CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Adversarial Black-Box Attacks in the Domain of Device Fingerprints

Master's thesis in Computer Science and Engineering

Joel Andersson
Gustav Örtenberg

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Adversarial Black-Box Attacks in the Domain of Device Fingerprints

Joel Andersson
Gustav Örtenberg

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Adversarial Black-Box Attacks in the Domain of Device Fingerprints

Joel Andersson
Gustav Örtenberg

Supervisor: Gerardo Schneider , Department of Computer Science and Engineering, Formal Methods Division
Advisor: Gökhan Berberoglu, Software Architect at Cyxtera
Examiner: Devdatt Dubhashi, Department of Computer Science and Engineering, Data Science and AI Division

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover:
Illustration of an Android devise that has disguised itself with an "APPLE" label.

Typeset in LaTeX
Gothenburg, Sweden 2020

Adversarial Black-Box Attacks in the Domain of Device Fingerprints

Joel Andersson
Gustav Örtenberg
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Network security products incorporate many different tools in order to secure large networks. State-of-the-art products often utilize machine learning in order to classify devices connected to a network to assign them different levels of trust without the need for authentication. These zero-configuration security mechanisms work similarly to image classifying Deep Neural Networks and are of interest for big organizations where large amounts of devices come and go every day. However, solutions leveraging the power of machine learning also inherit its vulnerability to adversarial samples. Previous work has shown that even in query-limited black-box scenarios, which is the most limiting for an attacker, image classifiers are vulnerable to adversarial attacks that make use of specially crafted input vectors [24]. This study shows that known attack techniques against image classifiers can be successfully reapplied to classifiers in the domain of device fingerprints in computer networks. We provide proof of concept that previously discovered adversarial sampling techniques are applicable in the domain of device fingerprints by attacking a well known commercial classifier. We show that across ten different devices on average 9.9% of the adversarial samples were successfully misclassified by the classifier. The most prominent of those devices had 36% of its adversarial samples misclassified. These results point to the need for more sophisticated training algorithms as well as the importance of not building solutions that builds on trusting device- or user-supplied data.

Keywords: Adversarial Machine Learning, Adversarial Samples, Black-Box Attack, Device Fingerprinting, Network Packet Sniffing, Network Security, Transferability.

Acknowledgements

We want to thank everyone who has been supporting us throughout our thesis, partners, pets, and family in addition to all the lovely employees at Cyxtera, who all helped us in various ways. But we would also like to make some dedicated thanks to everyone who has been directly helping us in our work:

Devdatt Dubhashi for recommending us the paper that our thesis rests heavily on and for providing crucial guidance mid-way through the thesis, that helped us realign our course.
Gerardo Schneider for invaluable feedback on our report and guidance, especially in the early stages of the project.
Gökhan Berberoglu for helping us getting access to all the resources and contact with all people that made this thesis possible.
Martín Ochoa for expert guidance on the technical parts of the thesis and feedback on our report and our day to day operations.

Joel Andersson, Gothenburg, August 2020
Gustav Örtenberg, Gothenburg, August 2020

Contents

List of Figures

List of Tables

1
Introduction

Computer network security has been a growing concern for many companies due to the tremendous growth rate of internet-connected services. While firewalls might have been sufficiently secure in the past, modern computer usage at companies calls for security measures to be implemented at the endpoints. In state-of-the-art network security, software clients installed at each endpoint may be used to enforce network perimeters within the computer network itself, controlling which devices can interact and communicate. However, this approach is not viable for the category of devices that provide limited to no capability for software additions. Internet of Things (IoT) devices are a part of that category due to their price-performance trade-off, with low prices combined with so-called smart functionality causing a rapid influx of them. Additionally, due to market pressure, IoT devices commonly lack proper security measures as these are considered expensive and time-consuming to incorporate, making the devices easy targets for attackers [5, 28, 17, 8].

However, the access rights of devices within this category may still need to be controlled. For instance, if a smart tea kettle and a printer were to have access to the same network, the kettle could potentially exfiltrate sensitive corporate data that was sent to the printer. On a home network, this can be solved by manually configuring access rights for each device due to the limited number (3.6 per capita by 2022) of connected devices [4]. However, such a solution does not scale to the enterprise setting, and therefore needs another solution. One such solution would be to incorporate device fingerprinting to profile devices connected to the network and assign them a device type. The device's type can then be used to enforce access control policies and limit what network-connected resources it has access to [13]. Network-based device fingerprinting is done by analyzing data emitted from a device and reading specific fields of it to identify that device. This procedure combines different observable characteristics of a device (e.g., MAC-address, TCP Options) to assess what hardware and software are running on the device. Those characteristics are a result of the underlying hardware and software implementations.

Prior work has shown that machine learning can be used for accurate classification of devices [21, 20]. Simultaneously, adversarial attacks against ML classifiers have been successfully demonstrated [11, 18], proving to render the classification ability of an ML classifier useless[10]. Papernot et al. [24] proved that, even in a black-box setting, their adversarial attack could achieve misclassification rates as high as **97.72%** against the Google Cloud Predictions API. They performed attacks against image classification systems where added noise, unnoticeable to the human

eye, can change the perceived class of the original image [10].

Incorrect classification would naturally be problematic for device identification systems as well, as this leads to unintended policies being enforced to devices if classified erroneously. Implications of such include non-authorized access to computer network resources, additional attack surfaces, and possibly the ability to authorize additional devices. In general, the consequences of a successful adversarial change in device identification all come down to the level of trust applied by the implementation of such a platform.

1.1 Problem description

This work aims to study the transferability of adversarial sampling techniques to the domain of device fingerprints. To the best of our knowledge, there is no prior research on whether adversarial sampling techniques work in the device fingerprinting setting. Showing that these techniques transfer to the new domain could open up avenues for crafting more subtle spoofing attacks against fingerprinters that might be more difficult to detect, as well as aiding the prevention of such attacks. Additionally, the proof would show weaknesses in the classifiers used today.

We believe that device classifiers are just as vulnerable to adversarial attacks as image classifiers. This leads us to think that it might also be possible to use the same techniques for generating adversarial samples in the domain of device fingerprints. Therefore, we would like to examine how well previously known methods for adversarial sampling work in this domain. Additionally, we would like to examine the possibility of converting the perturbation of an adversarial fingerprint into a sequence of network packets that could be interlaced with the original trace of network packets to produce that same adversarial fingerprint.

To paint a picture of an attack's layout, we present a hypothetical scenario in which a system is installed at the site of a company that our attacker wants to target. The system is a commercial all-in-one solution for all sorts of on-premise network security-related issues. One of the tools provided by the system is a network access control management tool. It allows the company to set up rules for who is allowed to access what devices on the network.

For example, a rule could be that everyone who works in department X is allowed access to all devices at department X and the printer of department Y. In order to define who works at department X, the system uses a client that is installed at each employee's computer in which they can authorize themselves to the system, which gives them their appropriate department access. However, for some devices, there is no existing software client, and instead, access is determined based on the device's type, e.g., for allowing smartphones to connect to printers. If an attacker can access the smartphone network and thus the printers, they might also be able to access confidential documents sent to the printers.

The attacker comes up with the devious plan to give away free IoT devices that have their malicious code on them. The code is built to do two things, firstly, emit network packets that trick the network security system into believing that the IoT device is a smartphone. Secondly, find printers on the network and then forward a copy of every document to the attacker. In order to enact their plan, the attacker analyses the security product and finds that it is vulnerable to adversarial attacks against the device classifier. The security product is expensive, so the attacker does not want to buy one for analysis. However, there is a free trial version that provides access to the same API but with a limited number of queries. The attacker uses the free API in order to create an adversarial sample that they can use to trick the system into believing that their IoT device is a smartphone, and using that sample, the attacker is ready to launch the attack.

1.1.1 Partnering Company

The work is conducted together with our industrial partner, Cyxtera, a company specializing in developing network security products [7]. They are interested in incorporating machine learning into new network security solutions and especially the effects of doing so. Before applying significant changes and additions to these types of products, their policy is to know and understand the potential vulnerabilities associated with the changes in design and behavior. If a machine learning-based device identification can be circumvented by a single well-crafted adversarial input, then more harm than good might be added to the system. Possibly more important is understanding the impacts caused by exploiting these design flaws. This is why both researchers and developers employed at Cyxtera has aided our work by supplying us with office space, computing resources, data, and knowledge, in order to make the most out of the results created from this report.

1.2 Research Objectives

In our thesis, we set up three null hypotheses to prove that existing adversarial sampling techniques in other problem domains can be utilized in the fingerprint domain. Disproving the null hypothesis defined in this section shows that the alternate hypothesis, which we define as the contradictory statement to the null hypothesis, holds.

H1: It is not possible to transfer the principles of adversarial black-box attacks from the image recognition domain to the network fingerprint domain.

H2: Generated adversarial samples will not successfully be interpreted by the device classifier as the target device.

H3: It is not possible to fabricate a network packet sequence that yields the same fingerprint as the generated adversarial sample, while still containing the network packets that yielded the benign fingerprint, which the adversarial sample itself was based on.

1.3 Research Methodology

We elected controlled experiments as our research strategy since our hypothesizes are exploratory, can be answered deductively, and lend themselves to be analyzed quantitatively. Controlled experiments aim to test a hypothesis by altering one or more independent variables and measuring how it affects one or more dependent variables. Each combination of independent variables is considered as a *treatment* and a study must contain at least two treatments that are compared to each other. In our work, we primarily employ the research strategy to hypothesis **H2**, where we compare the output from the device classifier, our dependent variable, between our two treatment groups, with or without adversarial perturbations added to the original fingerprint.

The adversarial samples should be humanly indistinguishable from samples of regular network traffic information and able to convert into a series of network packets interlaced with regular unperturbed traffic. Furthermore, we consider an attack successful if a device gets erroneously classified as another target device. The target device is preferably one known to have a high level of trust assigned to it.

In order to achieve the most realistic scenario, we will assume that the attack targets a commercial classifier. We also assume a query-limited approach for the attack, since the monetary cost of an attack would increase with the number of queries issued to the classifier. Also, we will assume a black-box setting for the attack since the internal settings of network security products are not usually made public for security reasons. These assumptions will limit the effectiveness of techniques an adversary can apply since many adversarial sample generation algorithms assume a white-box setting. The same techniques can be applied in a black-box setting, but first, the attacker would have to gather enough data to make a local copy of the classifier, approximating its function. The copy can be either costly to make (many queries) or inaccurate in its predictions (too few data points), which can affect the effectiveness of the adversarial sampling techniques.

1.4 Contributions

Here we list this thesis' contributions in order of importance. All mentioned implementations are available online [15].
The contributions are as follows:

- **C1** A Proof of Concept that adversarial sampling techniques are applicable in the domain of device fingerprinting.
- **C2** Method and implementation for transforming device fingerprints into vectors of real numbers, suitable for machine learning setups.
- **C3** Implementation of custom network traffic capturing and fingerprint extraction software.
- **C4** Suggested approach for transforming adversarial fingerprints into network packet sequences.

C5 The adversarial samples we generated were, on average, 9.9% successfully misclassified by the black-box classifier. The most prominent device examined had 36% of its adversarial samples misclassified.

As stated by C1, we aim to shift the domain for traditional adversarial sampling techniques into a new area, specifically from images to network communication. The following two points (C2, C3) are deemed necessary provisions in order to form the evidence. The purpose of C4 is to illustrate how the attack can be made practical for real-world use.

