UNIVERSITÀ DEGLI STUDI DI TRENTO

UNIVERSITY OF TWENTE.

Department of Information Engineering and Computer Science

MASTER'S DEGREE IN

Security And Privacy

FINAL DISSERTATION

TITLE

IOT Device Profiling for honeypot generation

SUPERVISOR UniTrento

Prof. Fabio Massacci

[illegible]

SUPERVISOR University of Twente

Dr. Andreas Peter

[illegible]

STUDENT

Filip Karchev

student number Trento: MAT. 204976

student number Twente: s2021234

ACADEMIC YEAR: 2018-2019

Contents

Extended abstract

In the next year(2020) it is estimated that there will be more than 20 billions IOT devices[1]. Some of the areas they can be found are: home automation, industrial sector, automated vehicles and much more. The diversity of functionalities they posses facilitate many tedious daily tasks, or improve significantly our productivity. The easy access and control over these "things" change the way we communicate with other people and the world around us. However, this expansion of connectivity hides many new risks that have not been observed before. Due to the severe competition between IOT manufactures, many of them decide that they will have success when they reduce the price of their products. To do so, they reduce the quality of the hardware components they use and skip important software development practises just to reach the market as soon as possible. Hence, their products become insecure and potential victims to cyber criminals.

From decades cybersecurity experts are trying to protect the digital world from every new threat that the hackers create. That constant battle between "good" and "evil" has significantly changed the security features involved in the technologies used today. To successfully react to new challenges, the security experts need to understand what are the intentions of an attackers and how they try to penetrate given product. One approach that can answer these questions without risking the security of a real product is using honeypots. A honeypot is a virtual clone of a given device or a service that aims to trick the attacker to believe that have hacked the targeted device. Then the honeypot is able to determine what actions are being performed from the hacker and possible to collect any files that the attacker has uploaded. But in order to successfully fool the hacker, the honeypot should work as close as possible as the original device. Such similarity requires knowledge of the technology the device is using and the services it provides. In this document we will refer to this knowledge as the profile of the device. However, there are thousands of types of IOT devices and this makes it impossible to have a profile for every one of them.

In this document I will present the results of a research I have conducted over how IOT devices can be profiled. I will use my findings to create a first prototype of an IOT Profiler project. I will use the obtained profile to configure another project called the server. The server then will be able to generate a virtual copy of the device. Furthermore, I will use different approaches and penetration testing tools to compare the results from the original IOT device and the created honeypot. Finally, I will explain how such profile should be extended in the future to optimize its performance and results.

Acknowledgements

I would like to thank Simon Dimitriadis, the Project manager of the IOT Decoy project, who was my mentor and assisted me during my internship by following my progress and guiding me with the Cybertrap vision of the project characteristics.

I would also like to express my gratitude to all Cybertrap personnel who were involved in the project to any extend. In particular those are: the CTO of Cybertrap - Avi Kravitz, the head of the Research and Development department - Stefan Schwandter and the lead developer of the Decoy implementation - Patrick Pacher.

I would also like to thank Fabio Massacci, professor at the di Ingegneria e Scienza dell'Informazione (DISI)" of the University of Trento and member of the DISI Security Research Group, who supervised me during my internship and the preparation of my master thesis.

Part I

Context and Background

Chapter 1

Introduction

1.1 Problem

The uprising number of devices connected to Internet create new possibilities, but also create new threats. Nowadays every gadget could be made ”smart”, by giving the user the possibility to control it and adjust it to the needs and desires they have. The modern term for these things that are connected to Internet is IOT(Internet of Things). This global description includes devices from every major field. These devices aim to increase our comfort or productivity. With a quick glance over a home equipped with IOT devices we can see a situation which just only two decades ago could only be part of a science fiction movie. Inside such home we can see variety of smart devices like: sensors, relays, cameras, lamps, fridges and much more. Even the couch we were used to relax after a long day at work could be now connected to internet and controlled to adjust our needs.

Like any other system that has not be designed properly, these devices give the possibilities to be used in situations outside of the scope their manufacturer desired. The immerse number of IOT devices and the poor or non security decisions taken into account make this domain one of the top goals for black hat hackers. The attackers easily obtain access to such vulnerable devices. Then, the devices can be used for different tasks which will gain some benefit for the attacker.

Some of the most recent notorious and global attacks involve usage of IOT devices. One example is from 2016 where more than 600k IOT devices were infected and become the base of one of the biggest botnets ever registered[2]. The name of that botnet is Mirai and it was used for massive distributed denial of service attacks(DDOS) all around the world. The source code of Mirai was later on published as open source. It lead to significant increase on the number of people which try to use it. The complexity of the attack also drastically increased, which made it possible for new type of devices to be controlled. The victims of the attacks ranged from game servers, telecoms, and anti-DDoS providers, to political websites and even other Mirai servers.

Second example of an IOT based attack is focused on another hot topic in the digital world recently - blockchains and crypto currencies. Alongside with PCs and mobile devices, IOT are a major player in crypto mining[3]. Crypto mining is a process where a user participates in cryptocurrency calculation operations where they are rewarded with small amount based on the level of their participation. Usually crypto mining requires very powerful hardware capable of calculating heavy tasks for a short time. Hence, IOT devices are not a logical source for such operations based on their limited resources. However, the huge number of such devices and the easy access some of them provide for the attacker, make them intriguing goal for such attacks. Being part of a cryptomining network increases the power consumption of the IOT device and leads to direct financial lost for their owners.

Mirai and crypto mining are just two examples of the increasing number of attack vectors which involve IOT products. To reduce or even eliminate potential attacks against the IOT domain, new security approaches should be involved. Unfortunately, the biggest factors that make IOT devices so vulnerable would hardly be improved soon. These factors are the lack of security precautions that have been taken from the creator of the device and the poor understanding of the user how they should protect themselves. There are also many users who want to protect their systems from possible breaches, but they do not know if the devices they have are vulnerable and how to protect them. To increase our knowledge we aim at obtaining direct information from the attacker how they approach given device, what they do to attack it and how they use an infected IOT device.

1.2 Introduction to honeypots

1.2.1 What is a honeypot?

Exploitation of newly discovered vulnerabilities is often unexpected and comes as a surprise for the system administrators of a given system. Freely available databases with possible exploits and multiple tools for massive global scanning for vulnerabilities enable adversaries to compromise computer systems easily when the system is prone to vulnerabilities or shortly after new vulnerabilities become known.

One way to get early warnings of potential attacks over a given system is to install and monitor another computer program component on the same network that we expect to be broken into. Every attempt to contact these components via the network is suspect. We call such a system a honeypot. If a honeypot is compromised, we study the vulnerability that was used to compromise it. A honeypot may run any operating system and any number of services. The configured services determine the vectors an adversary may choose to compromise the system.

