconnaissance and somewhat intrusive. Therefore, we only attempted to extract this feature within the authors' organizational networks. The information discovered in device description files is only used in performance evaluation.

4 Device Fingerprinting and Classification

In this section, we present the core algorithms of OWL, i.e., multi-view wide and deep learning for device manufacturer/type/model classification and abnormal device detection.

4.1 MvWDL Algorithm Overview

In theory, device identification is a classification task, which predicts manufacturer/type/model labels for new devices based on models learned from training data. In Section 3.3, we have extracted features from packets of BC/MC protocols. As observed in our initial analysis and discussed in the literature (e.g., [42, 43, 53, 64]), features from each protocol provide certain degree of discriminatory power in device identification. However, when the sample size increases from tens to tens of thousands, none of the protocols provides enough information to differentiate the whole spectrum of devices. For example, we have observed that two devices `magic-cast` and Apple TV share very similar mDNS messages, but they differ significantly in their DHCP and SSDP packets. Meanwhile, Apple's iPhone and MacBook generate almost identical DHCP packets, while their mDNS packets differ from each other.

Our initial observations demonstrate that: (1) each BC/MC protocol generates an *independent* set of features that could contribute to device classification; (2) sets of features from different protocols complement each other in terms of discriminatory power in the identification of large volume of devices and labels; (3) not all protocols are available in all devices, due to devices' capabilities and network configurations. Intuitively, our observations suggest the use of multi-view learning, where features from different protocols are naturally organized into views, and classification functions on all views are jointly optimized. In practice, multi-view learning recognizes the inherent diversity and relationships of the features, which confirm each other in some regions of the feature space, while complementing each other in other regions. Multi-view representations are integrated so that different feature spaces (views) are transformed into the same latent space, to improve overall classification and generalization performance.

Multi-view learning enforces view consistency in training. When a testing sample triggers strong inconsistency across views, it is either a new device label that is not in the training data, or a malicious device, whose networking components have been tampered with. Especially, when multiple views indicate strong confidence in contradictory predictions, the device is highly likely to be spoofed or fabricated.

In this paper, we present a multi-view wide and deep learning (MvWDL, Figure 2) framework for device classification and abnormal device detection. MvWDL consists of three components: (1) MvWDL first extracts features from BC/MC packets and learns multi-view embedding representations as device fingerprints. (2) Inspired by the wide & deep learning model [8], a hybrid-fusion multi-view artificial neural network is designed to fuse dense embeddings from six independent and complementary views in two structures: (a) a deep neural network for early fusion is designed to maximize the generalization performance, and (b) a wide component for late fusion is added to improve the memorization of label-view interaction, i.e., how does each view response to each manufacturer/type/model of devices. (3) Malicious devices are detected with view inconsistency. A "positive" loss function enhances view consistency for benign samples. Meanwhile, when malicious devices are confirmed and labeled, they are incorporated through a "negative" loss function, which captures the view inconsistencies caused by malicious devices.

4.2 Device Fingerprinting

We first construct device fingerprints from features extracted from BC/MC packets. In practice, key-value pair features and pseudo natural language features are processed differently.

Fingerprints from key-value pair features. Formally, the set of all key-value pair features for a device are defined as: $KV_i = \{k_{i,1} : v_{i,1}, \dots, k_{i,n} : v_{i,n}\}$, in which $k_{i,j}$ and $v_{i,j}$ denote the jth key and jth value of device i, respectively. The *global key list* is defined as an ordered collection of all the keys in the entire dataset: $K = \{k_1, k_2, \dots, k_N\}$. Corresponding to the order of keys in the key list, we define the key-value pair feature vector of the device i as: $\bar{V}_{\mathrm{KV},i} = \{v_{i,1}, v_{i,2}, \dots, v_{i,N}\}$. All categorical values are tokenized. When a key k_j does not exist in KV_i, we set the corresponding value $v_{i,j}$ to "null".

Fingerprints from pseudo natural language features. To fully utilize information resides in the content of pseudo-natural-language features, we explore two content modeling algorithms to generate fingerprints: (1) word to vector (word2vec), and (2) Latent Dirichlet Allocation (LDA).

(1) *Word2vec.* The word2vec approach [44] is based on the distributional hypothesis, which indicates that terms occurring in the same context tend to have similar meanings. Its main purpose is to vectorize words in the text corpus so that words appearing in similar contexts are represented by vectors close to each other in the feature space. In its existing implementations, the contexts are captured with sliding windows. However, they are not suitable in our application, in which the context of a token in the pseudo-natural-language features must remain in the scope of a "sentence" (a RR in mDNS, a field in the sniffed packet). Hence, we implemented our own word2vec scheme in three steps: (1) We build a *word to id* dictionary to tokenize terms in the dataset. (2) The entire corpus is used to train a neural network model M_{w2v} to maximize the conditional probability of a word given its context, i.e.:

$$\underset{\theta}{argmax} \prod_{(\omega,c)\in D} p(\omega|c;\theta) \tag{1}$$

in which θ is the optimization goal while maximizing the conditional probability of word (ω) given the context (c). D is the set of all (ω, c) pairs. The context (c) of a word (ω) is composed of a window of 5 terms centered at ω, and restricted in the same string as ω. (3) We apply M_{w2v} to each word to get its corresponding vector. The feature vector of the entire string is constructed as the mean of all its token vectors.

(2) *Latent Dirichlet Allocation.* LDA is a classic topic modeling approach based on the Bag-of-Words model [6]. Its idea is to construct a model of document-topic-term relationship using unsupervised learning. Different from word embedding, LDA generates human-interpretable topic models. With the observation that different devices usually show diverse topic distributions, we utilize LDA to statistically model the topic distribution of the pseudo natural language features.

4.3 Multi-view Wide & Deep Learning

Features extracted in Section 4.2 are organized into views based on their host protocols, as listed in Table 3. Some simple protocols, such as ARP, generate identical packets from different devices. They only contribute to the *protseq* feature, which records the sequence of protocols used by a device.

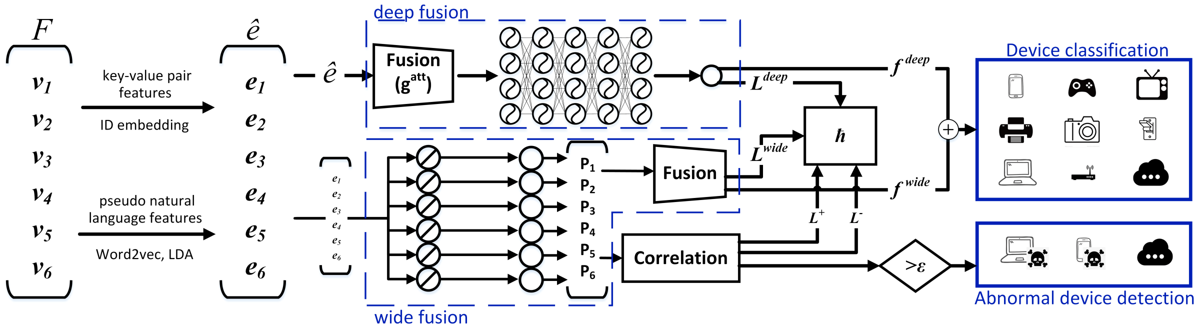


Figure 2: An overview of the multi-view wide and deep learning framework.

Table 3: View dimensionality before/after embedding.

view	base protocols	dimensionality
DHCP	`DHCP` and `DHCPv6`	85/680
mDNS	`mDNS`	7/128
SSDP	`SSDP`	67/536
LBN	`LLMNR`, `BROWSER`, `NBNS`	16/128
UDP	other `UDP` features	1/128
protseq	protocol sequence & MAC prefix	2/136

In multi-view embedding representation, six views are formally denoted as: $F = \{v_1, v_2, v_3, v_4, v_5, v_6\}$. We learn a global word embedding space and LDA topic space for each view. The embedding representation of view v_i is defined as:

$$e_i = f^i(v_i; w_i) \tag{2}$$

where w_i is the view-specific column-index matrix for v_i. f^i is a range of column-specific embedding operations (id embedding, `word2vec`, and LDA), followed by a `concat` operation to generate view v_i's final dense embedding e_i.