1.5 Thesis outline

We have elected to divide this report into six chapters. After the introduction, in chapter two, we provide some explanations of terms and concepts which are central to our thesis. Then in the third chapter, we describe the attack and the steps we take to enact it. In chapter four, we show the different measurements that we took during each stage of assembling the attack. In the fifth chapter, we discuss the results found and methods used in this thesis. Finally, in the sixth chapter, we present our conclusions and suggestions for future work within the field.

2
Background

This chapter contains the necessary background knowledge to understand the work presented in the report. First, there will be a short overview of machine learning, assuming that the reader does not have any prior knowledge of its mechanics. Since our attack model is a query-limited black-box attack, we present a summary of a paper by Papernot et al. [24] that describes such an attack technique, but for another problem domain, that we base our attack upon. Finally, we explain how device fingerprinting works in technical detail and which metrics are used in the commercial fingerprinter that we attack in our work.

2.1 Machine Learning basics

Machine learning is a powerful tool for computer scientists and can be used to solve a wide variety of problems. One of its strengths, but also one of its weaknesses, is its ability to derive complex relations from large sets of data. The relations between the pixels in a picture and the content of that picture is one such example. This section contains a brief explanation of how machine learning works and how it can be used to generate adversarial samples.

2.1.1 Deep Neural Networks

A deep neural network (DNN) is a term describing a generic mathematical model capable of learning the relationship between an input and an output. It is called a deep neural network due to its architecture, which consists of multiple layers of neurons, hence deep. Each layer of neurons is connected to the next and the first and final layers are connected to the input and output respectively as can be seen in figure 2.1. The input and the output are commonly referred to as the *input vector* and the *output vector* respectively. Note that the vector part of their names refers to a specific instance of a problem, not taking into account the actual dimension of the input or output data itself.

Each neuron can be modeled by a function $f(\vec{x}) = \sigma(\sum_i \vec{w}_i\vec{x}_i + b)$ where $\sigma(\cdot)$ is the *activation function*, b is the bias, and $\vec{w}$ is the weights of the neuron. The activation function is a differentiable function that serves the essential purpose of introducing non-linearity to a neural network. There are several alternatives of activation functions, such as `sigmoid`, `softmax` and `ReLu`, each one with their pros and cons. However, the activation function is an integral part of a neural network since

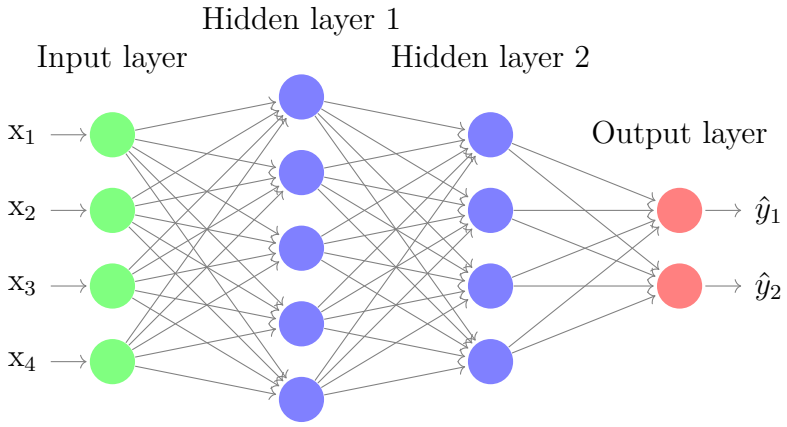

Figure 2.1: Generic DNN

if the function the neural network is trying to learn is a non-linear one, it would be impossible to approximate it accurately by only including linear components to do so. The bias and weights are parameters that are calculated and updated as a part of training the DNN.

Gradient descent is the process of optimizing a DNN's weights and biases. In its most basic form, the process is done by calculating the partial derivative for the total error of the network for each weight and bias individually and updating them by subtracting their partial derivative scaled by a factor, which is known as the learning rate. The intuition is that if the partial derivative for a certain weight or bias is positive, that means that it is adding to the total error and would be better of if it was contributing less, hence subtracting its derivative would likely give a lower total error. The similar case but in reverse holds true but for the case of a negative derivative. This explains the name gradient decent since its visual interpretation is to take steps towards the direction of steepest descent in, for example, a hilly landscape eventually reaching a low point. Note that the total error is dependent on which *loss function* you decide to use for your calculations, and there is a variety of different functions that are good for different problems. For instance, when dealing with classification problems, a widely used loss function is the *cross-entropy* function, which gives a logarithmic penalty for an erroneous prediction. There is a variety of different algorithm which takes the standard optimization process of gradient descent adds their twist to how the process of updating the weights and biases are carried out. For example, there is a well-known optimization algorithm called *Adam* which improves upon gradient descent by speeding up the learning process by looking at the general direction that the updates have taken in prior iterations to decide how to make the next update. Recall the analogy of the hilly landscape, and imagine that the steepest direction for each step forces you to traverse down a winding path, due to the local steepness. Consider now that you could remember that you have traveled in this zig-zag pattern. Then you could make an educated guess that there is a shorter route that you might not have noticed only observing the local steepness. However, by looking at your previous steps, you can make a better guess for your next step, so you do not have to travel from side to side.

2.1.2 Normalization

Normalization of data can be done in various ways. In general, normalization is the act of transforming a dataset such that each feature has roughly the same range of values the other features. One of the simplest ways to normalize data is to scale the initial range of values to a user-specified range. When training a DNN, normalization can speed up the training of the network. The intuition is that when performing backward propagation, more iterations are required when the data is not normalized since a small step in one feature might be a huge step in another. Since all parameters share the same learning rate, going too far in one direction affects all parameters. This potentially leads to spending more steps to reach a cost function minimum instead of taking the shortest path.

2.1.3 Adversarial Samples

An *adversarial sample* is a synthetically generated input vector that only slightly differs from a legitimate input vector, but manages to achieve a class different than the legitimate vector from a classifier. A famous example of an adversarial sample can be seen in figure 2.2, in which an image of a panda was altered slightly by adding some carefully selected perturbations and then classified as a gibbon by a classifier [9]. The image labeled ”gibbon” would still be perceived as a panda by a human, and is therefore misclassified by the classifier.

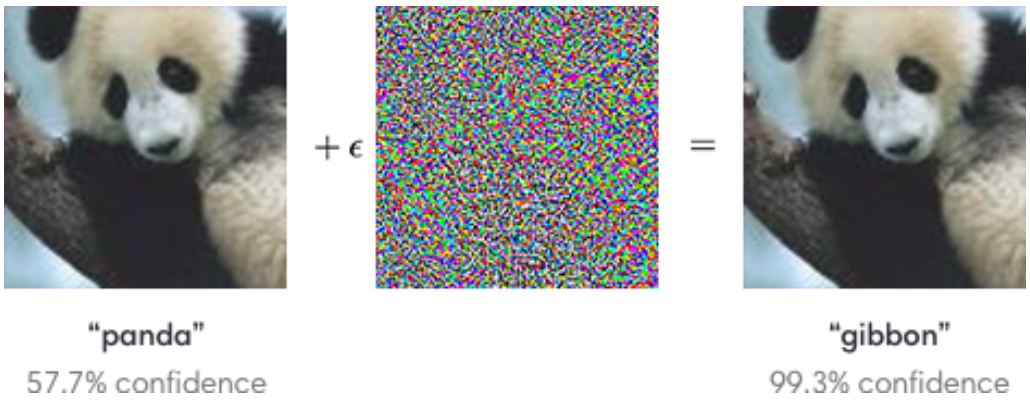

Figure 2.2: Famous adversarial example, where an image of a panda is misclassified as a gibbon after slight modification.

2.2 Attacker Model

We assume an adversary whose goal is to extract sensitive data from a company computer network, possibly for financial gain from either selling to a third party or by blackmailing. The fictional company in question is known to have adopted a policy of Bring Your Own Device (BYOD), having high confidence in their ML-based access control mechanism. Employees are allowed to connect any of their own devices to the network, such as smartphones, laptops, or IoT devices. Each device, once connected to the computer network, are given a class based on some

known properties of the device's generated network traffic. The class, together with other properties (e.g., the role of the user), is used to assign the device a trust-level. However, if the classifier can not assign any other properties than the class, then the class might be the only basis for the trust-level. Our attack seeks to exploit this scenario where the class of a device is the only, or most important, property used to assign the device a trust level. The scenario could arise for example if the device cannot run software normally used to assigning the device a "role to a user".

The adversary is assumed to know how fingerprint data is generated and can send a limited number of queries to the oracle, meaning they can access the labels given to each fingerprint. The limited number of queries is imposed on the attacker since there may be an associated cost to querying the oracle. Thus, an adversary would not find the attack valuable to pursue if it becomes too costly. Additionally, the attacker knows about standard classifications given to specific devices, which means they can have a qualified guess to what device to mimic. However, the attacker does not have any knowledge of how the oracle works.

2.3 Overview of Original Black-Box Attack

Papernot et. al. [24] have provided a successful attack approach for the same scenario this thesis is based on, with the only exception being the operating domain. Their query-limited black-box adversarial attack was originally designed for the domain of image recognition. Here the goal is to deceive an image classifier into wrongfully labelling images with the smallest possible data perturbation. Any human observer should not be able to tell the difference between benign and adversarial images. Additionally, this attack has the added limitation of having a fictional cost added to oracle queries in tandem with hidden internal oracle settings, to make the scenario closer to real world cases.

Attack Overview In the paper [24], the attack is described as a two-stage rocket: first they collect data and train a substitute model, to act as the black-box oracle, then they craft adversarial examples using conventional techniques for white-box scenarios. Below, we summarize each step in chronological order.

1. *Data Collection:* Due to the sophisticated nature of their training algorithms, there's no need for large amounts of data collection which otherwise is required for training a neural network. For instance, 10 images of handwritten numbers (with the numbers 0 through 9) is sufficient as it represents the complete data set.
2. *Architecture Selection:* According to the authors, an educated guess or estimation of number of layers and neurons needed for the substitute network is enough. A high-level understanding of what architectures normally work for a given classification task is stated to be adequate for the attack.
3. *Substitute Training Phase:* Separated in three substeps: Labeling (query the oracle), Training (applying data, label pairs to train the substitute) and Augmentation (artificially create more training data). These substeps are repeated

for a given number of times, each called a *substitute training epoch*.
4. *Adversarial Sample Crafting:* Here the authors generate adversarial samples using two fundamentally similar white-box techniques with the substitute network as target: the *fast gradient sign method* and the *Papernot et al. Algorithm*.
5. *Attack Validation:* By performing the attack against multiple classifiers and with two separate sets of data, the transferability rate of adversarial examples from substitute to black-box network is measured.

2.4 Attack Modifications

The main difference between the attack by Papernot et al. and ours is the change of domain in order to explore new ways to use already established methods. Also, we elect to use a binary classifier, rather than a multi-class classifier, which means that our substitute network will only have two outputs in its last layer: $(X, \neg X)$ (*class* X and NOT class X), which is easy to interpret. This choice is due to two reasons, the goal of the adversary is to reach a specific target class, which they can find adversarial samples for in a shorter time when using the $(X, \neg X)$ premise. Secondly, it provides significantly fewer variations to test and simpler management of the results.