There are different types and varieties of honeypots based on their physical characteristics and level of simulation. A physical honeypot is a real machine with its own IP address. A virtual honeypot is a simulated machine with modeled behavior, part of which is the ability to respond to network traffic. Multiple virtual honeypots can be simulated on a single system.

Virtual honeypots are attractive for system administrators, because they require fewer computer systems, which reduces maintenance costs. Using virtual honeypots, it is possible to populate a network with hosts running numerous operating systems.

The concept of a honeypot begins in the early 90s of the 20th century and is widely used from many security companies to detect and deflect an unauthorized use of a given system[4]. An example of a recent usage of a honeypot technology is from 2017 when the Dutch police used a honeypot to detect and eventually shut down an online darknet market called Hansa.

An abstract and simplified model of how a honeypot is integrated in a production system and what is the main purpose of it is shown on Fig.1.1

1.2.2 Functionalities

The main functionalities that one honeypot can implement are:

Data Control: Contain the attack activity and ensure that the compromised honeypots do not further harm other systems. Out bound control without hackers detecting control activities.

Data Capture: Capture all activity within the honeypot and the information that enters and leaves the Honehoneypotynet, without hackers knowing they are being watched.

Data Collection: Captured data is to be securely forwarded to a centralized data collection point for analysis and archiving.

Attacker Luring: Generating interest of attacker to attack the honeypot

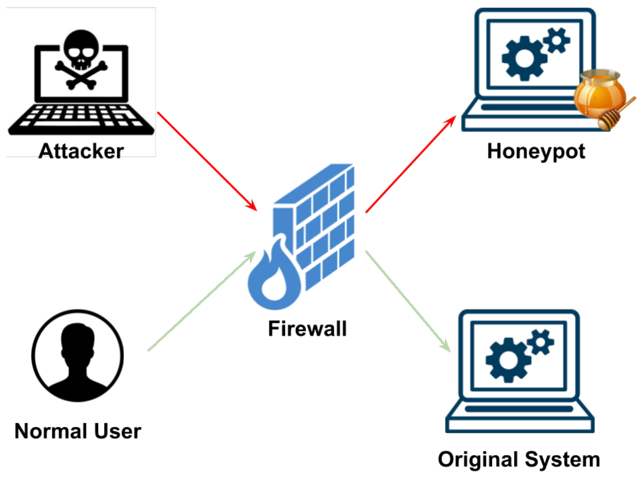


Figure 1.1: Honeypot Integration

Static web server deployment, making it vulnerable

Dynamic IRC, Chat servers, Hackers forums

1.2.3 Types

Based on the complexity and the design of their structure, honeypots can be divided into four categories:

pure honeypots Pure honeypots are fully functional production systems. The activities of the attacker are monitored and transmitted over the network. They do not require additional software to be installed. Hence, the level of control over them is limited and not suitable in many scenarios.

high-interaction honeypots High-interaction honeypots imitate the activities of a production systems. Usually they mimic variety of services and, therefore, an attacker may waste a lot of their time. It is possible to host multiple honeypots on one physical machine. Therefore, when a honeypot is breached, it can be quickly restored. In general, high-interaction honeypots provide more security by being difficult to detect, but they are expensive to maintain.

low-interaction honeypots Low-interaction honeypots simulate only the services frequently requested by attackers. They have a short response time, and less code is required. That limited complexity reduces level of security.

medium-interaction honeypots As Georg Wicherski describes in his paper about Medium-Interaction Honeypots[5], they try to combine the benefits of low and high-interaction approaches while removing their shortcomings. The key feature of Medium-interaction honeypots is application layer virtualization. These kind of honeypots do not aim at fully simulating a fully operational system environment, nor do they implement all details of an application protocol. All, that these kind of honeypots do is to provide sufficient responses that known exploits await on certain ports that will trick an attacker in interacting with the honeypot.

Deciding what type of honeypot would be created is critical for the proper implementation of it. This decision is based on the level of security one wants to support. But there is one more component that can determine the type of honeypot one should implement. This is the knowledge they have about the system that should be mimicked. Normally we consider that the creator of the honeypot have full access and information of the system that is being simulated. However, there are situations where this data is not available. Such scenario is when we want to generate the honeypot dynamically, without interacting with the targeted system in advance. Hence, in order to determine how that system works, the first step is to generate a fingerprinting profile of that device.

Looking at the IOT domain again and assuming we have full access to a given device, it would be easier to create a honeypot for it, compared to some more complicated structures. However, the diversity of functionalities, services and hardware make it impractical to create a honeypot for every device that we want to observe. That is why the approach I decided to focus on in this document is the dynamic generation of an IOT cloned device.

1.3 Contribution of the thesis

In this thesis, I research, propose and implement first prototype of a Medium interaction honeypot system for IOT devices. The created solution is independent from any device specific hardware. It works with application level simulation which helps in adopting the solution for any IOT device. In the first prototype of the project, the main focus is to determine the correct approach that would give fast and easily extendable solution for different types of integrated services. The project extracts a device fingerprint and creates a virtualisation system that emulates the information from that fingerprint. Several types of services were researched, analyzed and integrated in the project, the results of which helped me to determine the required steps for a more general solution. The created honeypot is tested and evaluated with multiple techniques that prove the potential of such solution and the advantages of the project over other systems that try to clone an IOT device.

Chapter 2

Company Case Study

2.1 Cybertrap

My research and implementation were performed during my internship in Cybertrap. CyberTrap is a cybersecurity company located in Vienna, Austria. The motto of the company is to ”always be ahead of the attacker to learn and improve.” Cybertrap achieves that by analyzing their client’s product, puts specific objects (called lures) on selected interesting places for an attacker and redirects the attacker to a decoy when they reach one of the lures. After that, Cybertrap follows every move of the hacker inside that decoy, by obtaining system and program logs. Cybertrap immediately informs the client when there is a breach on their system. Then they are able to determine any security flows in the system and analyze the most common attack vectors that have been used.

Many different honeypots have been created and used. What Cybertrap offers is not just a honeypot, but so called Deception platform. A Deception platform contains honeypots as essential components but go beyond that: they provide the automatic roll out and decommissioning of decoys and services that run on them. They allow for the automatic rollout of lures to the endpoints, which lead attackers to the decoys. Furthermore, they gather, analyze and visualize the collected data, enabling the forensic investigation of breaches and the support of counter actions.