4.3.1 Deep Fusion

The deep fusion component implements the early-fusion model of multi-model learning. We fuse the dense embedding $\hat{e} = [e_1, e_2, \dots, e_6]$ into one compact vector $\mathbf{e}$ as the initial input of the fusion neural network. Based on offline experiments, we choose affine transformation as the attention operation $g^{att}(\cdot)$, instead of other popular operations such as sum fusion, max fusion or concatenation fusion.

$$g^{att}(\hat{e}; W_a, b_a) = \text{softmax}(\tanh(W_a^T \hat{e} + b_a) \tag{3}$$

$$\mathbf{e} = g^{att}(e_1, e_2, \dots, e_6) \tag{4}$$

where W_a and b_a are the affine transformation parameters.

Next, we feed the fusion vector $\mathbf{e}$ into a deep neural network $f^{deep}(\cdot)$. Its main component is a standard multi-layer perceptron (MLP), where the output of layer k is defined as:

$$\ell^{(k+1)} = \sigma(W^{(k)} \ell^{(k)} + b^{(k)}) \tag{5}$$

σ is the *ReLU* (Rectified Linear Unit) activation function (except that the last layer is a fully connected layer). $W^{(k)}$ and $b^{(k)}$ denote the perceptron weight and bias, respectively.

The objective loss $\mathfrak{L}^{deep}$ of the deep component is defined as a maximum likelihood estimation function $\bar{P}$:

$$\mathfrak{L}^{deep} = \bar{P}(y = t|\mathbf{e}; \theta) = \prod_{c=1}^{C} \mathbb{P}(y = t_c|\mathbf{e}; \theta)^{\mathbb{I}(y=t_c)} \tag{6}$$

where C is the training set, and t_c is the label of sample c. $\mathbb{P}$ is the conditional probability of a sample being labeled as t_c under input $\mathbf{e}$, with θ as the parameter set. $\mathbb{I}$ is the indicator function. The optimization progress is denoted by maximum log likelihood and stochastic gradient descent ∇:

$$\nabla_\theta \mathfrak{L}^{deep} = \underset{\theta}{\text{argmax}} \mathbb{E}_{\mathbf{e} \sim \hat{p}_{view}} \log(f^{deep} \circ g^{att}(\mathbf{e})) \tag{7}$$

where $\mathbb{E}$ is the expectation and $\hat{p}_{view}$ is the distribution of $\mathbf{e}$. As a feedforward neural network, the classification probability $f_{t_c}^{deep}$ of the deep fusion network is defined as:

$$f_{t_c}^{deep} = \frac{\exp(\ell_{t_c}^{(K)})}{\sum_{t_c} \exp(\ell_{t_c}^{(K)})} \tag{8}$$

4.3.2 Wide Fusion

Besides the deep component for generalization performance, we add another *wide component*, which implements the late-fusion model of multi-model learning, to memorize the interactions among features, views and labels. The wide component takes $\hat{e}$ as input, applies affine transformation on each view e_i and trains a wide linear model to produce:

$$\mathfrak{p}_i(y = t|e_i) \tag{9}$$

where $\mathfrak{p}_i$ is the classification result from view i. Similar to the deep component, we also use maximum log likelihood and stochastic gradient descent to define and optimize the loss of wide component $\mathfrak{L}^{wide}$:

$$\nabla_\theta \mathfrak{L}^{wide} = \underset{\theta}{\text{argmax}} \sum_{e_i \in \hat{e}} \mathbb{E}_{e_i \sim \hat{p}_{view}} log(f^{wide}(e_i)) \tag{10}$$

where $f^{wide}(\cdot)$ represents a one-layer network for multi-class classification, whose output is also narrowed by softmax. The c-th element of the wide fusion output, $f_{t_c}^{wide}$, indicates the probability of the sample (device) being labeled as t_c. It is defined as the sum of view-wise probabilities:

$$f_{t_c}^{wide} = \sum_{e_i \in \hat{e}} \mathfrak{p}_i(y = t_c | e_i) = \sum_{e_i \in \hat{e}} \frac{\exp(\omega_i^T e_i + \gamma_i)}{\sum_{t_c} \exp(\omega_i^T e_i + \gamma_i)} \tag{11}$$

As defined in [8], there are two essential requirements for the wide component: (1) only linear operations are allowed in the wide model. Therefore, ω_i and γ_i in Eq. (11) are affine transform parameters. (2) the output of the wide component is linearly merged to the deep component. Hence, we fuse the output of the wide and deep components and define the final conditional probability of the classifier output as:

$$f_{t_c}^{final} = f_{t_c}^{wide} + f_{t_c}^{deep} \tag{12}$$

4.3.3 View Consistency and Malicious Device Detection

Besides the wide and deep learning approach for mobile device identification, our second objective is to identify malicious devices through BC/MC network features. To achieve this, the following two assumptions are necessary:

Assumption 1. For a benign testing sample, label probabilities $\mathfrak{p}_i(y = t|e_i)$ generated from different views in the wide component shall demonstrate strong consistency.

Assumption 2. When label probabilities from different views demonstrate certain level of inconsistency/disagreement, the device is either new to the model, or fabricated/forged.

In multi-model learning, view consistency (Assumption 1) is a fundamental objective that is often referred to as the consensus principle. It is achieved by different mechanisms such as co-training or shared latent sub-space. In the wide and deep components of OWL, views are jointly optimized so that view agreements are implicitly included in the objectives. To further enhance the mutual agreements across views, we define the correlation-based loss to explicitly maximize view consistency in training. First, view correlation is defined as:

$$corr(\mathfrak{p}_u, \mathfrak{p}_v) = \|\mathfrak{p}_u - \mathfrak{p}_v\|_2^2 \tag{13}$$

where $\mathfrak{p}_u$ and $\mathfrak{p}_v$ are output vectors from two different views of the wide component. Their correlation is defined with L-2 norm. The loss function is defined as a *sigmoid* function:

$$L^+ = \sum_{(u,v,k) \in \mathcal{D}_{pri}} log \frac{1}{1 + e^{-corr(\mathfrak{p}_u^k, \mathfrak{p}_v^k)}}, \tag{14}$$

where L^+ is the loss of augmented correlation, $(\mathfrak{p}_u^k, \mathfrak{p}_v^k)$ denote the output from views u and v for sample k, and $\mathcal{D}_{pri}$ denotes the priori dataset which includes all labeled benign samples. Note that we do not have any known malicious sample in the initial dataset. We also assume that benign samples always significantly outnumber malicious samples in the dataset.

Assumption 2 denotes that malicious devices that attempt to fabricate identities often cause inconsistencies in the BC/MC packets. In OWL, we quantitatively model the degree of view inconsistency, and use it for malicious device detection:

$$\sum_{1 \le u,v \le 6} (\eta corr(\mathfrak{p}_u^k, \mathfrak{p}_v^k) + (1 - \eta)\mathbb{I}(A(\mathfrak{p}_u^k) \neq A(\mathfrak{p}_v^k))) > \varepsilon \tag{15}$$

where A returns the index with the largest probability in $\mathfrak{p}^k$. η is a trade-off parameter in $[0,1]$, which balances the probability inconsistency $corr(\mathfrak{p}_u^k, \mathfrak{p}_v^k)$ with the type inconsistency. ε is the threshold that separates benign and suspicious samples.

Last, when malicious devices are detected and confirmed through manual investigation, they are formally labeled and used to train a fourth loss function, which attempts to maximize view inconsistency for known malicious devices:

$$L^- = \sum_{(u,v,k) \in \mathcal{D}_{pos}} log \frac{1}{1 + e^{corr(\mathfrak{p}_u^k, \mathfrak{p}_v^k)}} \tag{16}$$

where $\mathcal{D}_{pos}$ denotes the posterior dataset of labeled malicious samples. They are also removed from the benign set $\mathcal{D}_{pre}$.

Finally, four loss functions are combined to learn all parameters jointly (Eq. 17). In summary, the deep component ($\mathfrak{L}^{deep}$) is a maximum likelihood estimation function optimized towards the best classification performance for device labels under input features; the wide component ($\mathfrak{L}^{wide}$) is to optimize classification performance on each view; the L^+ component is optimized towards the maximum view agreement for benign samples; and the L^- component is to maximize the view inconsistency for malicious devices. All four objectives are integrated in the MvWDL model (Figure 2).

$$\hbar = \mathfrak{L}^{deep} + \mathfrak{L}^{wide} + L^+ + L^- \tag{17}$$

5 Implementation and Experiments

In this section, we briefly introduce the implementation of OWL, and then present our experiment results.