2.5 Device fingerprinting

Device fingerprinting is the act of gathering network data emitted from a device and deriving its most probable identity in terms of hardware, operative system, and network-related software (e.g., a web browser). Typically, the gathered data is information from the different protocol headers in a sent network message.

In this project, the device fingerprinting service Fingerbank API [14, 12] is used as an oracle for fingerprints. The service performs behavioral analysis of the gathered network data and returns a most probable device label and a certainty score. Fingerbank was selected due to its popularity with corporations such as Cisco, Kaspersky, and Nokia [12]. The API uses the following device characteristics to calculate it's label and certainty score: MAC address, DHCPv4 fingerprint, DHCPv6 fingerprint, DHCPv4 vendor, DHCPv6 enterprise number, mDNS services, UPnP server string, UPnP user agent, HTTP user agent, destination domains, TCP SYN signature, TCP SYN-ACK signature, and hostname. In order to understand how this data can be used to identify devices, one needs to understand the overall operation of the underlying protocols. This chapter provides simplified description of each of these protocols and details the specific part used for fingerprinting.

MAC address Fingerprint The Media Access Control (MAC) address is a unique identifier of a Network Interface Controller (NIC), either locally or universally. Hence, a network node with multiple NICs will have one MAC address for each. In fingerprinting, Universally Administered Addresses (UAAs) have the first

three octets reserved as an identifier for the hardware manufacturer of the NIC, for instance, `78:2B:CB:XX:XX:XX` which is reserved for Dell Inc [6]. UAAs thus provide a reliable source for hardware information, in contrast to Locally Administered Addresses (LAAs) that are selected by the network administrator and overwrite the "burned-in" universal hardware address.

DHCPv4 Fingerprint and Vendor Identifier Commonly known as the acronym DHCP, the Dynamic Host Configuration Protocol is an automatic network configuration protocol used to connect Internet Protocol (IP) addresses to MAC addresses of NICs. It was originally created for Internet Protocol version 4 (IPv4) but has since been modified to handle IPv62.5. The protocol typically operates using the User Datagram Protocol (UDP), commonly located in layer 4 of the Open Systems Interconnection (OSI) model, and in 4 separate steps, each consisting of a message sent between a client and a DHCP server:

1. The client sends a `DHCP Discover message` to IP address `255.255.255.255` and destination MAC address `FF:FF:FF:FF:FF:FF` (the broadcast MAC address).
2. DHCP server replies with the `DHCP Offer message` containing a suggested new IP address and other details such as lease time.
3. The client responds by sending a `DHCP Request message`, requesting the offered IP address.
4. Finally, the server confirms the new IP address with a `DHCP Acknowledge message` to the client, terminating the session.

In the initial `DHCP Discover message`, there usually is a field at the very end of the network packet defined as *DHCP Options*, where a limited set of options can be added, defined in Request For Comments (RFC) 2132 [1]. One such option is `option 55: Parameter Request list` which allows the client implementation to specify configuration parameters. This means it is possible to identify a device based on the order these parameters are noted, as different implementations will have different parameters listed. For example: `{1,3,6,15,31,33,43,44,46,47,119,121,249,252}` is the list used by `Microsoft Windows Kernel 10.0` as of the time of writing this report [12]. Similarly, the DHCP client implementation can also add `option 60: Vendor Class Identifier` [1] in the `DHCP Discover message`. This option allows the implementation vendor (manufacturer) to tell the server the name and version of the implementation, which the server has to ignore if it cannot interpret this information. Otherwise, it is allowed to respond with `option 43`. A self-explanatory sample of data added by this option is `android-dhcp-8.0.0` while another less obvious example is `udhcp 1.21.1-LSDK-10.2-00082-4`.

DHCPv6 Fingerprint and Enterprise Identifier DHCP for IPv6, or DHCPv6, is an updated version of the original DHCP to fit the new IP version. Operation is similar to DHCPv4, with four steps and four messages transmitted over UDP between client and server, except for different terminology, packet distribution and, of course, IP addresses. It's specified in RFC 8415 [22] as follows:

1. The client sends a `Solicit message` to broadcast address `ff02::1:2`, UDP destination port `547`.
2. Compatible DHCPv6 server responds with an `Advertise message`, suggesting a new IPv6 address and providing details.
3. The client replies by sending a `Request message` with the chosen address.
4. The session ends by the server replying with a `Reply message` to confirm the new address.

By the same principle as in DHCPv4, there is an option identified as `Option Request Option` with opcode `6`. In contrast to version 4, this option is required to be present in any one of the client messages for version 6. In the project implementation, this information is located in outgoing packets by first identifying IPv6 before passing checks for UDP destination port `547`, which is the standard server-side port for this protocol. Examples of fingerprints include:
`{24,23,17,39} (Windows OS, user agent: Microsoft NCSI)` [12].

The enterprise number/ID is located in `option 16: Vendor Class Option` and `option 17: Vendor-Specific Information Option`, which is an integer mapped to a specific vendor/manufacturer and maintained by the Internet Assigned Numbers Authority (IANA). One such example is `311`, which belongs to Microsoft [12].

mDNS Services Introduced as a stand-alone zero-configuration protocol for small-sized networks, the multicast Domain Name System (mDNS) is designed to act as a replacement in the absence of dedicated DNS servers. It operates exclusively on hostnames having `.local` as their Top-Level Domain (TLD), which most often mean private networks. Instead of, as in conventional DNS, sending a unicast message to a name server, a network node instead issues a multicast request to all other nodes on the network, querying the identity of the node connected to a specific name. The responding node then replies with another multicast message, allowing all nodes on the network to update their mDNS-cache.

`.local` names are set to default values by each device's manufacturer, providing additional information for our case about a device, which works well in tandem with other fingerprinting data. However, because any user can modify `.local` hostnames [3], they cannot by themselves provide enough reliable information about each system. Some examples of such hostnames are `{_googlecast._tcp.local, _companion-link._tcp.local, and Philips hue - 2476FF._hap._tcp.local}`

UPnP Server String and User Agent Universal Plug and Play (UPnP) is a network implementation of the widely used plug-and-play set of protocols. In short, UPnP is intended to allow a wide range of devices seamless and robust initiation of a connection to a network with minimum to no configuration. It utilizes properties of several protocols to this end, including HTTP and multiple others, which presents the ability to identify both UPnP servers and user agents by looking at packet header information added by the underlying protocols.

HTTP User Agent Specified in RFC 1945 [2], the User Agent string of a Hypertext Transmission Protocol (HTTP) request is used for, amongst other purposes, *"automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations"*. It is part of an HTTP request-header field and traditionally provides user/client system information to a web server. This header field takes the shape of a human readable string, which makes the data self-explanatory; the following string from an Asus Android phone/tablet for example
`Mozilla/5.0 (Linux; Android 5.1.1; ASUS_X013D Build/LMY47V; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/75.0.3770.101 Mobile Safari/537.36`

Destination Domains/Hosts This field adds the human-readable name of an endpoint to which the device is sending data, represented as the unresolved part of a DNS record, e.g., `static-mobile.qustodio.com`. It can be the domain to which an HTTP request is sent when a device connects to a website or querying some other data. Fb's API documentation consider destination domains to be behavioral data, and it *"will not be persisted to the Fingerbank database"*. This means that the information is used to identify a device based on its behavior. The data will not be stored in their database, most likely due to the dynamic nature of domain names.

Hostname In the same manner as with Destination Domains, the hostname of the device is defined as behavioral data and does not persist in the Fb database. Instead, this is temporarily used for device behavior analysis as a user typically has full ability to set their hostname, making this data unsuitable for static analysis. Some examples found during this project are `ESP_07AEFB`, `Lightify-7B5F6B`, `Philips-hue`. The information is located in `DHCP option 12: Host Name Option` [1], see DHCPv4.

TCP SYN/SYN-ACK Signature The Transmission Control Protocol (TCP) in layer 4 of the OSI model provides a set of features for reliable computer communication, sacrificing speed for the guarantee of network packets' proper arrival at a specific endpoint. Within the TCP specification, there are header fields with data allowed to be set by the implementation (i.e. the OS), similarly to DCHPv4 and v6. The combination of this data allows one to infer information about a device and its OS, yielding a "signature". The practice commonly referred to as "TCP/IP stack fingerprinting" or sometimes "OS fingerprinting", is the practice of detecting device signatures in TCP messages based on this data. TCP stack fingerprinting is very often used by vulnerability scanners [19] and Intrusion Detection Systems (IDSs). The parameters editable by the OS include the following:

- Initial packet size
- Initial TTL
- Window size
- Max segment size
- Window scaling value
- "don't fragment" flag
- "sackOK" flag

- ”nop” flag

Each of these parameters can be found in both the TCP SYN and SYN-ACK packets and are normally a reliable source for OS information since there is no protection against this type of fingerprinting is in place. In order to send this data to Fb for labeling, it has to be formatted to according to the `p0f`-tool [27] standard format, see the following examples:

- Philips IoT: `4:64+0:0:1460:29200,3:mss,sok,ts,nop,ws:df,id+:+`
- Windows OS: `4:128+0:0:1460:8192,:mss,nop,nop,sok:df,id+:+`
- D-Link IP Camera: `4:64+0:0:1460:5792,1:mss,sok,ts,nop,ws:df:+`

In addition, these signatures are considered to be behavioural data and are used by Fb to analyze a device dynamically in the same way as with Hostnames and Destination Domains.

3
Method

In this chapter, we describe how our attack against the black-box network was carried out. The first section details the initial investigation of the service and a summary of how the attack detailed in the paper by Papernot and Goodfellow [24] could be modified to suit our problem. Then, the following four sections go into more detail of what modifications were done to tailor the attack to our scenario. We dedicate the last section to describe how we evaluate the performance of the attack and the measurements we took along the way in order to find our final adversarial network.

3.1 Attack Overview

The attack is divided into four separate steps, based on the description by Papernot et al. [24]. This approach was chosen because the premises on which Papernot et al. designed the attack completely matches this project's setting, apart from the domain and size of the problem. Originally, the process was designed to attack the integrity of classifiers of the MNIST dataset containing images with numbers. Each image is 27 by 27 pixels which yield 729 parameters to perturb, in contrast to this project's 253 total input parameters operating over converted network information instead of image data. The following are the basic steps, or stages, of the attack.

1. Data Collection and Processing
2. Substitute Network Training
3. Generation of Adversarial Examples and Attack on Substitute
4. Test of Transferability Rate to Black-box Model

In this chapter, we describe each step in detail with emphasis on the domain conversion necessary to fit the attack to its new setting.

3.2 Data Collection

According to the training algorithm by Papernot et al., the initial data only need to be representative of the whole data set used and thus no great quantities are initially required. The amount of training data is artificially doubled with each iteration using what is called *Jacobian-based Data Augmentation*, which we explain in more detail in section 3.4. As stated in the paper [24], this method effectively generates new data, given that the provided original data is representative of the whole domain of each parameter. In order to obtain the network data and extract

device fingerprints, we designed a custom packet sniffer and parser software which we implemented, along with all other code within the project, in Python 3.6.

Figure 3.1 describes how our data collection is carried out. Each passing packet is copied from the network and broken down layer by layer following the OSI model. We can then identify all relevant protocols in each layer and send their information to the corresponding content parser, which in turn extracts and formats the fingerprint data. Once enough information is collected, we pass it to our labeling logic, which queries the Fingerbank API [14], and stores the resulting device ID label and fingerprint pair in a file which we use to train our substitute network.