Cybertrap has complex infrastructure, required to provide many valuable features to the customers. Some of the most important components are variety of possible lures to be installed, software to monitor and response arising attack actions, live dashboard for easy control of the system and much more. All of them are based on the idea that the Decoy will be able to fool the attacker and collect the information of their actions. A simple representation of the Cybertrap platform is shown on Fig.2.1

On Fig.2.1 we see that in the Production Network are situated specific lures for every component. When the attacker communicates with any of these components, the lure would be activated and any further interaction would be redirected to the monitored decoys. The main assumption of the decoys is

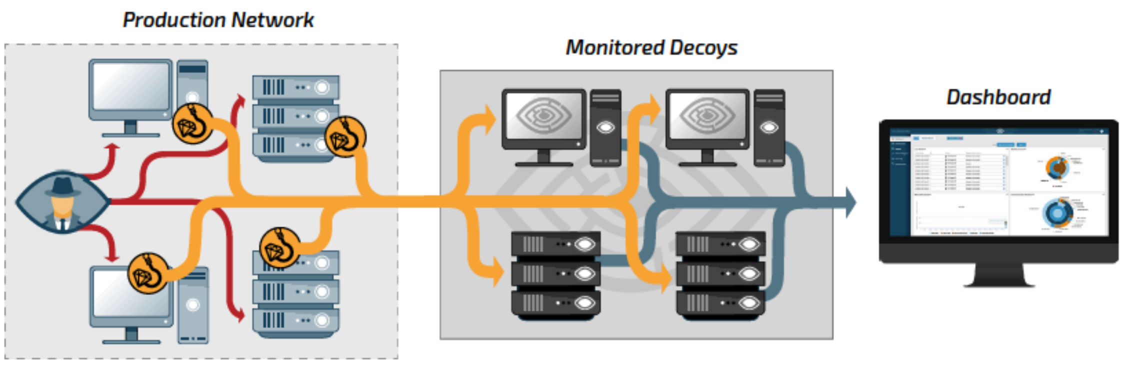


Figure 2.1: Cybertrap platform structure

that they generate similar behaviour which will keep the interest of the attacker and they will believe that they are interacting with the original product. For that purpose all of the monitored decoys need to be adjusted to the original product characteristics. The decoy provide the environment where all of the actions of the attacker are observed, monitored and analyzed in a safe isolated system that would keep all of the malicious attacks away from the company product. The decoys are connected to the dashboard server where they report any action that has been recorded. This data is analyzed and represented in a user-friendly way inside the dashboard website. The user is then able to observe the performed attacks on their system in a systematic way and to respond properly. From the dashboard, the user is able to set, create and modify all the lures and decoys that are integrated in their system in a way to get new and improved insights on the attacks. With all collected information, the user would now know what security problems are available on their system and how they should be mitigated. The desired result is an improved and more secure product that will be in constant state for further change when new attack vectors are registered.

The top features provided from Cybertrap are:

Endpoint deception CyberTrap implements the concept of endpoint deception. Lures that are placed on the endpoints within the production infrastructure direct the attacker to the decoys, where their actions are automatically monitored.

Web application deception CyberTrap can also be used to protect productive web applications. Lures are placed within the web applications that direct attackers to a deceptive web application hosted on a decoy to gather vital threat intelligence.

Tailor-made deception The deception environment appears to be a part of the production environment. Decoys are configured to look like production machines using deceptive services and data. A varied set of lures is deployed throughout the production network. Since the deception environment is tailor-made for each production environment, it is not finger printable.

Automated deception deployment The web interface allows for the rapid and dynamic creation of new deception campaigns. Services can be configured, filled with data and rolled out to the decoys. Lures can be generated and automatically rolled out to endpoints.

High quality threat intelligence The proprietary monitoring component is invisible to attackers and collects detailed information about every process, thread, file, network

High-confidence alerting Suspicious activity on the decoy triggers an alarm in the web interface, via syslog and email notifications. In addition, the TrackDown service provides alerts when deceptive documents are opened.

Data analysis The Dashboard user interface provides a visual overview of the deception environment enabling a quick overview as well as a detailed forensic analysis down to single system events.

Attacker infrastructure attribution The attribution algorithm of CyberTrap shows the connection between historical IP addresses and their corresponding domain names. An attribution of a command and control server (which is usually used by remote-administration-tool malware) reveals the infrastructure used by the attacker and can also predict from where the next attack could potentially originate.

Integration with security infrastructure CyberTrap integrates with MISP as a proxy for your security ecosystem and feeds SIEMs over syslog.

API All CyberTrap functionality can be accessed via a REST API. This enables the user to integrate the full CyberTrap functionality into existing security solutions.

In order to collect the proper data, the decoy should keep track of low level system calls and processes. To do so, the decoy should be able to communicate with and control the processes running on the Operating system level. Hence, the implementation of any Decoy software will be specific for any supported Operating system. This component would be the core of any decoy that will be created. It will be able to save and report the actions that are happening inside that platform. In a second step, this core solution that runs the same Operating System as the original product will be adjusted to the clients software characteristics. This will be done by installing specific programs and modifying important settings.

At the moment CyberTrap offers only Windows-based Decoys. Since the majority of web based applications are using windows servers, it gives Cybertrap the chance to reach and collaborate with the biggest group of potential clients. The goals of Cybertrap are not only to exploit this market share, but to create solutions for new clients that are not using windows as the OS of their products. For that reason, a new Linux-based Decoy is currently being developed. The first prototype of it has been already created and soon it will be distributed to test users and potential clients.

2.2 IOT Decoy

Another expanding area in the modern technologies in the last decade are IOT devices. Cybertrap wants to enter in this field. The goal of the company is to be able to create decoys for different type of IOT devices. In that domain there is no single Operating Systems(if any) adopted from every product. Hence supporting many types of devices would require significant investment in money and time for every one of the devices that should be supported. Therefore, the approach used for the already existing Windows and Linux decoy can not adopted.

The new corresponding approach should take into consideration the differences in the IOT world. For that reason Cybertrap considered that they should change from Operation System level tracking, to application level manipulation. The idea behind that is divided into 2 steps:

- The first is to scan an IOT device and create a fingerprinting profile of it. That profile should include every information that can be obtained and which can be used from an attacker to identify which is the scanned device.
- On the second step, that profile will be used from another software component to generate a honeypot.

The idea of using a profile could not be easily adapted in a way that the full functionality of the device would be included in that profile. Therefore, Cybertrap is interested in creating a Medium interaction honeypot. As I have already explained in the previous section, a Medium interaction honeypot works on the application layer and supports only the most important services a given product is running. As a first step, Cybertrap was mostly interested in devices that can be used in the industrial area. Some of the products they were focused on were printers and cameras and the corresponding services they use for printing a document or sending a video stream.