5.1 Dataset and Data Labeling

At the finest granularity of the device identification task, each device is expected to receive three labels:

```
{Manufacturer, Type, Model}
```

We refer to the literature [23, 24, 33, 43] to define 34 types of devices. Examples of popular types of devices are:

```
phone computer pad router camera smart-plug
smart-switch virtual-machine game-console tv
lightbulb printer kettle watersensor watch
```

Based on the availability and trustworthiness of labels, our samples are categorized into four sets: (1) samples with validated labels (i.e., the ground truth data); (2) samples labeled in the semi-automatic process; (3) samples with auxiliary (SSDP) features; and (4) samples without any label.

Ground Truth Data. A portion of our data was collected in controlled environment, such as our own lab network or home network. We obtained the true labels of such devices. We were also able to verify the manufacturer/type/model of some display items in electronic stores. In total, we have 423 devices with validated labels in our *ground truth dataset*. Note that each device in this category receives all three labels.

Semi-automatic Device Labeling. The majority of the samples were collected from uncontrolled environments. To create labels, we design a semi-automatic labeling process: (1) for an unlabeled device, we manually examine human-interpretable text in the sniffed packets (fields listed in Table 2). If the information appears to be benign and consistent, we label the device accordingly. Note that we may not learn all three labels for a device. (2) When patterns are observed from a specific manufacturer/type/model of devices, we create labeling rules in the form of `{Condition => Label}`. For instance, `{MAC:D-link; HostName:DCS-930LB => (D-link, camera, dlink_camera_dcs-930lb)}` states that when MAC prefix indicates D-link and DHCP Option-12 (HostName) contains string "DCS-930LB", the device labeled as D-Link camera DCS-930LB. (3) The rules are used to process all unlabeled samples. All automatically generated labels are verified by the creators of the rules. Rules may be refined and re-applied during this process. We then move to (1) for the next unlabeled device. (4) All labels, manually or automatically created, are further reviewed and confirmed by another member in the team.

Eventually, we annotated 4,064 devices to the finest granularity: {manufacturer, type, model}, among which 410 distinct device models were identified (the ground truth data is not included here). In addition, 6,519 devices were annotated with {manufacturer, type}, while the exact models were unknown. 15,895 devices were labeled with {manufacturer} only. They are called the *annotated dataset*. 4,871 devices were left without any label, i.e., the *labelless dataset*. Last, 78 devices were set aside as supplementary testing data (to be discussed).

Among the three labels, manufacturer is the easiest to identify and reveals the least amount of information. It does not tell administrators how the access policies could be configured, or tell other users if the device could be suspicious. Meanwhile, sometimes the MAC prefix only tells the manufacturer of the network components, instead of the manufacturer of the device itself. For instance, we have seen several different devices carrying TP-Link's mac prefix. Therefore, some devices are left unlabeled although the MAC prefixes are known.

The Sanitized Dataset. We used human-interpretable textual content in network packets for device annotation. The text content is also processed as pseudo natural language features and used for device identification in OWL. Meanwhile, we also like to answer this question: "*How much does OWL rely on human interpretable textual features to identify devices?*" For this purpose, we sanitized all the annotated samples by removing all identifiers (labels were preserved for evaluation purposes). That is, we removed all the keywords that are used in `{Condition => Label}` rules, including all the MAC prefixes, to create this *Sanitized Dataset*. We verified that neither the labeling rules nor the human annotators were able to distinguish any device (at any granularity) in the sanitized data.

Samples with Auxiliary Features and Supplementary Testing Data. As introduced in Section 3.3, we followed the device description URLs in `SSDP notify` to obtain auxiliary features for devices in our organizational network. We collected meaningful device descriptions for 180 devices. They are utilized in two ways: (1) for 102 samples that are annotated to {type} and {model} levels in the labeling process, we employ auxiliary features to validate the labels. They are included in the annotated dataset. (2) For samples that are labelless or only labeled with {manufacturer}, we deliberately set them aside and only used them in testing. We call this set of 78 devices the *Supplementary Testing Dataset*.

Trivial Features and Unidentifiable Devices. Feature sets that carry identical values across many device types are called trivial features. Devices with only trivial features are unidentifiable in theory. We identify such devices in four steps: (1) `apriori` is used to find feature frequencies for each device type. (2) Devices with informative protocols (`mDNS`, `SSDP`, `DHCP`, `DHCPv6`, `LLMNR`, `NBNS` and `BROWSER`) are eliminated. (3) In the remaining devices, feature sets that appeared in more than N device types are called trivial features. (4) Devices that contain only trivial features are marked as unidentifiable.

5.2 Experiment Results

To test the performance of the MvWDL model presented in Sec 4, we evaluate its performance from three aspects: (1) the accuracy and coverage of classification in comparing with other methods, (2) performance on sanitized data (extreme condition), and (3) the speed of device identification.

Metrics. The performance of device identification is evaluated by three metrics: (1) the *coverage* (C) denotes the fraction of all devices that OWL (or another approach) could generate a label for; (2) the *accuracy* (A) is the fraction of labeled devices that are correctly labeled; and (3) the *overall identification rate* (OIR) denotes the faction of all devices that are correctly labeled. They are formally defined as:

$$C = |\{\text{labeled devices}\}|/|\{\text{all devices}\}| \quad (18)$$

$$A = \frac{|\{\text{correctly labeled devices}\}|}{|\{\text{labeled devices}\}|} \quad (19)$$

$$OIR = \frac{|\{\text{correctly labeled devices}\}|}{|\{\text{all devices}\}|} = C \times A \quad (20)$$

We compare the performance of OWL with state-of-art device identification mechanisms, which could be roughly categorized into fingerprint-based and rule-based approaches. Fingerprint-based approaches extract features from network traffic and then employ supervised learning for device identification. Among this category of approaches, WDMTI [64] produces good performance on MC packets (DHCP). Rule-based approaches extract text keywords from payload of unencrypted network traffic to create `{keywords->device}` rules for device identification. ARE [24] is the state-of-art approach in this category of solutions. We implemented WDMTI, extracted its features from our dataset, and trained/tested it in the same way as OWL. We also implemented ARE to extract rules from our training data and applied them on testing data. Note that we employed ARE on different protocols from [24].

Performance Evaluation on Ground Truth Data. We compare the performance of WDMTI, ARE and OWL on the ground truth data. We compare their accuracy, coverage and OIR at three different granularity, from coarse to fine: {manufacturer}, {manufacturer, type}, and {manufacturer, type, model}. We perform a 10-fold cross validation and demonstrate the results in Figure 3 (a). The results show that: (1) OWL provides the best overall performance (OIR) at all granularity levels. Its coverage is consistently the highest, as OWL could always extract features from the network traffic and predict a label. At finer granularity, OWL significantly outperforms both ARE and WDMTI in OIR. (2) ARE has the best accuracy but limited coverage, especially at fine granularity levels. It is able to correctly identify the manufacturer of more than 80% of the devices, since MAC prefixes are used for this label and they are mostly available. For type and model, the informative textual terms are not always available in network traffic, hence, ARE is unable to identify the majority of devices. (3) WDMTI solely depends on features extracted from DHCP packets, hence, its coverage is always limited.

Performance Comparison on Annotated Data. We evaluate all three approaches on the annotated dataset, as shown in Figure 3 (b). First, since we were able to annotate the manufacturer of all the samples in this dataset, they all contain enough features for OWL and ARE to identify the manufacturer, i.e., they both achieve $C = 100\%$ on {manufacturer}. Not all devices contain enough information for ARE to identify their type and model. For the same reason, we were unable to annotate type and model these samples in the dataset. However, OWL could still utilize non-human-interpretable features to classify these devices. Hence, OWL's coverage is significantly higher than ARE and WDMTI for the two fine-grained labels.

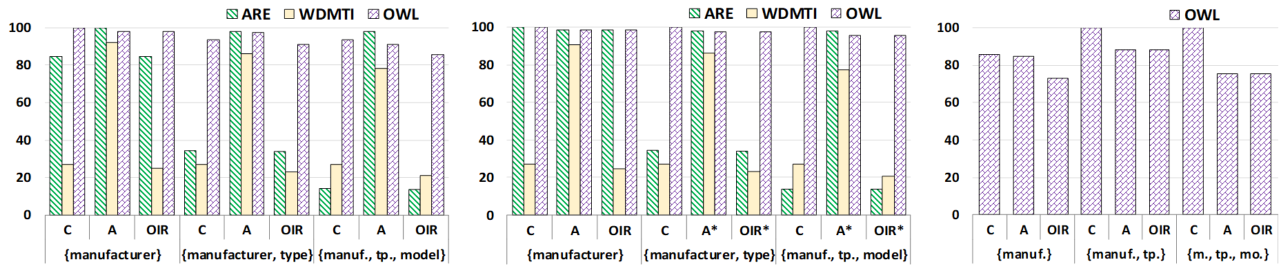
Next, we evaluate the accuracy of all three approaches based on the annotations. We do not have {type} and {model} annotations on more than 50% of the samples. Although OWL is able to estimate these labels for unannotated samples, we cannot tell whether such estimations are correct. That is, for the {manufacturer, type} and {manufacturer, type, model} granularity, coverage (C) is evaluated on *all* samples in the annotated dataset, while accuracy (A) is only evaluated on *partial* data–samples with {type}, and {model} annotations. Therefore, we mark accuracy and OIR with A* and OIR* in the figure, where $OIR^* = C \times A^*$. From Figure 3 (b), we can see that OWL achieves similar accuracy to ARE. We can also expect OWL to generate better OIR in these two categories, due to its significant advantages in coverage.