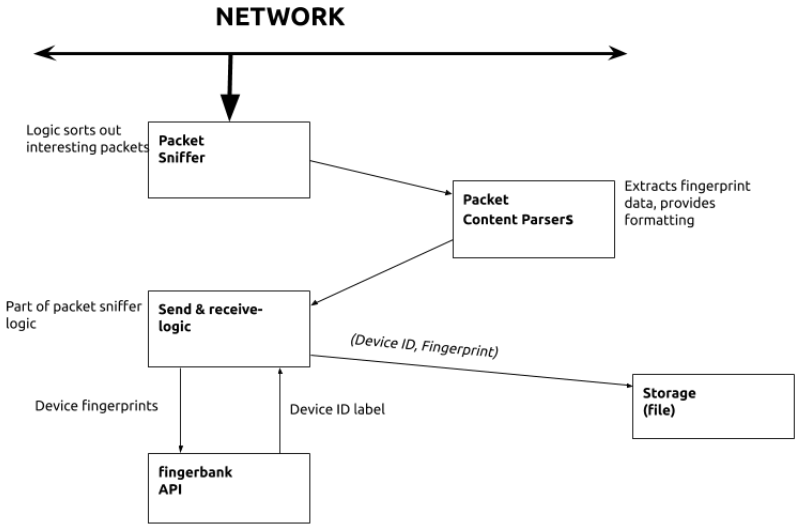

Figure 3.1: Diagram representing the data collection process. Our custom-made packet sniffing software connects to the chosen computer network and sorts out selected packets. This content is parsed and labelled by the Fingerbank API service before being sent to storage.

3.3 Data Processing

For the training the substitute network we use TensorFlow [26] and Keras [16]. In order for us to use the collected data with these libraries, we needed to convert it into arrays of numerical values. Additionally, we normalized our data to prevent individual features from gaining a disproportionate range of expression compared to the other features, which would slow down the rate of learning. However, the conversion into numerical values is not trivial, as the fingerprint data varies widely. Some of the parameters, such as HTTP User-Agent and Destination Hosts, are pure text strings while others, such as MAC Address and DHCP Fingerprint, are, close to, numbers or lists of numbers. This means that a data processing function needs to apply one of two conversions, depending on whether the input is a text string or a number. The processing is illustrated in Figure 3.2.

Additionally, each of the described converting functions has a corresponding inverting function. This is necessary as the results of the attack are only meaningful if the generated adversarial examples can be translated into actual device fingerprints. These functions work by performing the conversion in reversed order, i.e., reverse look-ups in text string dictionaries and multiplication of numerical max value for numbers.

3.3.1 Text String Conversion

In the case of text strings, the conversion to numbers was made by performing a separate initial iteration over all collected data. In this iteration, all strings are collected and mapped via dictionaries to a value in the range $[0.0, 1.0]$, evenly distributed among all occurrences. Alternatively, one could do letter by letter conversion of each string to its Unicode counterpart, which would provide the substitute with more freedom by adding more parameters. However, this approach immensely increases the complexity of the problem, as most string generated by the substitute would not be humanly readable. We therefore elected to proceed with the more simple approach that used valid values found in data collection.

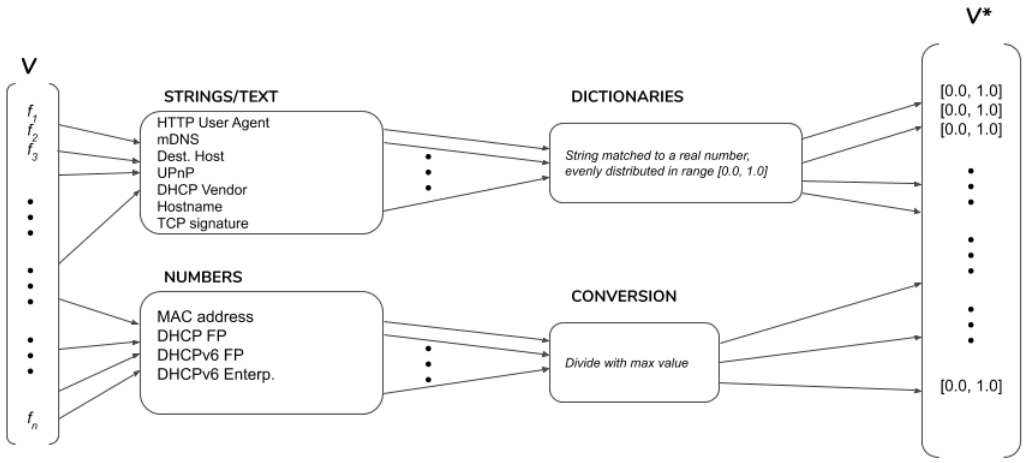

Figure 3.2: Data processing diagram. The left vector **V** holds a varying number of network features while the right vector **V*** is normalized to 253 elements, each in the range $[0.0, 1.0]$.

3.3.2 Number Conversion

In contrast to converting the text string data, the numbers are more easily modeled to fit the code libraries. It is only a matter of dividing each number with the defined max value for the property in order to reach the desired range $[0.0, 1.0]$ leaving the 0 as an empty spot. Thus, once the data has been obtained, it merely becomes a matter of iterating over the list, performing the conversion, and appending to the final array while maintaining the order. Figure 3.2 illustrates the conversion. For

example, an instance of a DHCP fingerprint may be

$$1, 121, 3, 6, 15, 119, 252, 95, 44, 46$$

which in this case is from an Apple implementation. After conversion, the data is instead a normalized array of real numbers:

$$0.1, 0.121, 0.001, 0.1, 0.15, 1.0, 0.8, 0.95, 0.00044, 0.0023$$

3.4 Substitute Network

To limit the work, we elected to construct the substitute network as a binary classifier simplifying the problem by only having it discern between a specific class and everything else. Based on that and the feature representation described in section 3.3, it seemed suitable to use an n-layered DNN. Papernot et al. [24] mention that the specific number of layers and neurons have a relatively small impact on the success of the attack. Thus, in this report, we primarily examined one network with four layers and 128 neurons in each layer. The neurons were fully connected and used sigmoid or softmax as their activation functions. Additionally, binary cross-entropy was used as the loss function and Adam as the optimizer during the training of the network.

To train the substitute network, we followed the method described in the paper by Papernot et al. [24]. The method utilizes Jacobian-based data augmentation, generating new training samples in each substitute epoch. For each existing training sample, a new and slightly modified sample is generated and then labeled by querying the oracle. The slight modification is derived from the Jacobian of the substitute network, with regards to the specific dimension that is related to the samples output label. After generating these new samples, they are added to the training dataset before the next substitute epoch of training begins. Each substitute epoch consists of ten training rounds where the network is trained from scratch.

For training our network we elected to train for six substitute epochs after initially testing to train for ten epochs without seeing any significant improvement to the substitute network past six epochs. Our training data was based on ten samples that portrayed apple devices and ten devices portraying non-apple devices. We based the number of samples based on a similar choice made in [24], where they used ten samples per category to train their substitute network with (the handcrafted set). These choices also made the training of the substitute network somewhat quick, approximately 5h in total, comparing to training the network for ten epochs which took more than 34h. The majority of the time for training the network was consumed by waiting for Fingerbank API, which is rate limited to 300 requests per hour.

3.5 Adversarial Sampling

For adversarial sample crafting, we decided to use the Jacobian-based Saliency Map method (JSMA) detailed in a paper by Papernot [25]. The reason for electing to

use this method over the fast gradient sign method (FGSM), both mentioned in the paper by Goodfellow and Papernot [24], was that JSMA was described as suitable for source-target misclassification and that JSMA added smaller perturbations compared to FGSM. In our implementation of the adversarial sample crafting, we used the code of JSMA provided in the CleverHans library and based our code on a tutorial provided in the same repository [23]. The JSMA implementation takes as input the gradients of the substitute network, a benign fingerprint, a target class, a maximum percentage of perturbed features (γ), and the size of those perturbations (θ). The method utilizes the gradients of the substitute network in order to calculate what features of the benign fingerprint should be altered in order to get the fingerprint to be interpreted as the target class. The adversarial fingerprint is created with this information and is then checked against the substitute network and the black-box network to validate the successfulness of the adversarial sample.

To test the fingerprint against the black-box network it first has to be transformed from its vector encoding to the fingerprint form, which is described in section 3.3. After producing the adversarial samples, we stored them in a text file for later analysis. To calculate the successfulness, we wrote a script to go through the text files and summarize how many of the fingerprints were successful against the substitute and the black-box, respectively. This script, complete with all other implementation, can be found online [15]. Additionally, we measure whether each adversarial sample works on: both networks, only the substitute, only the black-box, or none of the networks. Full summary of this outcome matrix and its layout can be found in Table A.1.

In order to find what combination of parameters yielded adversarial samples, we ran tests for all combinations of γ and θ where they both took on the values $[0.1, 0.2, 0.3, 0.4, 0.5]$ respectively. As we found an effective combination of γ and θ, we use these values in our algorithm configuration to produce 100 adversarial samples for each of ten different device types. Those samples were all based on ten different fingerprints for each device type.

3.6 Evaluation

In practice an adversarial attack only needs to find one adversarial example to be successful. However, for the sake of evaluating the performance of the attack while also checking consistency, several hundreds of adversarial examples are generated for an array of devices using a constant amount of fingerprint samples. We consider an adversarial example to be successful when it manages to change any devices' perceived class into a specific target class. Thus, the aforementioned procedure's performance is measured using the following metrics.

- Success rate on substitute
  - The ratio of samples that are successfully interpreted as the target label by the substitute
- Success rate on the black-box network (Fingerbank)

  - The ratio of samples that are successfully interpreted as the target label by the black-box
- Sample connectivity
  - The samples' success ratio against: no network, substitute network only, black-box network only or both networks
- Accuracy of substitute network during each substitute epoch
  - The ratio of test samples that are correctly labelled by the substitute, input at the end of each substitute epoch

4
Results

In this chapter, the findings and results of the work are presented. First, a description of the data used to produce said results, followed by the results of the substitute network training. Then, the parameters used during the adversarial sampling phase are shown before the findings of that phase presented along with one of the successful adversarial samples. Finally, we add a summary of all adversarial samples.

4.1 Description of data

The data used in this work was collected from two sources: live data and previously recorded, publicly available Packet CAPtures (PCAPs) [21]. The live data was captured from the network at Cyxtera's office using the custom data collector described in section 3.2. The PCAPs were replayed onto the network and then captured in the same manner as the live data. From these two sources a wide variety of IoT devices and regular office devices are represented. The data from the office consists of desktops, printers, laptops, and other handheld devices of various kinds and has been sampled at multiple instances. In the PCAPs a broad selection IoT devices are found, they have all been sampled once due to the data's static nature. The collection contains 93 unique fingerprints with 14 different device classes, assigned by fingerbank. A summary of the captured devices can be found in Appendix A.2.

4.2 Training the substitute network

In figure 4.1, the accuracy of the substitute network is plotted. Each point describes the accuracy reached after ten rounds of training on the same data. Between each point, more data is augmented and added to the training set before retraining the network from scratch to reach the next point in the plot. We can see that the network's accuracy increases for each round of data augmentation and training. Note: accuracy in this plot refers to the likelihood for our substitute to label a data-point correctly, according to fingerbank.