The described approach have many open questions that needed research. Some of them are:

- How such profile should be created?
- What format it should be?
- What information it should contain?
- Which are the services that should be supported and what information about them should be profiled?
- How the profile can be used to generate a medium interaction honeypot without knowing how that service actually works?
- What approaches or tools can be used to validate that the generated decoy is working correctly?

Answering to these questions is not trivial task and requires dedicated research that can elaborate if the goal for an IOT decoy is a feasible task and if the idea of generating a profile of that device is the proper way to do it. The research should answer what are the advantages and disadvantages of such approach and should compare them with other projects and solutions that have been focusing on creating honeypot for an IOT device.

2.3 Initial goals

The main focus in the performed research is to identify a scenario how a given unknown IOT device could be profiled. This profile should contain sufficient information so it can be the base of generating a honeypot clone(also referred as a decoy) of that device. The profile should also consists of all the data that can be used to identify that device(a fingerprint).

To cover to maximum extend how an IOT device works, I need to identify an approach and target the most valuable information of it. The full behaviour of any IOT device is considered as a combination of all services it is running. The variety of services that could be supported from a device in the IOT domain is huge and consists of thousands of possibilities. Many of them are custom protocols of the manufacturing company whose software is kept in secret. Due to these circumstances, creating a profiling tool that supports every possible service is extremely difficult task which at that point is considered as not necessary.

During the initial phases of the research and the vision Cybertrap had of the desired future product, I decided that I should focus on some specific services which are of their primary interest. For every one of them I have to identify which information should be part of the fingerprint and identify a scenario how I can use this fingerprinting information to create a clone copy of that service. After looking in several services I have to develop an approach that will be easy to integrate into other services so they can also be included in the profile. Hence, this approach should be as clear and as general as possible.

Covering only specific services is possible approach that will be sufficient for the initial device profile. Properly selecting these services which are of the main interest for an attacker will keep them busy when that honeypot is generated. As stated before, any custom company service will be very hard to be fingerprinted and supported in our profile. However, this service will also be unknown to the attacker itself. Most of the hackers have difficulties targeting something that they are totally unfamiliar with. I assume that when an attacker starts working on such devices they would try to penetrate the system through the services they are most familiar with and which are prone to security issues.

The performed research and future implementation should be synchronized with the desires, infrastructure and current products of the host company that would like to adopt the approach and continue its implementation. Many characteristics of the used technologies and the vision of the research should be systematically discussed with a company representative, so a maximum level of awareness and usability are achieved from the results of the performed research.

The process of how new service is analyzed to identify what information should be profiled would be based on the characteristics of that service in general, the complexity of it and the extend that the company want to support it. A valuable insights of how the service works and which are the primary points of interests for an attacker could be obtained by using any penetration testing tool that targets this service. Those tools would then be one of the main components that would be used to determine if the profile and the generated decoy are working correctly and the level of similarity to the original device based on the results of the tool findings.

After researching several services and how they can be simulated, the focus of the research would be concentrated to evaluate if the selected approach is sufficient enough for the goals of the company. I should suggest to what extend that profile could be generalized and how easy it will be to include new services that will be scanned, fingerprinted and simulated.

2.4 Role and Responsibilities

At the beginning of my internship at Cybertrap I was introduced with the current products of the company and with the idea of the IOT decoy project. My role was set to perform an individual research on the questions they needed answers so they can successfully evolve that project. My main focus was on creating a tool that can be used for scanning a given device and generating the fingerprinting profile of it. I also had to evaluate my findings by implementing the second part of the project that uses the profile to simulate a virtual device which is running the scanned services. During my internship I have been guided and advised by the project manager of the project - Simon Dimitriadis and with the lead developer for the Decoys implementation - Patrick Pacher. I was also performing regular meetings where I was presenting my progress and findings.

In the following sections I will present the results of the performed research, the findings for every of the analyzed services and what approach was selected to perform that service. Since one of the requirements of Cybertrap was that the server that generates the decoy should have minimum knowledge about the profiled service, all of the approaches that are used have general method of work. This makes them easy for adaption to other services which work on the same principal. Based on the main principal a given service works, I have designed several methods. This way for the majority of services I am able to use one of the approaches to simulate that service in the decoy. The services that have been analyzed are selected to differentiate in the way they work and primary purpose. Such diversity helped me to reach higher coverage of approaches and easy adoption in the next steps of the project.

Part II

My Contribution

Chapter 3

Solution Description

In this section I will present the structure of the project I have created to answer the research questions that have been assigned. I will present details of every of the components that are developed and I will explain why they have been designed and implemented in the selected way. Then I will present the results of the analysis of the researched services and based on the findings there I will explain step by step how the approach to integrate them in the profile have been performed.

3.1 Project Structure

There are three main components that are in the essence of the project. (1)A profiler is scanning the device to generate a (2)profile, which is transmitted to a (3)server that reads the profile and runs as a Honeypot(Decoy). This collaboration is presented on Fig. 3.1. Every one of these modules is explained in details in the following subsections. After the decoy is generated, the user is able to communicate with the decoy in the same way as the original device.

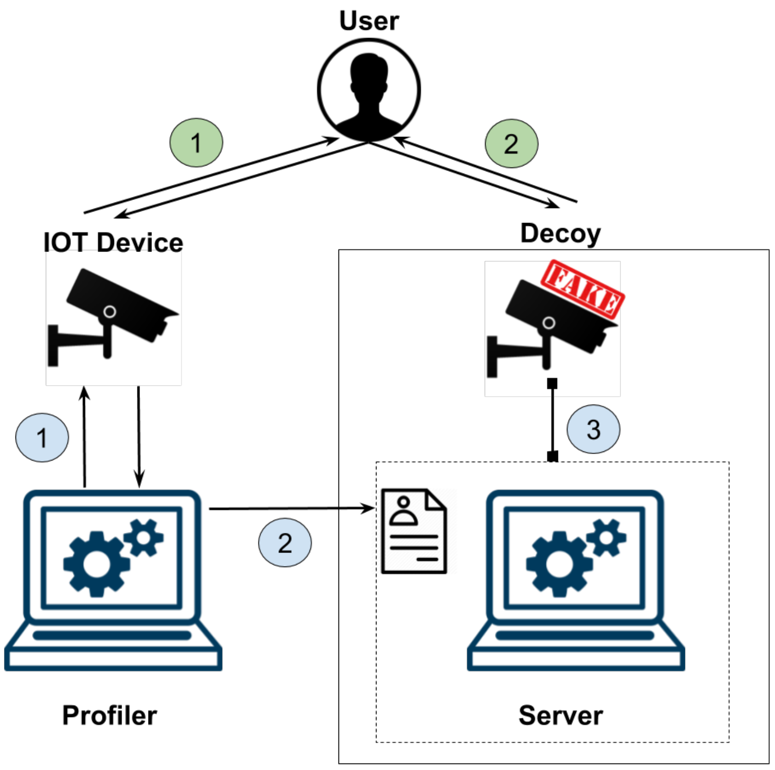


Figure 3.1: Project Structure Diagram

3.1.1 Profiler

The first software component is the profiling tool. It is the main focus on my research. The goal of the tool is to extract every valuable information about the targeted device that could determine the behaviour of that device and which would be used from the other main component (the server) to create a honeypot. That honeypot should be as similar as possible as the original device.