At the {manufacturer, type} granularity, OIR* was calculated from 10,583 samples that have the {type} label. The other 15,895 samples did not contain enough information for human annotators to recognize their types. Therefore, A^* and OIR^* represent the upper-bound of the actual A and OIR. To estimate the lower-bound of A, we sanitized the 10,583 annotated samples by removing all textual features and MACs – now they provide even less information than the 15,985 unlabeled samples. OWL achieved 88.4% accuracy on sanitized data. A reasonable estimation is that $OIR \in [0.884, 0.975]$. From another angle, our true groundtruth dataset has very similar device/protocol distributions with testing data. OWL achieved 90.98% OIR on groundtruth data. Therefore, OIR on annotated data is expected to be similar: $OIR \approx 0.9098$.

Performance on Sanitized Data. We generated a sanitized dataset to test OWL's performance in extreme conditions, where all human-interpretable contents are removed from raw data. OWL's coverage, accuracy and OIR on the sanitized dataset are shown in Figure 3 (c). In particular, category 1 ({manufacturer}) was evaluated against all 26,478 sanitized samples. Category 2 was evaluated on 10,583 samples with {manufacturer, type} labels; while category 3 was evaluated against 4,064 samples with all three labels. OWL achieved 100% coverage in the later two categories, since the basic protocol features still existed after data sanitization. However, after removing MAC prefixes, some samples in category 1 cannot be identified since no meaningful feature was left. OWL's accuracy is still high, in the range of $[0.75, 0.88]$.

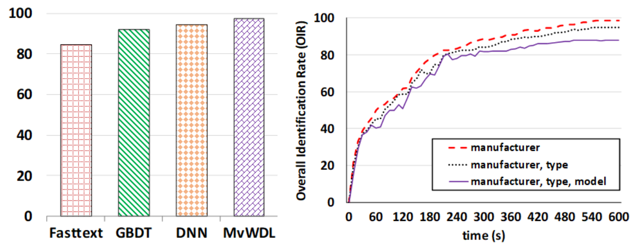
Performance Comparison with Other Classifiers. We have explained the rationale of choosing multi-view learning in Section 4.1. Meanwhile, the choice of specific classifier and fusion strategy is mostly empirical: (1) we have enough features and samples to support deep learning, which demonstrated superior performance in ML literature; (2) we need a late-fusion component to measure inconsistencies among views for anomaly detection; (3) we need to handle the different distinguishability of different protocols against different device types. Now we experimentally compare MvWDL with other popular classification algorithms. The Gradient Boosting Decision Tree (GBDT) [26] is among the best non-NN classifiers for categorical features. The fastText [31] is a state-of-art word embedding and classification library by Facebook. We also employ a generic deep neural network (DNN). We use 10-fold cross validation on annotated data, which has significantly more samples than other datasets. Figure 4 (a) shows the average accuracy of each classifier over all labels. MvWDL achieves the best performance, while DNN is a close second. fastText was the least accurate, which may be caused by the smaller training set than fastText's expectations.

Device Detection Speed. Another important metric is the time for OWL to recognize all the devices in a WiFi network. While it only takes mini-seconds for a trained MvWDL model to classify a new device, packets/features come to OWL slowly in real world settings. We tested the real-time perfor-



(a) (b) (c)

Figure 3: Experiment results: (a) Performance comparison of ARE [24], WDMTI [64] and OWL on *ground truth data* (X-axis: C: coverage; A: accuracy; OIR: overall identification rate). (b) Performance comparison on *annotated data*. * Note that accuracy and OIR was only evaluated on partial data in the later two categories (please see detailed discussions in Section 5). (c) Performance of OWL on *sanitized data*.



(a) (b)

Figure 4: (a) Performance comparison with other classification algorithms. (b) Device detection speed in ground truth data.

mance of OWL on the ground truth data. We assumed that OWL was connected to the network at t_0, and gradually fed packets to it according to their timestamps. We assessed OIR at 1-second intervals. As shown in Fig. 4 (b), OIR increased rapidly for approximately 240 seconds, when 80% of the devices were correctly identified at all granularity levels. OWL reached its peak performance in approximately 500 seconds.

Supplementary Data and Labelless Data. The supplementary testing data contains 78 device descriptions from URLs in SSDP notify messages. Device descriptions confirmed the classification of 63 devices, partially supported 14 devices (e.g. device labeled Samsung Galaxy while description says "Android"), and denied the classification of one device.

Last, devices in the labelless dataset did not provide enough features for labeling, however, we still attempted to validate the predicted labels with the limited information in the packets. 71% of the labels were supported, while only 1% of the labels were denied (e.g., a sample was classified as a camera but one packet contains keyword "windows").

Misclassified Devices. We manually examined the misclassified samples to identify their causes. First, most misclassifications at {model} level were classified into a similar model from the same vendor, e.g., Samsung Galaxy-note8 phones identified as Galaxy-note9, and HP M1536dnf printers identified as M227fdw. Many devices misclassified at {type} level were classified into the vendor's other product line, e.g., some Apple Watches with limited BC/MC packets were identified as iPhone. Finally, third-party WiFi modules of some devices caused confusions at {manufacturer} level.

Low confidence classifications are often unknown or forged devices, which may be sent to system admins for manual evaluation. Classifying them into "unknown" will decrease OWL's coverage but increase its accuracy. They are also very likely to be detected by OWL's malicious device identifier.

6 Malicious Device Identification

As discussed in Section 4, when a device demonstrates inconsistent features in different protocols, it could be malicious. OWL makes the first attempt to utilize view inconsistency to identify abnormal devices, as denoted in Eq. 15. We apply the algorithm on all the samples in our dataset, and manually examine the devices that trigger the alarm. We present three cases of suspicious devices identified in our experiments, as well as the case of (hidden) camera detection using OWL.

"Spoofed" AppleTVs. A group of 31 devices demonstrated similar abnormal behaviors that triggered the alarm. The `mDNS` view classified all these devices into AppleTV with strong confidence; however, none of the other views predicted these devices as AppleTV, and their confidence levels were all relatively high. We further manually examined these devices.

First, the devices were labeled as various models of TVs or stream casting receivers. Some samples were from the ground truth dataset so that their labels were physically checked with the device. Others were manually verified in the semi-automatic labeling process, especially, their MAC prefixes were consistent with the labeled manufacturers. Hence, the labels, as listed in Table 4, appear to be consistent with the actual device model. However, we further scrutinized the original `mDNS` packets from these devices, and confirmed that they are very similar to other AppleTV devices in our database.

Further investigation ties the behaviors from these devices to Apple's AirPlay feature. `AirPlay` is Apple's proprietary protocol suite for multimedia streaming over WiFi. Since Apple never open-sourced or licensed AirPlay, this feature is supposed to be seen on Apple devices only. However, the proprietary `AirPlay` protocol has been reverse engineered, and several open-source implementations are available on Github, e.g., `open-airplay`[4]. Our investigation also discover that the `AirPlay` protocol in all the malicious "counterfeit AppleTVs", except {MTN, TV}, was developed by a corpo-

[4] `https://github.com/jamesdlow/open-airplay/`

Table 4: Devices pretending to be AppleTVs.

Xiaomi,TV,4	Leshi,TV,x55	Leshi,TV,x65s
Gaoshengda,TV	Funshion,TV	Chuangwei,TV
Hisense,TV,vidaa	PPTV,TV	Changhong,TV,43s1
whaley,TV,w50j	MTN,TV	Changhong,TV,LED50
Rflink,TV	Nebula,TV	Tianmao,Magiccast,m18

ration named Lebo (or HappyCast) [5]. The homepage of Lebo corporation claims that they independently researched the protocol for casting streaming media from iOS system and developed the Lebo software suite.