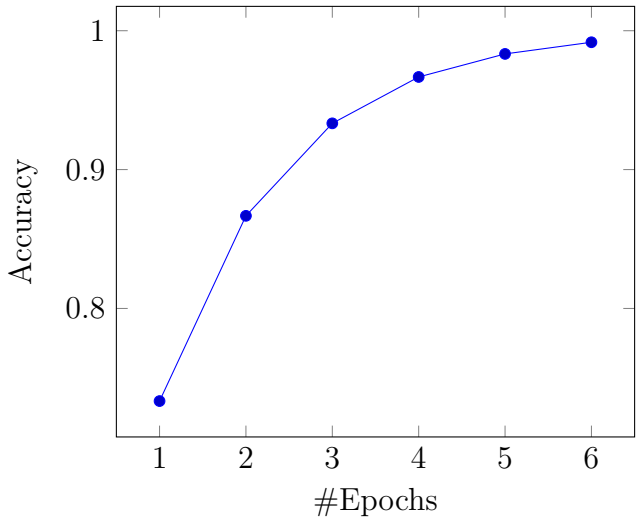

Figure 4.1: Our substitute network's accuracy after each substitute epoch, consisting of data augmentation and network training.

We observe that our network reached a similarly high accuracy to the network trained by Goodfellow and Papernot [24], ours reaching a peak value of 99.17%, compared to their 81.20% by Papernot et al. which they trained against the MNIST data set.

Figure 4.2 display in a similar manner as figure 4.1 the substitute network's loss after each round of augmentation and training. The loss was initially at 0.6457, and we attained 0.506 in the final epoch.

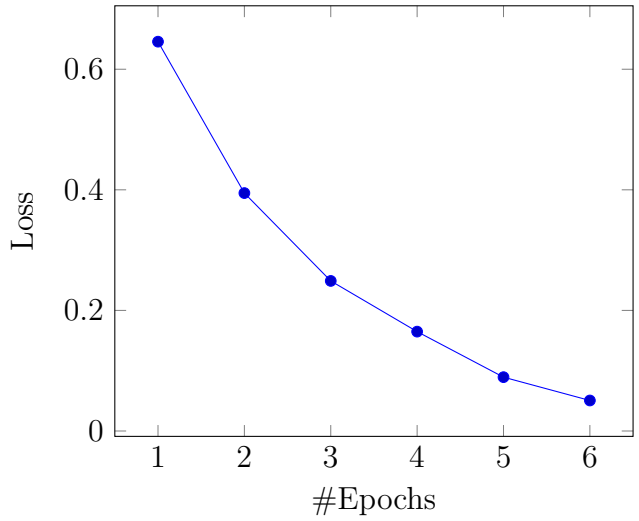

Figure 4.2: The substitute network's loss after each substitute epoch, consisting of data augmentation and network training.

4.3 Parameter Search

The search was made with a selection of 10 different benign source samples, in order to find the algorithm configuration that yields the highest number of successful adversarial samples. Once identified, these values for γ and θ were used as input to the function generating the adversarial samples. θ denotes the size of perturbation introduced on each feature while γ denotes the maximum percentage of perturbed features. In figure 4.3 one can see that the parameter combination of $\theta = 0.1$ and $\gamma = 0.3$ is the only one that successfully produces adversarial samples.

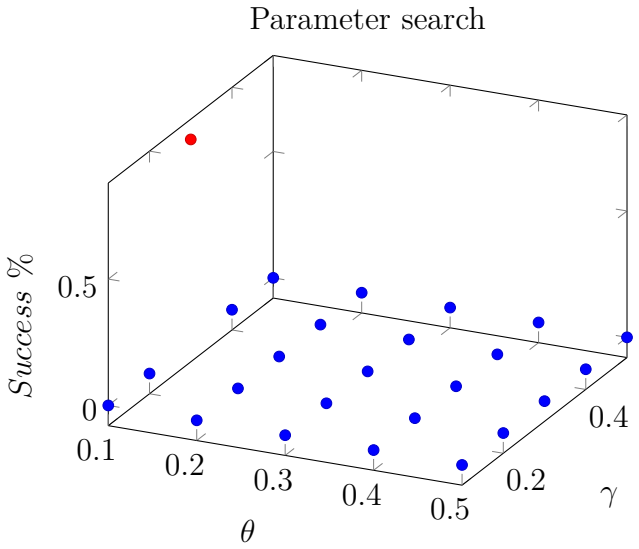

Figure 4.3: Parameter search to find the most successful combination of γ and θ, representing the algorithm configuration for the production of adversarial samples. The red point marks the only configuration with which adversarial samples had any success.

4.4 Adversarial samples

In figure 4.4 the number of adversarial samples that were successful against Fingerbank are shown. The numbers were taken from a sample of 100 adversarial samples per device. The substitute network used was trained for six substitute epochs.

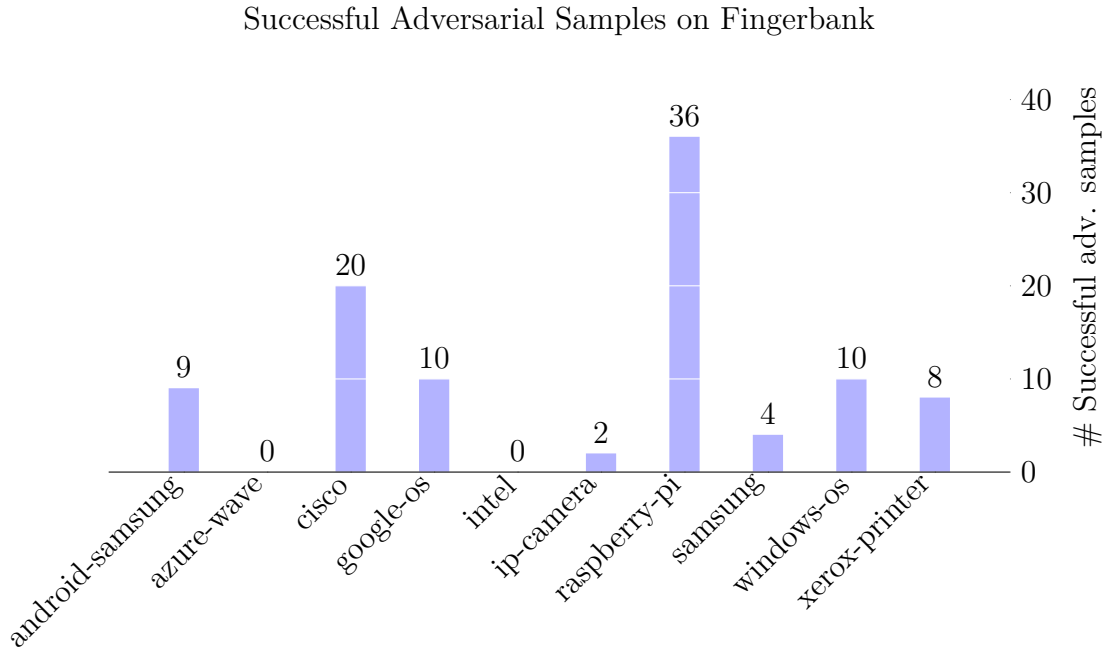

Figure 4.4: Number of adversarial samples out of a 100 which successfully were interpreted as the target device per original device category.

In figure 4.5, looking at one of those adversarial samples and its benign fingerprint counterpart, the changes made to obtain the adversarial sample can be seen. The benign fingerprint shown at the top of the figure belongs to a Raspberry Pi device and is classified correctly as a Raspberry Pi by Fingerbank. The adversarial sample at the bottom of figure 4.5 is based on the benign fingerprint, and Fingerbank classifies it as an Apple device. The sample introduces three new fingerprint characteristics: user agents, TCP SYN/ACK signature, and hostname, and modifies the original three characteristics: mac, destination hosts, and TCP SYN signatures.

```
Benign fingerprint
mac: b8:27:eb:6b:e8:78
destination_hosts: [clients4.google.com]
tcp_syn_signatures: 4:64+0:0:1460:29200,7:mss,sok,ts,nop,ws:df,id+:+

Adversarial sample
mac: b8:27:ea:ff:ff:ff
destination_hosts: [www.svtstatic.se, track.adform.net]
tcp_syn_signatures: 4:90+0:0:1460:29200,7:
    ws,sack,?n,mss,ws,eol+n,eol+n,eol+n,eol+n,eol+n:df,id+:+
user_agents: [
    Mozilla/5.0 (X11; Linux armv7l) AppleWebKit/537.36 (KHTML, like Gecko)
        Raspbian Chromium/72.0.3626.121 Chrome/72.0.3626.121 Safari/537.36,
    Mozilla/5.0 (X11; Linux armv7l) AppleWebKit/537.36 (KHTML, like Gecko)
        Raspbian Chromium/72.0.3626.121 Chrome/72.0.3626.121 Safari/537.36,
    Mozilla/5.0 (X11; Linux armv7l) AppleWebKit/537.36 (KHTML, like Gecko)
        Raspbian Chromium/72.0.3626.121 Chrome/72.0.3626.121 Safari/537.36]
tcp_syn_ack_signatures:
   4:0+0:1:0:0,:eol+n,eol+n,eol+n,eol+n,eol+n,eol+n,eol+n,eol+n,eol+n,eol+n:0
hostname: PerSvennsiPhone
```

Figure 4.5: Benign fingerprint that is classified as a Raspberry Pi followed by an adversarial sample based on the same Raspberry Pi generated from our software. This gets classified as an Apple device by the Fingerbank service.

Notice how the user agents and the TCP signature code *eol+n* are repeated in the adversarial fingerprint in figure 4.5. This is abnormal behavior for a fingerprint as *eol+n* should only be present once at maximum, and repeated user agents do not add anything. Despite this, it is still recognized by fingerbank as a valid fingerprint.

4.5 Outcome matrix

No adversarial sample managed to deceive both the substitute network and the black-box network simultaneously, as can be seen in table 4.1. However, the majority of the adversarial samples created were successfully misclassified by the substitute network and multiple adversarial samples managed to fool the black-box network.

Source Device	None	Substitute	Black-Box	Both
Android-Samsung	41	50	9	0
Azure-wave	50	50	0	0
Cisco	40	40	20	0
Google-OS	50	40	10	0
Intel	20	80	0	0
IP-camera	58	40	2	0
Raspberry Pi	24	40	36	0
Samsung	6	90	4	0
Windows-OS	30	60	10	0
Xerox Printer	32	60	8	0
Total Average	35,1	55	9.9	0

Table 4.1: A side-by-side comparison of the success rates for adversarial samples out of 100 attempts, indicating the amount that deceived either none, substitute only, black-box only or both for each source device. The target device in all cases is Apple iOS, thus deceive refers to "being classified as an Apple iOS device".

5
Discussion

This chapter provides our reasoning and discussion around the results in Chapter 4. Initially, the adversarial samples generated are discussed along with their obviousness as well as how they might be reduced. This is followed by some ideas on how the adversarial fingerprint might be turned back into network packets in order to increase the practicality of the attack. Finally, a summary of the attack success and mitigation is provided. Weak points of the study are mentioned under each subsection and discussed further in the next chapter.