While working on the profiler there are several sub-questions that have to be addressed. The first one is to determine which information about an IOT device is necessary to generate such fingerprint. Any output, banner, header, parameter, the order of these elements, the format of an important request, the response itself and so on, could be interesting for us and may be used from the profiler to exfiltrate the fingerprint of the device.

The profiler contains several methods working together that provide the important information for a given scanned service that would be saved in the profile. These methods are:

Information stored inside the profiler Every service that is part of the profiler has been previously researched. Based on the findings from that research I became familiar how that service works, which information is specific for a given device and how it could be obtained as it's fingerprint. Therefore, the profiler knows which requests it should perform so it can receive that information and how it should be extracted. Later on, based on the complexity of the service and the coverage that is supported from the server, I collect either the specific device information, or everything necessary to mimic the service without further knowledge.

Information obtained during the behaviour of the device. During the profiling phase, I give the opportunity to the user to interact with their device in anyway they consider important and which they what to be inserted in the profile. The user is able to request information that is typical for their system. Such information could be: interesting folders, addresses and files that are usually not part of the standard use for that service. For some cases they might enter specific credentials required for accessing the content they would like to be part of the profile.

During this phase, the profiler is collecting every request that has been performed and the responses that the IOT device responds with. Working as a reverse proxy, the profiler is able to store all that valuable information without the need to take care of complication caused by possibly involved encryption.

Information obtained dynamically from a given penetration testing tool(vulnerability scanner) The third possible way used to collect information from an IOT device is to use third party software(tool). The expanding attack vectors for a given service makes it very difficult to include these requests in the profile. I assume that in most of the cases the user would not be even familiar if their device is vulnerable to specific attacks. By using external tools I expand the coverage of potential malicious requests that could be performed and which now will be saved in the profile. There are two ways any external tool can be incorporated inside the profiler.

1. The first one is by starting the tool from the profiler itself. This implies that there are a limited number of tools that are specifically selected. Their results have been analyzed and they have been chosen as the best way to represent attacks for the scanned service. Executing these tools can be done automatically from the profiler which guarantees that it is familiar when the the tools execution is completed and can even analyze the final output. Every tool however is another dependency for the project and it needs to be installed on the host machine or bundled inside the profiler project.
2. The second way to scan the device with any third party tool is to start it during the listening phase. Similar to the situation where the user interacts with the device, they would be able to trigger every tool they would like to be part of the analysis. This option creates the opportunity that the profile is not just a result of some previously configured requests and selected tools, but it also could be extended for any new source of information.

The disadvantage of this method is that the tool should be manually started from the user, which breaks the automation execution I have focused on and it also assumes that the user is aware how this should be executed. Hence, this method could be used as an additional feature provided to advanced users who want to extend or customize their profile.

3.1.2 The Profile

The profile is the end result of the scanning(fingerprinting) phase performed from the profiling tool. It contains all valuable information that is necessary to generate the decoy. The profile could even include additional data that the server is not using at a given moment, but could be used in future.

Format

Currently, the profile is a file with JSON format that is easily transferred between the two main components in the Project(from the profiler to the server). During my research, several formats were considered as potential way to store the fingerprinting data. The most promising of them have been analyzed and based on the initial priorities of the research JSON was selected. The list of considered formats that have been considered contains:

- XML
- protobuf
- JSON

The reason why JSON is selected is because it has human readable format which will be useful for debugging purposes. Another advantage of using JSON and in particular as a combination with Python projects, is that the conversion from a class object to a JSON file and the other way around is automatically handled by the system. This way the need to create and update a schema of the profile format is removed(such schema is required in the protobuf format). Hence, I can focus on other more relevant tasks of the research about profiling and decoy creation. However, there are other aspects which may be of big importance for the future of the project, which may require using another format of the profile. Such aspects could be the size of the profile(a protobuf file will have significantly smaller size) and the security of the information inside(in protobuf, the data is not human readable and the schema is required before it can be parsed)

Profile data

The information that the profile contains depends on the services that are found during the scanning of the device. Based on the results from the profiling, the complexity of a given service and the importance of that service, I can structure the results in the following groups:

Service independent information. Some fingerprinting information is not part of the analyzed services. They can represent hardware component data or information that is part of a lower level protocol of the Internet protocol suite[6], and hence that is used from every application level service that I am focusing on. Examples of service independent data are the fields contained in the TCP and IP level protocols. The information of these fields can be used from an attacker to identify the OS of the targeted device. Hence, this data is valuable for the fingerprint and should be stored in the profile. It can be used from the server to adjust the communication parameters used in every application level service that is being simulated.

Not supported service. Services that have not been yet analyzed are not part of the profiling phase. No data is included in the profile about the way they work. Only a minimum support coverage is presented. It contains only scanning for the presence of that service. As I have already state before, nmap[7] is used to discover the ports that are open on the IOT device during the scanning phase. All open ports are stored in the profile and simulated after that from the server. This would be useful in situations when the attacker is performing service discovery on all ports and the existence of open socket on a given port can be used to determine the device type. However, further scanning on that port would easily reveal that it is not a properly functioning service.

In a future step of the project I plan to include additional abstract level of scanning of every service that is not being further supported. This could help us identify if that service can be simulated with some of the automated approaches. However, with zero knowledge of the way that service works, it is a very complicated task that requires further research.

Automated services. For some services the typical behaviour could be easily extracted during the scanning phase. They are of our primary interest and the majority of the approaches explained later in this document are tackling these type of services. For them, there are limited number of requests which determine the base functionality of how that service works. For the goals of a medium interaction honeypot, some of the supported services are simplified by covering only specific service versions or the most used requests that an attacker would be interested in. There are several methods I use to store the service data in such way that it can be repeated after that from the server without much knowledge on the analyzed service. These methods are explained in the next subsection.

Fingerprinted services. Some services are more complex than the others and it is almost impossible that they can be fully simulated without the server to have extended knowledge of how that service works. For these services, the normal communication with the device dependants on many factors. For example exchanged settings in previous request would drastically modify any further communication. Other services do not have straightforward pattern of request and responses that can be simulated easily.