For validation, we deployed the `open-airplay` library on a Windows laptop as a "simulator". Apple devices in the same network identified it as a valid AirPlay receiver. We captured BC/MC packets from the simulator, and further examined the `mDNS` packets. The `mDNS` Resource Records of the simulator, counterfeit AppleTVs, and the authentic AppleTVs were almost identical. The simulator and the counterfeit AppleTVs even shared higher similarities than that between the counterfeit and the authentic AppleTVs. All other BC/MC packets from the simulator behaved the same as the host laptop.

In this case, OWL was able to identify abnormal inconsistencies across views for a group of seemingly malicious devices. We discovered the root causes of the inconsistency through further manual investigation. Set aside legal implications of counterfeiting, this case demonstrates the capability of OWL in identifying spoofed devices in the real world.

Fake DHCP Server and Gateway. Another device in the labelless dataset also triggered the alarm in the experiment. The `DHCP` view labeled it as a *router* with high confidence, which was not agreed by other views. Further investigation showed that the device broadcast `DHCP Offer` and `DHCP ACK` messages to inform other devices the gateway of the network is itself. This behavior clearly resembled routers or gateways in WiFi networks. However, mDNS and SSDP views classified this device as a Microsoft laptop (model: surface_book). The MAC prefix also confirmed its vendor as Microsoft.

A reasonable explanation is that the Microsoft surface book spoofed a gateway to lure others to connect through it. Examination of the `DHCP request` packets from other devices in the network revealed that some devices did connect through this fake gateway, which could easily launch man-in-the-middle attacks, or use a captive portal to phish the victims.

To confirm our speculation, we simulated the same attack in our lab network. We employed Yersinia in Kali Linux to send `DHCP Discover` to exhaust the IP resource of the authentic router. DHCP service was then started on the Kali computer using itself as the gateway. Very soon, we observed new devices requesting IP addresses from the spoofed gateway. We sniffed the BC/MC packets from this gateway, and fed them to OWL, which generated an alarm that was very similar to the one for the rogue gateway in our dataset.

Virtual Machines. OWL identified several devices that demonstrated strong discrepancy between mDNS and LBN views. For example, several devices were identified as Macbooks on mDNS view and MAC prefix. Meanwhile, LBN view classified them to be computers manufactured by other vendors. Through further investigation, we concluded that these were computers running virtual machines that connected to the networks with *Network Address Translation* (NAT).

[5] `http://www.hpplay.com.cn/index_english.jsp`

In practice, a virtual machine has three mechanisms to connect to the network: (1) NAT, (2) bridged network, and (3) host-only network. With NAT, the VM and the host system share a single network identity, so that packets from the VM are directly disseminated by the host. With bridged network mode, the VM may get its own IP while sharing the same MAC with the host. The VM may also get its own MAC, where the VM vendor could be identified by the MAC prefix. For example, MAC prefix "00-05-69" denotes VMware and "00-1c-42" denotes Parallels. Last, VMs with host-only network only communicates within a private network on the host, hence, they do not connect to the external network at all.

When a VM runs in NAT mode or shared MAC in bridged mode, OWL is able to detect the inconsistencies caused by the shared identity. OWL is only effective when the guest OS differs from the host OS, so that discrepancies in the implementations of network protocols could be discovered. We further tested other guest/host OS combinations, including Android x86 VM running on MacBooks or Windows desktops, and confirmed that OWL was able to detect all of them given enough sniffed BC/MC packets. Last, for VMs with their own MAC addresses, they were correctly annotated as VMs in our dataset and accurately detected in the experiments.

Hidden Cameras. Surveillance cameras, especially the hidden ones, are often considered as sensitive/malicious devices that infringe users' privacy. Efforts have been made in the literature to detect hidden cameras based on their unique network traffic patterns during video streaming [10, 61, 61]. Meanwhile, we observed that the adversaries may set the (hidden) cameras to stand-by mode or to store videos locally to avoid traffic-based detectors. They only transmit real-time or stored video streams when they receive remote commands from their owners, who may pick a time when the victims' detectors are likely to be offline, e.g., late night or after hotel checkout. Nevertheless, these cameras still connect to the network in order to receive remote commands, therefore, they send out BC/MC packets and they can be detected by OWL.

In our experiments presented in Section 5, OWL achieved 100% accuracy in detecting cameras at {manufacturer, type} granularity, when the training set contains samples with the same {manufacturer, type}, but not necessarily the same model. For example, when we have `{dlink, camera, dcs-930lb}` in the training data, OWL can correctly identify DLink DCS-935l cameras as `{dlink, camera}`, even though it has never seen the DCS-935l model before, i.e., it does not have `{dcs-930lb}` in its label set. This is explained by the fact that the same manufacturer often reuses the hardware and software modules, especially for products in the same line. Meanwhile, OWL also identified several examples of OEM cameras in our dataset. For example, when we put only one camera `{lenovo, camera, snowman}` in the training data, a Xiaomi Dafang-DF3 camera and a Qihoo360 D302 camera were both classified as Lenovo cameras. Further examination of raw data confirmed that all three products shared nearly identical features in several views. Note that these two new cameras were also significantly different from other `Xiaomi` or `iQhoo360` devices. We could confidently infer that all three products shared certain software modules or they might be OEM devices from the same original manufacturer. Last, although they were correctly labeled as cameras, these devices also triggered alarms of unknown/malicious devices, which calls for the attention of the administrator or user.

7 Attacks Against OWL

In this section, we discuss three potential attacks against OWL: the naive attacks, the knowledgeable attacks, and the expert attacks. They share the same objective: to hide the identities of (potentially malicious) devices by confusing the device classifier and escaping from the malicious device detector.

7.1 The Naive Attacks

The Threat Model. The naive adversaries do not have the knowledge or capability (e.g., root privilege) to change system code/driver or privileged files/attributes. They can only employ OS-provided GUI to modify user-defined attributes that are adopted by the network modules. With the lowest technical barrier, naive attacks are highly feasible to novices.

The Approach. We examine the most popular OSs for consumer mobile/IoT devices (Android, iOS, Windows, and MacOS) to identify the system attributes that could be changed through system settings and then adopted in BC/MC packets. Naive attackers could configure the "device name" attribute in iOS (un-rooted) and MacOS (admin-only), which is adopted in the `HostName` field of DHCP, mDNS, and other protocols. Although users could change "device name" in Android, the attribute is only used as device identifier in Bluetooth, WLAN Direct, hotspot, and USB, while all BC/MC protocols use a manufacturer-assigned value in `HostName`. We also examine user settings of IoT devices in our lab and identify how the user-entered values are adopted in BC/MC messages. The devices and settings in IoT devices are more ad-hoc, as users could only change one or two attributes in a few devices that impact the MC/BC packets (mostly `HostName`). Note that we do not consider virtual machines, use of hacking tools or command line methods in the naive attacks.

Experiment Results. In the experiments, we randomly selected 1,000 devices with user-editable attributes from the annotated dataset with all {manufacturer, type, model} labels. For each device, we overwrite all user-editable attributes with values from another random device with different labels.

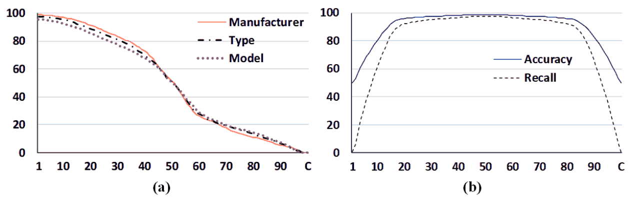


Figure 5: The knowledgeable attacks: (a) Device identification accuracy of OWL under attack. (b) Malicious device detection accuracy and recall of OWL. X-Axis: percentage of modified features.

As a result, OWL achieved $OIR = 0.985$, $OIR = 0.964$ and $OIR = 0.902$ at three granularity levels, respectively.

7.2 The Knowledgeable Attacks

The Threat Model. The knowledgeable adversaries have full control of the system and understand OS hacking. However, they do not have the system source code (e.g., Windows or proprietary IoT devices), so that they need to reverse engineer the system or to hack on OS/application binaries. Therefore, it could be challenging to completely overwrite *all attributes* from all BC/MC protocols, since the attributes could be derived or scatteredly distributed in the system. This represents the majority of the advanced adversaries against OWL.