5.1 Adversarial Samples

Table 4.1 shows that no adversarial samples manage to deceive both our substitute network and Fingerbank. One of the potential causes of this result is the training of the substitute network. The majority of all adversarial examples appear in the "Substitute" column, indicating that the substitute is easier to fool than the black box. In turn, that points to insufficient or ineffective training of the substitute, since it does not accurately represent the black box. The most likely cause of this lacks quantity or quality of training data. Another potential cause for the reduced transfer rate could be that the substitute's architecture might not be suitable for this problem. However, according to [24], architectural details will not impact the result significantly. Since the potentially insufficient substitute network is the basis for all adversarial sample generation, this could also explain why there are substantially fewer examples that manage to attack Fingerbank. In conclusion, while the substitute does not represent Fingerbank satisfactorily, the JSMA method still manages to generate some useful adversarial examples, giving validity to hypothesis **H1** of transferability to a new domain. It would be interesting to re-run the experiment but with better, higher quantity and quality, training data, and see if the transfer rate increases.

When considering the adversarial samples' appearance compared to their original counterparts, there is a set of obvious changes. The change of MAC address is most noticeable. It is a drastic update in device behavior as this is the lowest-level identifier of any device and, in most cases, "burned-in" to the hardware. Other changes include the repetition of entries that otherwise only appear once or very few times per fingerprint, such as the HTTP user agent string or the `eol+n` option in a TCP SYN/ACK signature, as in Figure 4.5. Together these new features form adversarial samples that, both with manual and automated observation, directly should raise

suspicion. Behavioral Intrusion Detection Systems (IDSs) should have no problems detecting significant changes.

5.2 Reduce Adversarial Sample

Many of the successful adversarial samples look strange to an observer with the knowledge of how fingerprints are created since there are many repetitions of different features, which generally only shows up only once, e.g., user agents. For instance, the adversarial sample in figure 4.5 has repetitions of its user agents and of the code `eol+n`, which would not occur in a properly created fingerprint. Thus it should probably not matter to the black box classifier if there are one or multiple instances of them in the fingerprint. Testing this theory reveals that it held true and that the fingerprint in figure 5.1 was also classified as an Apple device. Another notable feature in figure 5.1 is the hostname, as it contains the name of

```
mac: b8:27:ea:ff:ff:ff
destination_hosts: [www.svtstatic.se, track.adform.net]
tcp_syn_signatures: 4:90+0:0:1460:29200,7:ws,sack,?n,mss,ws,eol+n:df,id+:+
user_agents: [
    Mozilla/5.0 (X11; Linux armv7l) AppleWebKit/537.36 (KHTML, like Gecko)
        Raspbian Chromium/72.0.3626.121 Chrome/72.0.3626.121 Safari/537.36]
tcp_syn_ack_signatures: 4:0+0:1:0:0,:eol+n:0
hostname: PerSvennsiPhone
```

Figure 5.1: Adversarial fingerprint, reduced to look less suspicious.

a device (iPhone) designed by the target label Apple. This made for some interesting observations upon further investigation. For the purpose of classification the hostname `"PerSvennsiPhone"` should not need the personalized `"PerSvenns"` part. Removing it and re-sending the fingerprint to the oracle once more, we found that the accuracy had increased from 30% certainty in it being an apple device to 62% certainty. This exciting discovery led us to perform additional experimenting and eventually, the uncovering that any suffix to "iPhone" was ignored, but any prefix lowered the confidence. Going back to the original benign sample and just adding the hostname `"iPhone"`, we found it was by itself sufficient to create an adversarial sample. We applied this test on some other benign samples and were able to change the class of them as well. Additionally, this discovery indicates that there ought to be many more such universal adversarial feature values that could be used to change class into other device types. However, the feature representation that we decided on in 3.3 does not allow for this degree of freedom, selecting any possible combination of letters for the hostname as an example.

5.3 Adversarial Sample To Network Packet Sequence

In our report, we show how one can generate adversarial samples or, in other words, adversarial fingerprints. However, to realize an attack, an adversary would have to go one step further and turn those fingerprints into network packets a device can emit to achieve misclassification. In this section, we will discuss our ideas on how to transform a sample and details to consider.

Firstly, recall that we consider a fingerprint to be an aggregate of several network packets header data where the information presented in the fingerprint is the most recently available information of that kind. We illustrate this idea in 5.2, where P_1, P_2, P_3 form a sequence of packets emitted from a device and F is their corresponding fingerprint. Notice how F contains the latest (rightmost) available information for each kind of fingerprint data, A_3, B_1, C_2. It is also important to point out that in figure 5.2, we do not say anything about the value of, e.g., A_3. A_3 might contain the same value as A_1 and A_2, or not; we only know that they contain information of the same kind, e.g., MAC addresses. However, from the perspective of the function that aggregates the fingerprint, only the last available information is considered, and therefore, we will also do the same.

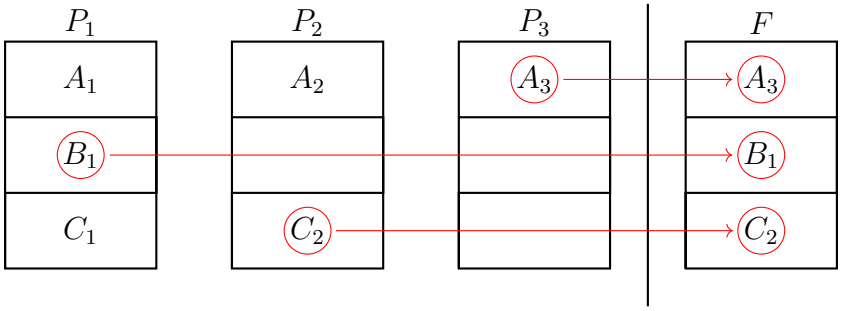

Figure 5.2: A simple example of how a network packet sequence (P_1, P_2, P_3) is transformed into a fingerprint (F). The fingerprint contains the latest available information for each type of fingerprint data.

Let us consider an adversarial sample F_x for the fingerprint F in figure 5.2. In figure 5.3 we can see that F_x is a modified version of F, where the difference is that the B feature is now the adversaries chosen value B_x. To introduce the value B_x to the fingerprint F_x, the adversary must insert their own packet P_x into the network packet sequence at a point after packet P_1 to override B_1. One such placement could be as shown in figure 5.3, but there are more valid placements to consider depending on the details of the fingerprint. It is also crucial for the adversary to determine what value, if any, to assign to A_x and C_x. In the remainder of this section, we will detail some intricacies of converting adversarial samples into network packet sequences that an adversary would need to find solutions to.

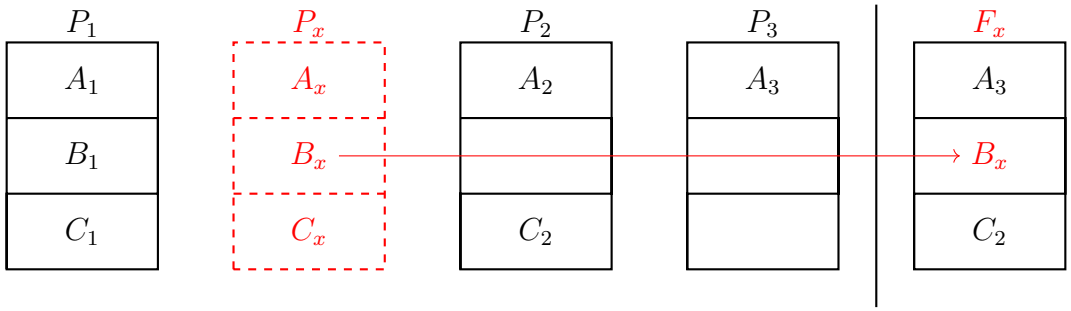

Figure 5.3: A simple example of how an adversary can introduce their adversarial fingerprint's feature into the original network packet sequence. B_x is the only new feature introduced. So the adversary can place his packet anywhere in the sequence, as long as he overrides B_1 from packet P_1.

First of all, the adversarial packet P_x must decide what data it should include in A_x and C_x. Some types of fingerprint data, e.g., the source MAC address, are mandatory for every packet to have, while others might be left out. There might also be interdependencies that the adversary has to account for, like some protocols higher up in the OSI model will only ever be carried over TCP or UDP, respectively.

Secondly, an adversary might generate an adversarial fingerprint that can not fit into a single network packet. E.g., the adversary needs to alter both UPnP data and DHCP data, which both operate on the application layer.

Thirdly, the adversary also needs to ensure that he can reliably achieve the packet sequence that yields his adversarial sample. For example, suppose there are tiny windows of time between each of the network packets in the original packet sequence. In that case, it might be difficult for the attacker to insert the adversarial packet in a specific position in the sequence. He might need to ensure that it would be okay for that packet to arrive one or more packets before or after the target position in the sequence.

To summarize, the attacker will need to find a combination of network packets that manage to both successfully overwrite the original values for each fingerprint data type the adversary wish to override and, at the same time, adhere to the rules for how network packets are composed. Although we do not think this is a simple task, we believe that a well-motivated attacker could automate the process of converting adversarial fingerprints into sequences of network packets.

5.4 Practicality of Our Attack

The attack itself can potentially be packaged in a black-box fashion, to which the adversary inputs network data and target class before receiving adversarial network traffic as raw byte data, ready to be sent over the network. However, given the large differences in device hardware, it is up to the adversary to provide programming that actually sends this information from the device over the network.

One possible solution is to provide the attack as a two-stage rocket: the first stage as an independent package to generate the adversarial network traffic and the second stage to deploy it, with several versions depending on hardware. If this works well as implementation, then many details can be abstracted away, and the only real bottleneck for the attack is the time to label network data for substitute training. Of course, the amount of labeling time required varies depending on limitations set up by the black-box network, as in the case of this project, where a query only could be sent to Fingerbank every 13 seconds on average. This limitation slowed training progress down to 1-2 full work days for several thousand queries. In conclusion, the attack's practicality depends, in essence, on the black-box network implementation as well as the resources and dedication of the adversary.

5.5 Attack Success and Mitigation

Based on the attack model and the results from our experiments, we can confidently state that the attack is viable, given some specific circumstances. Namely, the security mechanism relies heavily on the result of the fingerprint classifier and employ little to no additional inspection of the network traffic for irregular behavior. Additionally, it is most likely that no manual checking of network traffic logs would take place, and thus the attack could go unnoticed.

Regarding the repeatability of the attack, we noticed that once the attacker obtained an adversarial sample, it could be reused without changing the classification from Fingerbank. An example of this is in section 5.2, where altercation of the benign fingerprint to contain a specific hostname string changed the original device's perceived class. The static classification behavior of Fingerbank implies that a found adversarial sample could be mass distributed via a virus, which would make such an attack even easier to perform successfully.

What might hinder the attack is, as previously stated, advanced behavior analysis. A truly behavioral IDSs or IPSs can respond in one of two ways depending on the implementation: detect this sudden change in the device's behavior and take action accordingly, or assume a new unit and that the old device just stopped communicating. However, since this attack procedure aims not to hinder device function as not to expose the attack, adversarial packets' timing is vital. We deem that the attack has the best chance of success when the adversarial packets always are sent before any other traffic is commenced, i.e., directly upon network connection. This should be enough for misclassification, provided that classification occurs at the same instant as any new unit is successfully connected. Interlacing malicious packets with regular traffic are, for the behavioral IDS, a more noticeable change in behavior.