For such services, the generated profile contains specifically selected data that can identify the version of the service and some specific characteristics of it. This information can consist of request headers, banners, versions, dependencies and so on. They have been selected by careful analysis of the service specifications and behaviour. This data is then used from the server, to adjust a fully functional service running on the decoy. Such fully service coverage can only be achieved when a specific software that creates a server of the given service and is installed on the decoy. The data of the profile would just be used to make the proper adjustment on the software so it looks as similar as possible to the original device.

3.1.3 The Server

In the process of my current research, the server is a software component that is capable of reading the extracted profile and it creates the matching honeypot. The most important function of the server at this phase of the performed research project is to validate the results obtained during the scanning phase and help us identify the proper way a decoy should be created. The server can give valuable insights of what is actually useful to create a new virtual copy of any device from scratch. The server gives clear view of how the communication with the profiler should be updated in order to solve every limitation or an obstacle that is found in the process of decoy creation. However, the server is not designed to be fully functional honeypot that is ready to be used product. The implementation of such requires proper virtual machine that it will be working on, logging the events that occur, informing a back-end server for these events and much more. All these features are part of a future step in the whole IOT Decoy project. They would be implemented when there are significant insights of which information is important and how it could be used.

The result of running the server together with the previously generated profile, is the creation of a virtual device. It is running on the IP address that the server has been assigned with. Anyone with access in the same network can interact with that virtual device and verify to what extend the obtained responses are identical(or similar as possible) as the responses from the real device. This behaviour makes it very convenient to scan the real device and the decoy with the same commands or using the same third party tools and compare the responses. This comparison will give us clear observation to what extend the decoy is successfully mimicking the IOT device. If the results are not convincing enough, the user can create more advanced version of the profile by including more requests inside of it. It can be done by using more tools during the profiling phase which will increase the requests coverage from the decoy. Having more request would increase the possibility to respond properly to other attack vectors and would make the similarity between the decoy and the device bigger. This is possible due to the integrated approach of dynamically increasing the number of covered requests and the extendibility of the profile format that can save all desired data in a compact way.

In order to create the decoy from the profile, there are several steps that the server is performing:

1. The server reads the profile and creates a profile object from it.
2. For all the ports that are available in the Profile, the server opens a server socket, where it is listening for incoming connections.
3. When a new connection appears, the server is transferring further analysis of the request to a Manager class that is selected based on the port number which have been reached. Hence, for every supported service the server contains a Manager class that will analyze that specific service data is needed.
4. The manager that receives the request analyzes the data from the incoming request and then selects which data it should return back(if any).
5. If there is at least one data packet that should be returned, the Manager could perform different procedures to update the content of that response, based on parameters that are stored in the profile for that service(or for that concrete response).
6. The updated data packets are send back to the socket where the request was received.
7. For some of the services it is important to keep track for every new request that have been received. For them, the Manager that handles this request, is updating the status of that service.

These steps are illustrated in Fig. 3.2

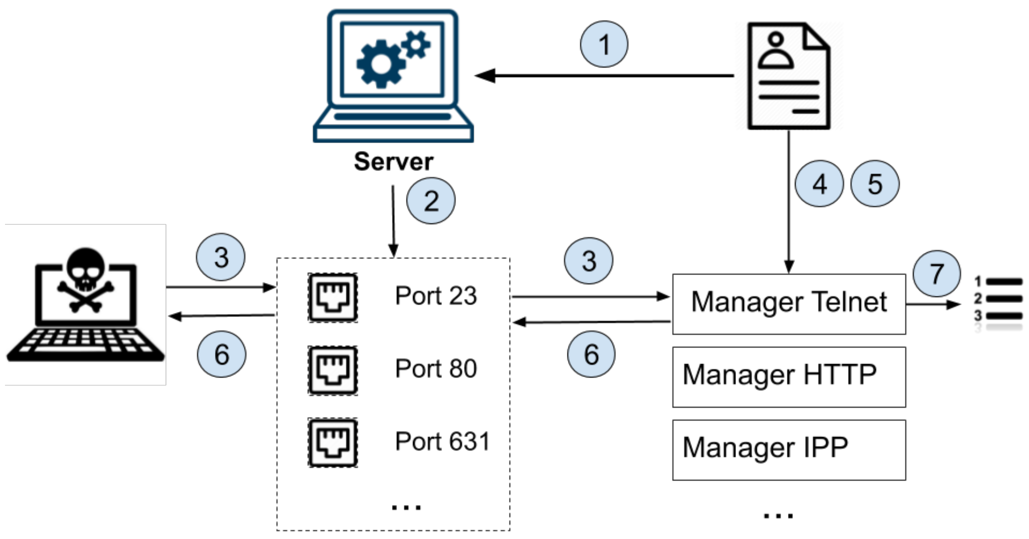


Figure 3.2: Steps for Decoy implementation

Having separate Manager class for every analyzed service is done with the purpose to individually handle any service specifications that can be observed during the research. However, the idea of having a general approach that is service independent on the server suggests having only one general Manager that takes care of the profile analysis. Only services which are simulated with individually installed server software should be separated. That goal is hardly possible at this initial phase of the research due to the different way the analyzed services work. After significant insights are received from how a service should be simulated, then a more general approach can be incorporated. For that purpose all scenarios to save service data explained in the next section, are not strictly service dependant but are designed with generalization perspective.

3.2 Scenarios for storing service data

3.2.1 Request-response scenario

This approach is used where there is no significant correlation between the currently performed and the following requests. For them I assume that the response is not dependant on some previous or future communication. This approach is used to simulate the behaviour of IPP and HTTP services. On Fig.3.3 We can see how this approach uses different ways to obtain valuable requests and how they are stored in the profile together with the received response.

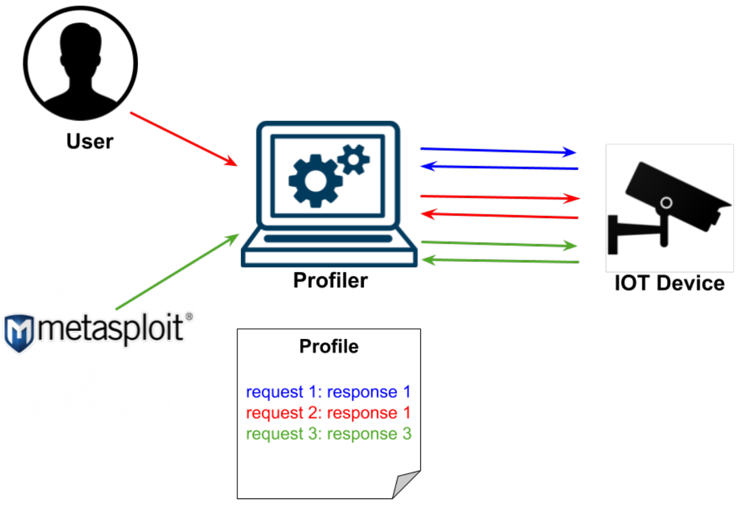


Figure 3.3: Request-Response based approach

In order to guarantee maximum correctness of the responses I have integrated several components that build the request-response approach. The first two components are, as the name suggests, the performed request and the received response.