The Approach. The knowledgeable adversaries always attempt to forge a specific device instead of randomly modifying each attribute, since this gives them the best chance to escape from correct identification. Formally, an adversary attempts to hide a suspicious device S by replacing n attributes (out of N total attributes) from its BC/MC packets with values from a benign device B. We want to answer two questions through experiments: (1) When n increases from 0 to N, how would OWL's device identification performance change? (2) How would OWL detect the suspiciously altered device?

Experiment Results. In the experiments, we randomly sampled 1000 devices with all three labels from the annotated data set. For each device S, we randomly selected another device B from a different {type}, and overwrote n attributes of S with corresponding values from B. Figure 5 (a) shows the device identification accuracy of OWL at three different granularity levels. When 20% of the features of S are overwritten by values from B, OWL's accuracy drops to 91.5%, 88.9% and 85% for manufacturer, type, and model, respectively.

We added 1,000 random benign devices to the above dataset to serve as negative samples. Recall (aka. detection rate) R is defined as: $R = \frac{TP}{TP+FN}$, i.e., ratio of correctly detected malicious samples out of all malicious samples. Accuracy A is defined as $A = \frac{TP+TN}{ALL}$, i.e. the ratio of corrected classified samples out of all samples. Figure 5 (b) shows the malicious device detection performance under the knowledgeable attacks. When 20% of the attributes are modified, R reaches 92.2% while the A is 95.95%. When majority or all of the features are modified ($n \to N$), S essentially becomes (almost) identical to B, hence, both A and R drops.

Table 5: OWL's performance against the expert attacks.

$\#_{view}$	1	2	3	4	5	6
A_{manuf}	.929	.806	.509	.208	.101	.001
A_{type}	.893	.793	.505	.212	.118	0
A_{model}	.878	.786	.502	.227	.136	0
A	.952	.977	.985	.976	.950	.500
R	.906	.957	.973	.955	.903	.003

7.3 The Expert Attacks

The expert adversaries have full control of the OS and they are capable of making arbitrary changes to the system. They can override all attributes of any BC/MC protocol from a (suspicious) device S with attributes from a benign device B.

Rows 2 to 4 of Table 5 show OWL's device classification accuracy against the expert attacks on 1,000 devices, when $n \in [1,6]$ randomly selected views are forged. When 3 or more views are forged, OWL's accuracy drops significantly at all levels. Rows 5 to 6 show OWL's malicious device detection performance against the set of 1,000 benign and 1,000 attacked devices. Both accuracy and recall peaks when half of the views are forged. When a malicious device successfully mimics all 6 views, its MC/BC packets essentially becomes identical to a benign device. Hence, OWL's detection recall drops to almost 0, while accuracy drops to 0.5, i.e., benign devices are correctly identified as negative (true negative), while malicious devices are also classified as negative (false negative). However, when the adversary fakes 5 views but misses one, OWL effectively detects the malicious device.

We also discuss two weaker versions of expert attacks: (1) MAC modifiers: privileged users may easily modify devices' MAC addresses, which is equivalent to overwriting half of View #6. (2) Software installers: privileged users may install software(s), e.g., DHCP server, which interferes with the device's native fingerprint. Note that the original BC/MC packets from the device stay intact. This is equivalent to partially modifying view(s). In our experiments, OWL's performance against both attacks is similar to the performance against expert attacks with same number of affected views.

Adversarial machine learning could be employed to attack the MvWDL model. However, attackers need to carefully engineer the BC/MC packets to generate adversarial attributes. In practice, it could be easier to perform expert attacks to overwrite all BC/MC packets in a device to hide its identity.

8 Discussions

In this section, we discuss several important issues: the security properties of malicious device detection, the undetectable devices, performance tradeoffs, OWL's limitations, etc.

Security Analysis. We first provide a brief security analysis of OWL, corresponding to the adversary model and two types of malicious devices introduced in Section 2. (1) For genuine devices that are disallowed in a network, such as hidden cameras or unauthorized routers, OWL detects them with high coverage and accuracy, as shown in Sections 5 and 6. More discussions on performance trade-offs and undetectable devices will be presented in this section. (2) When a fabricated, counterfeit, or forged device attempts to hide its original identity (e.g., laptop) and report a fake identity (e.g., network gateway), it is difficult to completely and accurately forge the entire software/protocol suite at all layers. OWL detects the subtle discrepancies among features from essential network protocols. Meanwhile, virtual devices are often employed in real-world attacks [52]. In OWL, virtual devices are either correctly detected as VMs, or trigger alarms due to dual identities on the same MAC, as discussed in Section 6. Effectiveness of OWL is shown in theory in Section 4, and demonstrated by case studies in Section 6. However, due to the unavailability of ground truth data, we are unable evaluate the recall of malicious device detection. Note that the threshold ε in Equation 15 could be tuned with real world data to improve detection rate. Meanwhile, false positives in malicious device detection are more tolerable than false positives in network IDS, since the number of devices in the WiFi network is significantly smaller, so that a few false alarms would not exhaust network administrators. Meanwhile, false alarms (mostly unknown devices) could be labeled and used to retrain OWL to improve classification accuracy and reduce false positives.

Silent Devices. Some devices are *silent* that they do not actively send BC/MC packet. They pose challenges to OWL's coverage. We discuss this issue from two aspects: (1) completely quiet devices, i.e., devices that send nothing at all, are very rare. In the controlled environment, we compared the ground truth data with the DHCP allocation table. We found that laptops in sleep mode were the only devices that did not generate any packet. Moreover, three TPLink smart switches/plugs only sent DHCP packets when they first connected to the network, and kept silent for more than 30 minutes thereafter. We also observed that printers sent BC/MC messages even in sleep mode. (2) Devices-of-interest are mostly not quiet. In our ground truth data, devices with more computational power and devices that are discoverable in the network all kept sending BC/MC packets. We have 93 cameras that sent BC/MC packets in our dataset. Meanwhile, we further examined 20+ popular webcams on the market and they all continuously transmit BC/MC packets, even when they were not capturing video streams. Last, some public wireless networks restrict BC/MC packets from being delivered to the network. In such networks, regular users cannot employ OWL to explore other (malicious) devices, however, system administrators could still deploy OWL for device identification, e.g., by mirroring traffic to the sniffing device.

The Trade-off between Coverage and Accuracy Besides the silent devices, some devices only send very few packets in their regular operation mode. For instance, 34.1% of the devices in our dataset only sent BC/MC packets in one protocol during our data collection process (Figure 1 (c)). Further examination showed that ARP, mDNS, DHCP and SSDP protocols were used in 26.9%, 31.3%, 6.8%, and 4.3% of such devices, respectively. Even with one packet, MvWDL could extract features (protocol features and MAC prefix) and classify the device into a known label. However, in some cases, this classification is like an "educated guess" with relatively low confidence. In practice, mDNS and DHCP protocols are both very informative, while `ARP` packets do not carry any device-specific information except MAC prefix. In OWL, there is a trade-off between coverage and accuracy: excluding the featureless devices will increase OWL's accuracy, but decrease its coverage. We can define various heuristics to identify featureless devices, for example, utilizing trivial features (as we have done in Section 5.1), or using simple rules on feature count and feature types, etc. We do not further elaborate on this aspect since it is more performance tuning than technical contribution. Last, devices frequently join and leave the network during our data collection process. When a device happened to join at the end of a data collection session, we were less likely to get full set of features. If we sniffed for a little longer, we would have obtained more features.

MAC Randomization. To defend against device tracking attacks (e.g. [14]), MAC randomization has been employed by mobile devices to hide their universally administered MAC addresses (UAAs) [39, 58]. As discussed in [39] and verified with our experiments on iOS and Android devices, MAC randomization is only employed at probing – randomized, locally administered addresses (LAA) are used in probe frames. When devices are associated to APs, their UAAs are used for all subsequent communications. In theory, a locally assigned MAC, identified by its 7th bit of the most significant byte, cannot be used in non-local communications since they are not guaranteed to be unique. In OWL, packets are only collected after devices are associated with APs, hence, OWL always sees real MACs. However, [39] noticed a portion of Windows/Linux devices using LAAs when associated to networks. This is also confirmed in our dataset: we found 140 devices (out of 31,850 devices) using locally assigned MACs.