6
Conclusion

In this thesis, we confirmed our main hypothesis **H1** by choosing methods proven effective in the domain of image classification, adding the processing and converting techniques described in section 3.3 and applying new data in the form of device fingerprints, outlined in section 2.5. Showing that, on average, 9.9% (Table 4.1) of the generated adversarial samples were successfully misclassified by the black-box classifier positively answers **H2**. The attack is still practical for an adversary, both giving them the choice of attacking device and target fingerprint. In our study, the attacker could utilize eight out of the ten devices for their attack (Figure 4.4). In total, 36% of the adversarial samples were misclassified as the targeted fingerprint. However, we found that the techniques were less successful in our domain (average 9.9%) than in image recognition (84.24%) [24]. Furthermore, we suggest an approach for converting adversarial fingerprints into network packet sequences. Being a theoretical and non-proven method, it does not satisfactory supply an affirmative answer to **H3**. This does not, however, nullify important insights generated by our work, but instead creates a starting point for future works. We found that the adversarial samples created were abnormal in their composition, with repeating information. With sufficient knowledge and low effort, an attacker can remove the repetitions, and the class given by the classifier was shown to be the same. Although the adversarial samples were successful, a human observer with knowledge of device fingerprints can point out that something has changed, but not tell whether a fingerprint is adversarial or benign. Knowing that adversarial fingerprints can be created, users of device fingerprinting must be aware of the inherent weakness of the method and account for the risk of misclassification when using it.

6.1 Future Work

As discussed in section 5.4, there is still work to be done to make the attack more practical by packaging it in a black-box fashion. This project only showed that the adversarial attack is possible to perform in a new domain. We believe the results can be additionally refined, with emphasis on tweaking the adversarial samples to be less obvious for a human observer, which would make the attack even less likely to be discovered.

In this thesis, we only explored the JSMA method but would have liked to compare it to the FGSM method. It would be interesting to see in a future work a comparison of the two methods to see which one works the best for crafting adversarial fingerprints.

Another interesting dimension to explore is the data representation of the fingerprint. The current representation described in section 3.3 does not allow for exploration; only re-using previously observed values. As mentioned in section 5.2, we accidentally discovered that the hostname "iPhone" was a powerful adversarial feature. However, our data representation did not allow the adversarial sample crafting to find this value. It is an excellent suggestion for future research and would shed light on the weaknesses of the specific black-box classifier we attacked.

In order to make the attack more practical, an automated conversion from adversarial fingerprint to network packet sequence would need to be implemented. We suggest that the implementation should be based on the approach outlined in section 5.3.

Bibliography

[1] S. Alexander and R. Droms. RFC 2132: DHCP Options and BOOTP Vendor Extensions - Internet Engineering Task Force. (accessed 2019-06-12) https://tools.ietf.org/html/rfc2132.

[2] T. Berners-Lee, MIT/LSC, R. Fielding, UC Irvine, and H. Frystyk. RFC 1945: Hypertext Transfer Protocol – HTTP/1.0 - Internet Engineering Task Force. (accessed 2019-07-01) https://tools.ietf.org/html/rfc1945.

[3] S. Cheshire, M. Krochmal, and Apple Inc. RFC 6762: Multicast DNS - Internet Engineering Task Force. (accessed 2019-06-13) https://tools.ietf.org/html/rfc6762.

[4] Cisco. VNI Complete Forecast Highlights. (accessed 2019-08-08) https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_Device_Growth_Traffic_Profiles.pdf.

[5] Cisco. Cisco 2018 Annual Cybersecurity Report. Technical report, Cisco, 2018.

[6] Gerald Combs. Wireshark Network traffic analyzer - /etc/manuf - Ethernet vendor codes, and well-known MAC addresses. (accessed 2019-06-12) https://code.wireshark.org/review/gitweb?p=wireshark.git;a=blob_plain;f=manuf;hb=HEAD.

[7] Inc. Cyxtera Technologies. Secure Infrastructure - Colocation - Cybersecurity | Cyxtera. (accessed 2019-09-16) https://www.cyxtera.com/.

[8] Christian J D'Orazio, Kim-Kwang Raymond Choo, and Laurence T Yang. Data exfiltration from Internet of Things devices: iOS devices as case studies. *IEEE Internet of Things Journal*, 4(2):524–535, 2017.

[9] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv preprint arXiv:1412.6572*, 2014.

[10] Goodfellow, Ian J and Shlens, Jonathon and Szegedy, Christian. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[11] Weiwei Hu and Ying Tan. Generating Adversarial Malware Examples for Black-box Attacks based on GAN. *arXiv preprint arXiv:1702.05983*, 2017.

[12] inverse. Fingerbank. (accessed 2019-02-18) https://fingerbank.org/.

[13] inverse. Packetfence. (accessed 2019-02-18) https://packetfence.org/.

[14] inverse inc. Fingerbank api. (accessed 2019-04-30) https://api.fingerbank.org/api_doc/2.html.

[15] Gustav Örtenberg Joel Andersson. Code implementation. https://gitlab.com/gustav.ortenberg/adversarial-attacks-on-device-fingerprints.

[16] Keras. Keras. (accessed: 2019-02-27) https://keras.io/.

[17] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[18] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial Machine Learning at Scale. *arXiv preprint arXiv:1611.01236*, 2016.

[19] Gordon Lyon. OS Detection | Nmap Network Scanning. (accessed 2019-07-01) `https://nmap.org/book/man-os-detection.html`.

[20] Yair Meidan, Michael Bohadana, Asaf Shabtai, Martin Ochoa, Nils Ole Tippenhauer, Juan Davis Guarnizo, and Yuval Elovici. Detection of Unauthorized IoT Devices Using Machine Learning Techniques. *arXiv preprint arXiv:1709.04647*, 2017.

[21] 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 *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 2177–2184. IEEE, 2017.

[22] T. Mrugalski, M. Siodelski, B. Volz, A. Yourtchenko, M. Richardson, S. Jiang, T. Lemon, T. Winters, ISC, Cisco, Huawei, and UNH-IOL. RFC 8415: Dynamic Host Configuration Protocol for IPv6 (DHCPv6) - Internet Engineering Task Force. (accessed 2019-06-12) `https://tools.ietf.org/html/rfc8415`.

[23] et.al Papernot, Goodfellow. CleverHans library. (accessed 2019-03-15) `https://github.com/tensorflow/cleverhans`.

[24] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Machine Learning. *arXiv preprint arXiv:1602.02697v4*, 2017.

[25] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[26] Google Brain Team. TensorFlow. (accessed 2019-07-08) `https://www.tensorflow.org`.

[27] Michal Zalewski. p0f v3. (accessed 2019-07-01) `http://lcamtuf.coredump.cx/p0f3/`.

[28] Wei Zhou, Yan Jia, Anni Peng, Yuqing Zhang, and Peng Liu. The Effect of IoT New Features on Security and Privacy: New Threats, Existing Solutions, and Challenges Yet to Be Solved. *IEEE Internet of Things Journal*, 2018.

A

Appendix 1

A.1 Outcome Matrix Summary

The number of adversarial samples based on a given device class that successfully fools either none (Ø), substitute only (S), black-box only (B) or both (SB).

Device Class	Ø, S, B, SB
Audio, Imaging or Video equipment: IP camera/D-Link IP camera	58, 40, 2, 0
Hardware manufacturer Azurewave Technology inc	58, 40, 2, 0
Hardware Manufacturer Cisco Systems inc	40, 40, 20, 0
Hardware Manufacturer Intel Corporate	20, 80, 0, 0
Hardware Manufacturer Samsung	6, 90, 4, 0
Internet of Things/generic IoT: Raspberry Pi	24, 40, 36, 0
Operating System Google OS	50, 40, 10, 0
Operating System Windows OS	30, 60, 10, 0
Phone/tablet or wearable: Generic android/Samsung Android	41, 50, 9, 0
Printer or scanner: Xerox printer	32, 60, 8, 0

A.2 Fingerprint Data

In this section, you can find the data used in this thesis.

No.	Device Name
1	Hardware Manufacturer/AzureWave Technology Inc.
2	Hardware Manufacturer/Cisco Systems Inc
3	Hardware Manufacturer/Intel Corporate
4	Hardware Manufacturer/Samsung
5	Internet of Things (IoT)/Generic IoT/Raspberry Pi
6	Operating System/Apple OS
7	Operating System/Google OS
8	Operating System/Google OS/Android OS
9	Operating System/Linux OS/Generic Linux
10	Phone Tablet or Wearable/Apple Mobile Device
11	Phone Tablet or Wearable/Generic Android/Huawei Android
12	Phone Tablet or Wearable/Generic Android/Samsung Android
13	Printer or Scanner/Xerox Printer
14	Router Access Point or Femtocell/Wireless Access Point/Ruckus WAP
15	Audio, Imaging or Video Equipment/Camera/Surveillance Camera/D-Link IP Camera

Table A.1: Device names mapped to a key used in the following tables.

Device	MAC address
1	6c:ad:f8:d2:83:37
2	54:75:d0:ba:41:2f
3	1c:4d:70:20:2d:7b
3	e0:9d:31:6f:95:50
4	30:07:4d:8f:cf:3d
4	80:4e:70:db:46:04
4	d0:b1:28:53:78:21
5	b8:27:eb:04:78:33
6	10:40:f3:a7:ee:38
6	34:36:3b:cb:28:68
6	3c:15:c2:c7:53:94
6	40:9c:28:d9:27:4b
6	60:f4:45:10:de:ef
6	a4:5e:60:dc:10:b7
6	dc:a9:04:99:3f:39
6	f4:0f:24:2a:fc:e7
7	6c:ad:f8:d2:83:37
8	6c:c7:ec:00:9d:4c
9	1c:4d:70:20:2d:7b
10	40:9c:28:d9:27:4b
11	10:b1:f8:b1:31:9e
12	08:78:08:17:24:78
12	44:78:3e:29:aa:c2
12	84:b5:41:9b:4e:a6
12	88:ad:d2:e6:14:c6
12	d0:b1:28:53:78:21
13	00:00:00:00:00:00
14	24:c9:a1:24:b7:62
15	6c:72:20:c5:17:5a
15	1c:5f:2b:aa:fd:4e
15	90:8d:78:a9:3d:6f
15	90:8d:78:dd:0d:60
15	90:8d:78:a8:e1:43

Table A.2: MAC addresses of the devices used.