The next component is used in order for the server to properly identify the correct request from all that are stored in the profile. I store every valuable information about the performed request. This includes the uri address, the GET and POST data, the size of the transmitted message and etc. Some of these fields are more valuable than others. For them, the value that is received can determine if a given request is the same or not. Others have less importance and their values are not considered from the server during profile analysis. For example a request can include parameters like username and timestamp when it was performed. While the username is valuable data, the timestamp can be ignored in the majority of the situations. To automate this process, for every request stored in the profile, I note the importance of the parameters which it contains. Based on them, the server is capable to properly say when the incoming request is presented in the profile. Note that this approach is only possible when that service has been previously analyzed and the important parameters are identified.

Another module is providing the ability to inform the server about the necessary differences between the raw data stored in the profile and an actual valid response that should be returned. Some examples that illustrate the necessity of such component are fields like:

date. The date when the request has been performed needs to be updated with the date that the incoming request is registered to the decoy.

ip address. In some situations the IP address of the IOT device is send inside the raw data. This address should be updated with the IP address of the Decoy.

request ids Some services, like IPP, has changing field for every received request that is assigned to it. The field should be updated with the new value coming in the request send to the decoy.

To guarantee this process, I save in the profile every difference that should be updated for a given request. By using multiple fields like contained data, regular expressions, exact location and more, I inform the server which part of the raw data should be modified. After the server finds the location of that data, it replaces it with the correct data also provided in the profile if possible(some differences like the date of the new request can not be known in advance and the server should be familiar how to proceed with them).

3.2.2 Specific sequence of requests and responses

The second approach combines the results of several following requests that are generating specific output. These order of the request is important and can determine which response should be returned from the server.

For most of the services which can not be supported with the request-response approach, it is because there are more than one response for every performed request. It is also possible that there are no responses at all. For these type of services usually there is initial phase(like handshake or negotiation communication) that first need to be performed from the two parties, before they reach the moment where they are exchanging the actual data that is the goal of that service emulation. Therefore, the order that these requests are coming is of significant importance when the server tries to profile that behaviour.

In the next section I will describe which services have been analyzed and integrated inside the project. During their research I have observed how this type of services work and realized that the order of received responses could vary. The main reason of this behaviour is caused of some concurrency between the operations inside the protocol. Unfortunately it affects the correct results when given response is matched to a request and saved in the profile. Hence, in order to avoid this kind of differences, I use an approach where I perform the captured requests with some delay. This eliminates the possibility to match a received response to the wrong request that is being sent.

Based on the service specifications and the level of simulation we plan to involve, the profiler obtains which requests determine the proper service communication. This process could either be done when these requests are stored in the profiler, or they can be obtained dynamically by analyzing the communication with the original device. The result of the profiling using this method is a list of requests that are following the behaviour of the device that should be cloned, the responses that are transferred for each of them and a way to determine if the responses are valid. The validation of every request would help the server to follow the created correct sequence of requests and return the proper responses if the validation is successful, or corresponding error messages when the validation fails.

Using one of the two approaches to capture the responses for a given service, I store the service specific information inside the profile. The biggest advantage of covering given service with the automated functionality is that the server which is reading the profile does not need to know how the service works. This level of abstraction is achieved by strictly following simple instructions integrated in the profile and using them to handle any incoming request by responding with the correct and updated responses.

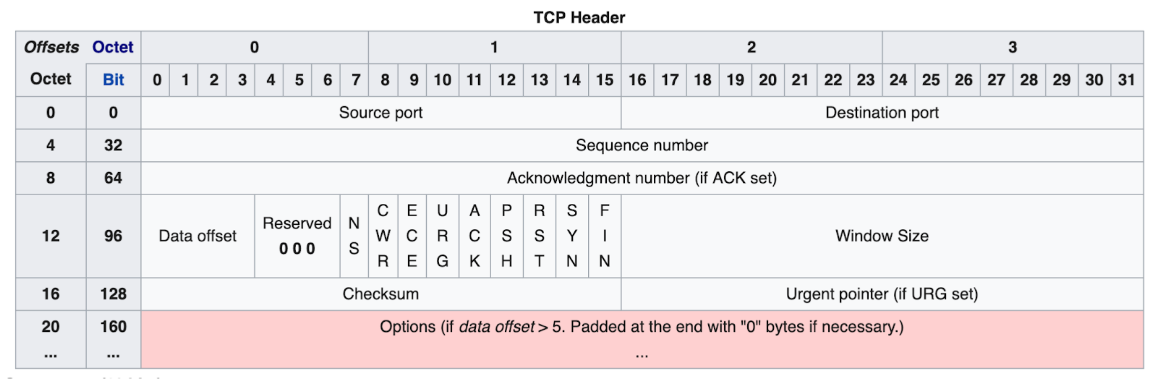
3.3 Analyzed services

3.3.1 Multi-Service data

In the current project I use a service based approach where I want to profile an IOT device behaviour by analyzing how the presented services work. The type of services which are interesting for simulation are those which the hacker can interact with easily. If we look at the Internet Protocol suite model, those are the application level services. They create the last layer of the model and work with the biggest abstraction from the device physical components. However, there are numerous other protocols and frameworks they depend on for their proper functioning. Some of these non application level services also contain information that could identify a specific device and hence they should be part of the device profile.

TCP

Most of the services that are supported in the project are based on the Transmission Control Protocol(TCP). TCP is a reliably protocol for delivering streams of bytes which represent every file that is being transmitted between two parties[8]. TCP divides every file that should be send into chunks, and adds a TCP header creating a TCP segment. The TCP header has predefined structure including different fields that are required for the proper functioning of the protocol. The structure of a TCP header is presented in Fig.3.4



TCP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 000			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

Figure 3.4: TCP header data

The construction of every TCP header is managed by the operating system through a programming interface that represents the local end-point for communication - the Internet socket. Hence, different operating systems have some differences on the way they instrument the TCP communication. Such changes in most cases are different default values for some of the fields that are part of the TCP header.