Unicast Traffic. OWL solely relies on BC/MC traffic. Unicast traffic has been used for device identification in the literature [24, 38, 57]. Unicast packets could be obtained using active probing or eavesdropped at the gateways. Although not available to normal users in the network, administrators may extract additional features from unicast traffic, such as timing and flow features, application-layer protocols, DNS, TLS handshakes, and textual features (banner grabbing). OWL may be extended to: (1) extract features from unicast traffic (such as [24, 42, 57]), and (2) add unicast-based views to the MvWDL model to improve identification performance. However, consider the overhead to monitor unicast packets, it may not be cost-effective to utilize them for device identification, since OWL already provides very high OIR. However, using unicast traffic enables the detection of software anomalies, which are usually not detectable from BC/MC protocols.

Additional Info from BC/MC Packets. Besides manufacturer, type and model, we also found other information in BC/MC packets that could be of interest to sys-admins. For instance, we identified 6,343 devices with OS name, and 474 more with OS version, either from the textual information revealed in BC/MC packets, or discovered from devices that run only one OS, such as iOS on Apple phones. We extended the MvWDL model with new labels, and tested the results on our ground truth data. OWL's identification accuracy was 98.2% on OS names and 78.4% on version. We also applied keyword spotting in the sniffed packets, and found 8 types of browsers from 1,021 devices and 9 video streaming applications from 101 devices, mostly from the `User Agent` field in `SSDP`. As an application of OWL, we developed a tool to match discovered devices, OS, and applications with CVE database. Examples of potential device vulnerabilities and OS vulnerabilities are shown in Tables 6 and 7. In practice, such information is very useful to network administrators.

Limitations. Finally, we also acknowledge that the current implementation of OWL has its own limitations, especially: (1) Expert adversaries may escape from OWL by perfectly mimicking all BC/MC packets from another device. For example, if an emulator runs the genuine Samsung version of Android including the original network module and full protocol stack, fakes a Samsung MAC, and ensures that the communication is not interfered by the host machine, OWL would report it as a benign Samsung device. (2) OWL is not designed to detect software or application anomalies that do not show any symptom in BC/MC traffic. However, as discussed above, it is practical to include unicast traffic as additional views in MvWDL, which has the potential to detect devices that generate abnormal unicast traffic. (3) OWL will label new devices (without any similar device in the training set) as malicious devices. (4) The current design of OWL cannot handle adversarial ML attacks against the MvWDL model.

9 Related Works

The problem of device identification has been studied from various angles. Earlier approaches focus on fingerprinting or authenticating individual devices. Various hardware features have been used, such as clock skew [2, 16, 29, 32, 50], RF parameters [46, 51], sensor imperfection/noise [3, 7, 15], etc. A survey of wireless device fingerprinting is available at [62]. Devices are also identified from software features, e.g., [21, 27, 34, 35, 48, 59]. OWL is significantly different from this group of approaches in objective, data, and methodology.

Network traffic has been used for both hardware and software identification, such as network modules or OS [12, 13, 25, 54, 55, 58]. The Internet-wide IoT device discovery approaches, such as Censys [18], Nmap [37], SHODAN,ZMap [20], and others [1, 19, 22, 63] mostly use banner grabbing to actively scan for devices in the IP space, collect and examine textual features such as hard-coded keywords from responses, and match them against known fingerprints. To

Table 6: CVE instances for collected device-types. (Score: CVE score; #: Device count)

manufacturer-type	CVE ID	score	Vulnerability	#
huawei_phone_p9	CVE-2016-8759	9.3	allow attackers to crash the system or escalate user privilege	25
huawei_phone_mate9pro	CVE-2017-17320	9.3	allow attackers to execute malicious code	18
philips_bridge_huebridge	CVE-2017-14797	7.9	allow remote attackers to read API keys	19
osram_light_lightify-home	CVE-2016-5053	7.5	allow remote attackers to execute arbitrary commands	6
samsung_phone_galaxy-s6-edge	CVE-2015-7888	7.8	allow remote attackers to create arbitrary files as the system-level user	4
apple_pad_ipad	CVE-2013-3955	6.2	allow local users to cause a denial of service	15

Table 7: OS-related CVE instances relevant to our dataset.

OS	CVE ID	CVE score	Vulnerability	device count
android	CVE-2018-9355	10	allow attackers to execute remote code with no additional execution privileges needed	1670
ios	CVE-2018-4337	9.3	a memory corruption issue was addressed with improved memory handling	1690
linux	CVE-2019-11683	10	allows remote attackers to cause a denial of service	129
mac_os_x	CVE-2018-4259	10	multiple memory corruption issues	530
ubuntu	CVE-2018-7584	7.5	allow attackers to copy a large string from stack-based buffer	11

tackle the scalability issue of manual labeling, ARE [24] creates a rule-discovery engine to mine labeling rules from text corpora crawled from the web. OWL is different from device discovery approaches that we only rely on passive traffic. Active port scanning may be considered intrusive and forbidden in many networks. OWL does not interrupt with the normal operations of devices, nor does it intercept any peer-to-peer traffic between the devices and their owners or clouds.

IoT Sentinel [43] extracts features from network traffic and utilize a random forest classifier to identify device types. Similarly, [53] employed six different classifiers on packets streams for device type recognition. AuDI [38] developed an unsupervised learning approach to cluster same type/model of devices, without any labeled data. [42] identifies IoT device types in a whitelist using features from TCP sessions. [28] examines the network flows to the vendors' servers for device identification. [36] utilizes the requested domain names in DNS traffic for the identification of vendors and device types. WDMTI [64] uses Hierarchical Dirichlet Process on DHCP features to classify IoT device types. OWL is different from existing approaches that: (1) OWL utilizes passively collected BC/MC packets, which does not require privileged access to the network or using monitor mode at the WiFi adapter. Existing approaches, except WDMTI, use peer-to-peer traffic. (2) OWL integrates two important network management functions, device identification and abnormal device detection, into one comprehensive solution. And (3) OWL is tested on a significantly larger dataset at three granularity levels.

Malicious IoT device detection have been studied [17, 41, 47, 49]. OWL is essentially different from them in the objectives–OWL detects fabricated or forged devices, while they mostly focus on devices' adversarial behaviors.

10 Conclusion

In this paper, we present a novel mobile/IoT device identification and abnormal device detection mechanism named OWL. OWL extracts features from structural and textual information embedded in the BC/MC packets. A multi-view wide and deep learning (MvWDL) model is designed to identify the manufacturer, type and model of devices. Meanwhile, OWL also discovers the subtle evidence of inherent discrepancies across views to detect fabricated/forged devices. Through large-scale experiments, we show that OWL outperforms existing approaches in the literature in accuracy and coverage.

References

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, A. Halderman, L. Invernizzi, M. Kallitsis, et al. Understanding the mirai botnet. In *USENIX Security*, 2017.

[2] Chrisil Arackaparambil, Sergey Bratus, Anna Shubina, and David Kotz. On the reliability of wireless fingerprinting using clock skews. In *ACM WiSec*, 2010.

[3] Zhongjie Ba, Sixu Piao, Xinwen Fu, Dimitrios Koutsonikolas, Aziz Mohaisen, and Kui Ren. Abc: enabling smartphone authentication with built-in camera. In *NDSS*, 2018.

[4] A Bartoli, J Hernández-Serrano, M Soriano, M Dohler, A Kountouris, and D Barthel. Security and privacy in your smart city. In *Proceedings of the Barcelona smart cities congress*, volume 292, pages 1–6, 2011.

[5] Ron Bitton, Andrey Finkelshtein, Lior Sidi, Rami Puzis, Lior Rokach, and Asaf Shabtai. Taxonomy of mobile users' security awareness. *Computers & Security*, 73:266–293, 2018.

[6] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[7] D. Chen, N. Zhang, Z. Qin, X. Mao, Z. Qin, X. Shen, and X. Li. S2m: A lightweight acoustic fingerprints-based wireless device authentication protocol. *IEEE Internet of Things Journal*, 4(1):88–100, 2016.

[8] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir,

et al. Wide & deep learning for recommender systems. In *ACM RecSys Workshop on DLRS*, 2016.

[9] Ningning Cheng, Xinlei Oscar Wang, Wei Cheng, Prasant Mohapatra, and Aruna Seneviratne. Characterizing privacy leakage of public wifi networks for users on travel. In *IEEE INFOCOM*, pages 2769–2777, 2013.

[10] Yushi Cheng, Xiaoyu Ji, Tianyang Lu, and Wenyuan Xu. Dewicam: Detecting hidden wireless cameras via smartphones. In *ACM AsiaCCS*, pages 1–13, 2018.

[11] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2017–2022. Cisco White Paper.

[12] Cherita Corbett, Raheem Beyah, and John Copeland. A passive approach to wireless nic identification. In *IEEE International Conference on Communications*, pages 2329–2334, 2006.