Device	MDNS services
1	[_674A0243._sub._googlecast._tcp.local]
1	[_CC32E753._sub._googlecast._tcp.local, _googlecast._tcp.local]
1	[_googlecast._tcp.local, _37F83649._sub._googlecast._tcp.local]
1	[_googlecast._tcp.local, _674A0243._sub._googlecast._tcp.local]
1	[_googlecast._tcp.local, _CA5E8412._sub._googlecast._tcp.local]
1	[_googlecast._tcp.local, _CC32E753._sub._googlecast._tcp.local]
1	[_googlecast._tcp.local, Chromecast-1031c5fbd96dc45d4184bb5fb73e854b._googlecast._tcp.local]
1	[_googlecast._tcp.local]
2	-
3	[b.5.8.6.6.1.5.6.5.3.b.b.1.8.b.4.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.e.f.ip6.arpa, 19.50.168.192.in-addr.arpa, ASUS-Ubuntu.local]
4	[64.50.168.192.in-addr.arpa, 5.5.5.7.9.D.8.1.7.8.2.C.E.F.2.A.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6]
4	[75.50.168.192.in-addr.arpa, 4.0.6.4.B.D.E.F.F.F.0.7.E.4.2.8.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6, Android-2.local]
4	[75.50.168.192.in-addr.arpa, 4.0.6.4.B.D.E.F.F.F.0.7.E.4.2.8.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6]
4	[Android-2.local]
4	[106.50.168.192.in-addr.arpa, 1.7.D.1.C.4.F.1.A.2.4.7.C.8.8.C.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6]
4	[65.50.168.192.in-addr.arpa, 1.7.D.1.C.4.F.1.A.2.4.7.C.8.8.C.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6]
5	[b.b.1.7.5.f.7.6.b.2.2.c.3.0.b.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.e.f.ip6.arpa]
6	[_companion-link._tcp.local]
6	[_companion-link._tcp.local, SE-IT-SPARE-MB._device-info]
6	[SE-IT-SPARE-MB._companion-link._tcp.local, _services._dns-sd._udp, 78.50.168.192.in-addr, 4.4.4.A.7.7.8.4.6.7.4.3.A.C.C.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6.arpa]
6	[SE-IT-SPARE-MB._companion-link._tcp.local, _services._dns-sd._udp, 78.50.168.192.in-addr, B.3.1.B.E.D.9.8.D.1.0.2.3.C.C.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6.arpa]
6	[SE-IT-SPARE-MB._companion-link._tcp.local, _services._dns-sd._udp]
6	[SE-IT-SPARE-MB._companion-link._tcp.local]
6	[_kerberos.****s-MacBook-Pro.local, D.8.A.8.1.1.3.3.7.F.2.3.1.4.4.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6.arpa, 66.50.168.192.in-addr]
6	[_rfb._tcp.local]
6	[_sftp-ssh._tcp.local]
6	[*****'s MacBook Pro._ssh._tcp.local, _services._dns-sd._udp, .\x00, \x01\x00\x00\x00x\x00\x06\x91\x00\x02\x00\x08µ\x00/\x01\x00\x00\x00x\x00\x06µ\x00\x02\x00\x08\x00\x00)\x05\x00\x00\x11\x00\x12\x00\x04\x00\x0e\x00r\x00\x08a46,(h, ****s-MacBook-Pro.local]
6	[**'s MacBook Pro._ssh._tcp.local, _services._dns-sd._udp, .\x00, 66.50.168.192.in-addr, **'s MacBook Pro._device-info]
6	[_companion-link._tcp.local, *****'s MacBook Pro._device-info]
6	[****'s MacBook Pro._companion-link._tcp.local, _companion-link._tcp.local, ****'s MacBook Pro._device-info]
6	[*****'s MacBook Pro._companion-link._tcp.local]
6	[_companion-link._tcp.local, CZ3035._device-info, CZ3035._companion-link._tcp.local]
6	[_companion-link._tcp.local, CZ3035._device-info]
6	[8.8.9.8.A.A.6.E.D.F.D.F.1.5.4.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6.arpa, 58.50.168.192.in-addr, CZ3035._companion-link._tcp.local, _services._dns-sd._udp]
6	[CZ3035._companion-link._tcp.local]
6	[8b830790._sub._apple-mobdev2._tcp.local, 40:9c:28:d9:27:4b@fe80::429c:28ff:fed9:274b._apple-mobdev2._tcp.local, _services._dns-sd._udp, 2.2.1.F.8.4.4.3.4.B.7.3.5.1.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6.arpa]
6	[goat._companion-link._tcp.local]
6	[_companion-link._tcp.local]
6	[CYXSEMB0003._companion-link._tcp.local, _services._dns-sd._udp, 25.50.168.192.in-addr, B.9.9.D.2.7.4.0.5.9.A.0.8.E.C.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6.arpa, _services._dns-sd._udp.local]
7	[_233637DE._sub._googlecast._tcp.local, _googlecast._tcp.local]
7	[_233637DE._sub._googlecast._tcp.local]
7	[_674A0243._sub._googlecast._tcp.local, _CC32E753._sub._googlecast._tcp.local, _googlecast._tcp.local]
7	[_674A0243._sub._googlecast._tcp.local, _googlecast._tcp.local, _ED01B6D7._sub._googlecast._tcp.local]
7	[_674A0243._sub._googlecast._tcp.local, _googlecast._tcp.local]
7	[_674A0243._sub._googlecast._tcp.local, Chromecast-1031c5fbd96dc45d4184bb5fb73e854b._googlecast._tcp.local, _googlecast._tcp.local]
7	[_674A0243._sub._googlecast._tcp.local]
7	[_googlecast._tcp.local, _233637DE._sub._googlecast._tcp.local]
7	[_googlecast._tcp.local, _674A0243._sub._googlecast._tcp.local]
7	[_googlecast._tcp.local]
7	[Chromecast-1031c5fbd96dc45d4184bb5fb73e854b._googlecast._tcp.local, _googlecast._tcp.local]
8	[84.50.168.192.in-addr.arpa, 4.2.D.E.B.C.D.C.F.B.1.F.9.F.8.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6]
8	[61.50.168.192.in-addr.arpa, 4.2.D.E.B.C.D.C.F.B.1.F.9.F.8.5.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6, Android-3.local]
9	-
10	[8b830790._sub._apple-mobdev2._tcp.local, 40:9c:28:d9:27:4b@fe80::429c:28ff:fed9:274b._apple-mobdev2._tcp.local, 24.50.168.192.in-addr, _services._dns-sd._udp, 2.2.1.F.8.4.4.3.4.B.7.3.5.1.0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6.arpa]
10	[8b830790._sub._apple-mobdev2._tcp.local]
11	-
12	[65.50.168.192.in-addr.arpa, 1.7.D.1.C.4.F.1.A.2.4.7.C.8.8.C.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6, Android.local]
12	[Android.local, 65.50.168.192.in-addr.arpa, 1.7.D.1.C.4.F.1.A.2.4.7.C.8.8.C.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.E.F.ip6]
13	-
14	-
15	[D-Link DCH-S160 Configuration Utility._http._tcp.local, _dhnap, a.5.7.1.5.c.e.f.f.f.0.2.2.7.e.6.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.e.f.ip6.arpa, _http, dch.local]
15	D-Link HNAP Service._dhnap._tcp.local, a.0.5.6.c.a.e.f.f.f.1.6.7.8.c.7.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.e.f.ip6.arpa,dchgFD4E, _services._dns-sd._udp.local, dchg.local]
15	[D-Link HNAP Service._dhnap._tcp.local, dsp.local, 176.10.10.10.in-addr, f.6.d.3.9.a.e.f.f.f.8.7.d.8.2.9.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.e.f.ip6.arpa, dsp3D6F]
15	[D-Link DCH-S220 Configuration Utility._http._tcp.local, _dhnap, 0.6.d.0.d.d.e.f.f.f.8.7.d.8.2.9.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.e.f.ip6.arpa, _http, 39.10.10.10.in-addr]
15	[D-Link DCH-S150 Configuration Utility._http._tcp.local, _dhnap, _http, 3.4.1.e.8.a.e.f.f.f.8.7.d.8.2.9.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.e.f.ip6.arpa, 72.10.10.10.in-addr.arpa]

Table A.3: MDNS services fingerprints.

Device	TCP SYN Signature
1	4:64+0:0:1460:14480, 6:mss, sok, ts, nop, ws:df:+
2	4:119+0:0:1380:60192, 8:mss, sok, ts, nop, ws::+
3	4:64+0:0:1460:29200, 7:mss, sok, ts, nop, ws:df, id+:+
4	-
5	-
6	-
7	-
8	-
9	-
10	-
11	-
12	-
13	-
14	-
15	4:64+0:0:1460:5840,1:mss,sok,ts,nop,ws:df,id+:+

Table A.4: Device signatures based on TCP SYN messages.

Device	Destination Hosts
1	-
2	-
3	[api.fingerbank.org, connectivity-check.ubuntu.com]
3	[api.fingerbank.org]
3	[connectivity-check.ubuntu.com]
3	[gitlab.com]
3	[api.fingerbank.org]
3	[cdn.hitta.se, tpc.googlesyndication.com, www.facebook.com, www.googletagservices.com, track.adform.net]
3	[connectivity-check.ubuntu.com]
3	[i.ytimg.com, youtube.com, www.youtube.com, id.google.se, api.snapcraft.io]
4	-
5	-
6	-
7	-
8	-
9	-
10	-
11	-
12	-
13	[api.fingerbank.org, connectivity-check.ubuntu.com]
13	[api.fingerbank.org]
13	[clients2.google.com, accounts.google.com, api.snapcraft.io, www.google.se]
13	[connectivity-check.ubuntu.com, api.fingerbank.org]
13	[connectivity-check.ubuntu.com]
13	[gitlab.com, api.fingerbank.org]
13	[i.ytimg.com, youtube.com, id.google.se, www.youtube.com, api.snapcraft.io]
13	[safebrowsing.googleapis.com, api.fingerbank.org]
13	[ssl.gstatic.com, track.adform.net, tpc.googlesyndication.com, www.facebook.com, www.googletagservices.com]
14	-
15	[r0802.dch.dlink.com, r0801.dch.dlink.com, api.dch.dlink.com, eu-api-elb-1068129369.eu-central-1.elb.amazonaws.com, s3-us-west-2.amazonaws.com]
15	[r0802.dch.dlink.com, r0801.dch.dlink.com, api.dch.dlink.com, s3-us-west-2.amazonaws.com, ntp1.dlink.com]
15	[r0801.dch.dlink.com", api.dch.dlink.com, eu-api-elb-1068129369.eu-central-1.elb.amazonaws.com, s3-us-west-2.amazonaws.com, ntp1.dlink.com]
15	[api.dch.dlink.com, ntp1.dlink.com, r0802.dch.dlink.com, r0801.dch.dlink.com, s3-us-west-2.amazonaws.com]
15	[r0802.dch.dlink.com, r0801.dch.dlink.com, api.dch.dlink.com, s3-us-west-2.amazonaws.com, dualstack.tzinfo-423124427.us-west-2.elb.amazonaws.com]

Table A.5: Unresolved destination host names.

Device	DHCP Vendor
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	android-dhcp-9
9	-
10	-
11	HUAWEI:android:VKY
12	android-dhcp-8.0.0
12	android-dhcp-7.0
13	-
14	udhcp 1.15.2
15	udhcp 1.21.1–LSDK-10.2-00082-4
15	udhcp 1.21.1–LSDK-10.1.432

Table A.6: Strings identifying the DHCP implementation.

Device	DHCP Fingerprint
1	-
2	-
3	-
4	-
5	-
6	1,121,3,615,119,252,95,44,46
6	1,121,3,6,15,119,252
7	-
8	1,3,6,15,26,28,51,58,59,43
9	1,28,2,3,15,6,119,12,44,47,26,121,42,249,33,252
10	-
11	1,3,6,15,26,28,51,58,59,43
12	1,3,6,15,26,28,51,58,59,43
13	-
14	1,2,3,6,12,15,28,42,43,44
15	1,3,6,12,15,28,42

Table A.7: Fingerprints generated by DHCP traces.

Device	uPnP Server Strings
1	-
2	-
3	-
4	-
5	-
6	-
7	-
8	-
9	-
10	-
11	-
12	-
13	-
14	Linux/2.6.32.24 UPNP/1.0 ZD1106/9.7.1.0
15	-

Table A.8: Server strings used in the uPnP protocol.