[13] Cherita L. Corbett, Raheem A. Beyah, and John A. Copeland. Passive classification of wireless nics during rate switching. *Eurasip Journal on Wireless Communications & Networking*, 2008(1):1–12, 2007.

[14] Mathieu Cunche. I know your mac address: Targeted tracking of individual using wi-fi. *Journal of Computer Virology and Hacking Techniques*, 10(4), 2014.

[15] A. Das, N. Borisov, and M. Caesar. Do you hear what i hear?: Fingerprinting smart devices through embedded acoustic components. In *ACM CCS*, 2014.

[16] L. Desmond, C. Yuan, C. Tan, and R. Lee. Identifying unique devices through wireless fingerprinting. In *ACM WiSec*, pages 46–55, 2008.

[17] Rohan Doshi, Noah Apthorpe, and Nick Feamster. Machine learning ddos detection for consumer internet of things devices. In *IEEE S&P Workshops*, 2018.

[18] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. A search engine backed by internet-wide scanning. In *ACM CCS*, 2015.

[19] Zakir Durumeric, Michael Bailey, and J Alex Halderman. An internet-wide view of internet-wide scanning. In *USENIX Security*, pages 65–78, 2014.

[20] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In *USENIX Security*, pages 605–620, 2013.

[21] Peter Eckersley. How unique is your web browser? In *PETS*, pages 1–18, 2010.

[22] Claude Fachkha, Elias Bouharb, Anastasis Keliris, Nasir D Memon, and Mustaque Ahamad. Internet-scale probing of cps: Inference, characterization and orchestration analysis. In *NDSS*, 2017.

[23] X. Feng, X. Liao, X. Wang, H. Wang, Q. Li, K. Yang, H. Zhu, and L. Sun. Understanding and securing device vulnerabilities through automated bug report analysis. In *USENIX Security*, pages 887–903, 2019.

[24] Xuan Feng, Qiang Li, Haining Wang, and Limin Sun. Acquisitional rule-based engine for discovering internet-of-things devices. In *USENIX Security*, 2018.

[25] Jason Franklin, Damon Mccoy, Parisa Tabriz, Vicentiu Neagoe, Jamie Van Randwyk, and Douglas Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *USENIX Security*, 2006.

[26] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 2001.

[27] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In *WWW*, pages 309–318, 2018.

[28] Hang Guo and John Heidemann. Ip-based iot device detection. In *Workshop on IoT Security and Privacy*, pages 36–42, 2018.

[29] Suman Jana and Sneha K. Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. In *ACM MobiCom*, 2008.

[30] Beth H Jones and Amita Goyal Chin. On the efficacy of smartphone security: a critical analysis of modifications in business students' practices over time. *International Journal of Information Management*, 35(5), 2015.

[31] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[32] Tadayoshi Kohno, Andre Broido, and K. C Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable & Secure Computing*, 2(2), 2005.

[33] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric. All things considered: An analysis of iot devices on home networks. In *USENIX Security*, pages 1169–1185, 2019.

[34] Andreas Kurtz, Hugo Gascon, Tobias Becker, Konrad Rieck, and Felix Freiling. Fingerprinting mobile devices using personalized configurations. *PETS*, 2016.

[35] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *IEEE S&P*, pages 878–894, 2016.

[36] Franck Le, Jorge Ortiz, Dinesh Verma, and Dilip Kandlur. Policy-based identification of iot devices' vendor and type by dns traffic analysis. In *Policy-Based Autonomic Data Governance*, pages 180–201. 2019.

[37] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

[38] Samuel Marchal, Markus Miettinen, Thien Duc Nguyen, Ahmad-Reza Sadeghi, and N Asokan. Audi: Toward autonomous iot device-type identification using periodic communication. *IEEE JSAC*, 37(6):1402–1412, 2019.

[39] Jeremy Martin, Travis Mayberry, Collin Donahue, Lucas Foppe, Lamont Brown, Chadwick Riggins, Erik C Rye, and Dane Brown. A study of mac address randomization in mobile devices and when it fails. *PETS*, 2017.

[40] Jeremy Martin, Erik C Rye, and Robert Beverly. Decomposition of mac address structure for granular device inference. pages 78–88, 2016.

[41] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici. N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22, 2018.

[42] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, and Y. Elovici. Detection of unauthorized IoT devices using machine learning techniques. *arXiv preprint arXiv:1709.04647*, 2017.

[43] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. Iot sentinel: Automated device-type identification for security enforcement in iot. In *IEEE ICDCS*, 2017.

[44] Tomas Mikolov, Kai Chen, Gregory S Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv:1301.3781*, 2013.

[45] Keith W Miller, Jeffrey Voas, and George F Hurlburt. Byod: Security and privacy considerations. *IT Professional*, 14(5):53–55, 2012.

[46] N. T. Nguyen, G. Zheng, Z. Han, and R. Zheng. Device fingerprinting to enhance wireless security using nonparametric bayesian method. In *INFOCOM*, 2011.

[47] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N Asokan, and Ahmad-Reza Sadeghi. DÏoT: A federated self-learning anomaly detection system for IoT. *IEEE ICICS*, 2019.

[48] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *IEEE S&P*, 2013.

[49] Jesus Pacheco and Salim Hariri. Anomaly behavior analysis for iot sensors. *Transactions on Emerging Telecommunications Technologies*, 29(4):e3188, 2018.

[50] Sakthi Vignesh Radhakrishnan, A. Selcuk Uluagac, and Raheem Beyah. Gtid: A technique for physical device and device type fingerprinting. *IEEE TDSC*, 12(5), 2015.

[51] Saeed Ur Rehman, Kevin W Sowerby, and Colin Coghill. Analysis of impersonation attacks on systems using rf fingerprinting and low-end receivers. *Journal of Computer and System Sciences*, 80(3):591–601, 2014.

[52] SecureWorks. Virtual machines used to hide activity. https://www.secureworks.com/blog/virtual-machines-used-to-hide-activity, 2016.

[53] Mustafizur R Shahid, Gregory Blanc, Zonghua Zhang, and Hervé Debar. Iot devices recognition through network traffic analysis. In *IEEE Big Data*, 2018.

[54] Zain Shamsi, Daren B H Cline, and Dmitri Loguinov. Faulds: A non-parametric iterative classifier for internet-wide os fingerprinting. pages 971–982, 2017.

[55] Zain Shamsi, Ankur Nandwani, Derek Leonard, and Dmitri Loguinov. Hershel: single-packet os fingerprinting. *Measurement and Modeling of Computer Systems*, 42(1):195–206, 2014.

[56] R. Srikant and J. F. Naughton. *Fast algorithms for mining association rules and sequential patterns*. 1996.

[57] Jianhua Sun, Kun Sun, and Chris Shenefiel. Automated iot device fingerprinting through encrypted stream classification. In *SecureComm*, 2019.

[58] M. Vanhoef, C. Matte, M. Cunche, L. S. Cardoso, and F. Piessens. Why MAC address randomization is not enough: An analysis of Wi-Fi network discovery mechanisms. In *ACM CCS*, 2016.

[59] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. Fp-stalker: Tracking browser fingerprint evolutions. In *IEEE Security and Privacy*, 2018.

[60] Ning Wang, Long Jiao, Pu Wang, Monireh Dabaghchian, and Kai Zeng. Efficient identity spoofing attack detection for iot in mm-wave and massive mimo 5g communication. In *IEEE GLOBECOM*, pages 1–6, 2018.

[61] Kevin Wu and Brent Lagesse. Do you see what i see? detecting hidden streaming cameras through similarity of simultaneous observation. In *IEEE PerCom*, 2019.

[62] Qiang Xu, Rong Zheng, Walid Saad, and Zhu Han. Device fingerprinting in wireless networks: Challenges and opportunities. *IEEE Communications Surveys & Tutorials*, 18(1):94–104, 2015.

[63] Kai Yang, Qiang Li, and Limin Sun. Towards automatic fingerprinting of iot devices in the cyberspace. *Computer Networks*, 148:318–327, 2019.

[64] Lingjing Yu, Tao Liu, Zhaoyu Zhou, Yujia Zhu, Qingyun Liu, and Jianlong Tan. WDMTI: wireless device manufacturer and type identification using hierarchical dirichlet process. In *IEEE MASS*, 2018.

[65] K. Zeng, K. Govindan, and P. Mohapatra. Non-cryptographic authentication and identification in wireless networks [security and privacy in emerging wireless networks]. *IEEE Wireless Communications*, 2010.