IP

The Internet Protocol (IP) is the main communications protocol in the Internet protocol suite for relaying datagrams across network boundaries. Its routing function enables internetworking, and essentially establishes the Internet[9]. IP generates the Internet layer that both TCP and UDP protocols depend on. Similarly to TCP, IP creates an IP header for every packet that is being transmitted. The structure of the IP fragment is shown in Fig.3.5

Both protocols(TCP and IP) are often referred together as TCP/IP which is the essence of the Internet protocol suite. Lippmann researched the device identification from the TCP/IP packet

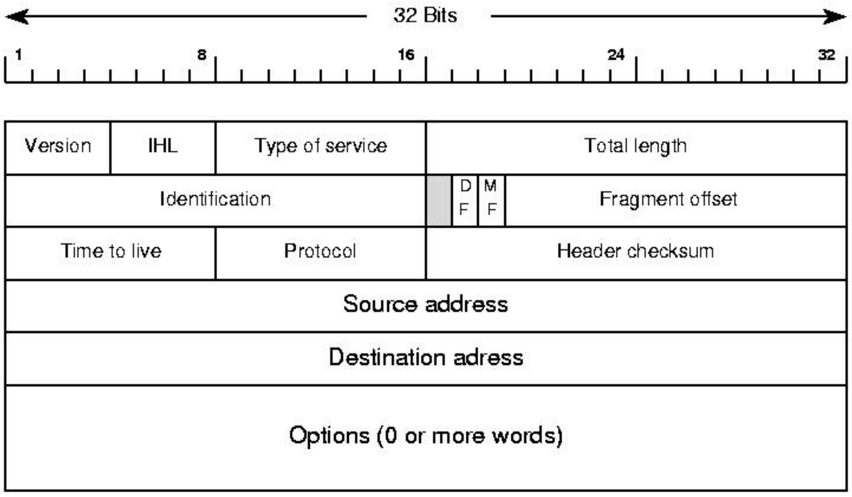


Figure 3.5: IP header data

headers[10]. On Figure.3.6 are shown his results where he identifies which fields in the headers of both protocols are used to identify the operating system of a device.

Feature Name	Range	Description
TCP Window Size (WS)	0-65,535	Data bytes a sender can transmit without receiving an acknowledgement equal to buffer size available on the receiver.
IP Time to Live (TTL)	0-255	Number of routing hops allowed before the packet is dropped, decremented by one by each router (prevents accidental routing loops).
IP Don't Fragment (DF)	0-1	Instructs routers not to fragment this IP packet, but to drop it if it is too large for the next network segment.
TCP Max Segment Size Option* (MSS)	0-65,535	Maximum size of data component of packet that a receiver can accept.
TCP Window Scaling Option Flag* (WSO)	0-1	Flag that indicates the TCP scaling option is being used to obtain bigger WS windows.
TCP Selective Acknowledgments Options Flag* (SOK)	0-1	Flag that indicates when the TCP selective acknowledgements option was set.
TCP NOP Option Flag* (NOP)	0-1	Flag that indicates one or more NOP's were added to align other options on a word boundary.
Packet Size (PS)	0-255	Length of packet in bytes.
TCP Timestamp Option Flag* (TS)	0-1	Flag that indicates one of the TCP timestamp options was included.
SYN vs SYN-ACK Packet Flag (SYN)	0-1	Flag set to one for SYN-ACK packets and zero for SYN packets.

Figure 3.6: TCP/IP features used to identify an Operation System

Inside the profiler, I extract the values of these fields from the received packets during scanning of an application level service which is based on the TCP/IP modules. I save this information inside the final profile. Since both protocols are managed by the operating system I am not able to directly set these field values inside the Server/Decoy source code. However, this information is important part of the fingerprint of any device and will be used in a future phase of the project. Modifying the values of the TCP/IP fragments can be achieved by using raw sockets[11] inside the server, which are currently supported in the majority of the modern programming languages. However, using raw sockets requires many other tasks that should be implemented pragmatically instead of handled automatically by the system. Another alternative is to directly set these values for the Internet Socket of that operating system. This is possible only when we have access to the kernel source code of the operating system, which is not part of the project scope at the moment.

3.3.2 IPP

What is IPP

The first service that has been researched and implemented is IPP. Cybertrap wished to support printer devices fingerprinting and the popularity of IPP over other printing protocols, gave me the confirmation that IPP is a good candidate for our project.

IPP(Internet Printing Protocol) is a secure application level protocol used for network printing[12]. It defines high-level requests that a client can use to ask the printer for a set of capabilities and settings. The client is also able to send direct commands to the printer and initiate tasks to print a document. IPP is supported by all modern network printers and supersedes all legacy network protocols including port 9100 printing.

IPP defines an abstract model for printing, including operations with common semantic. IPP uses HTTP as its transport protocol. Each IPP request consists of HTTP POST message with a binary IPP data and possibly a file send for printing. The corresponding IPP response is also structured as a POST response. The IPP protocol supports different levels of security. The connections can be unencrypted, TLS encrypted based on HTTP OPTIONS fields, or encrypted immediately with HTTPS.

To communicate with a printer using IPP, the client should use that printer address, also referred as Universal Resource Identifiers (”URIs”). There are two schemes that IPP supports: ”ipp” or ”ipps”, where the second one is using encryption. This URI is used by the client to send the desired operation following the protocol scheme encoding.

Integration in the Project

Integrating IPP in the project requires two steps. The first one is to properly adjust the profiler to make correct IPP requests, and the second step is to implement how the server should read the IPP data from the profile and responds with the right messages.

By analyzing the IPP protocol and the fact that it is running on top of the HTTP, I decided that the most appropriate way to simulate it is by using the request-response approach explained before. The two main reasons for this choice are the fact that every performed request results in exactly one response, and that every request is independent from the others(no sequence needed). Hence, it will give us the possibility to simulate IPP without the need to support fully functional IPP Server running on the decoy.

The IPP protocol have in total 16 Job operations[12] that can be used to support any task send by the user. It is also possible that the manufacturer of the device can introduce other custom operations that will be supported from that printer. On Table 3.1, are shown the most common IPP operations.

The limited number of possible operations that can be performed in interaction with the printer remove the uncertainty of which requests should be investigated during scanning. Covering these requests would ensure sufficient level of interaction that would satisfy a medium interaction honey-pot. From the 16 operations that are described in the protocol specifications, I selected the 10 most used by analyzing the communication of the printers with different client applications and tools. I have prepared the IPP Manager inside the profiler to perform these requests as a valid IPP client, by inserting all required fields and attributes. The list of supported operations could be easily extended in the future if that is considered necessary.

For easier and less prone to mistakes approach, I use Python library implementation for IPP requests[39], which I have updated to work with Python3 and according to the requirements of the

Table 3.1: List of most common IPP operations and their attributes

Operation	Required Attributes(syntax)	Optional Attributes (syntax)
Cancel-Job	job-id (integer), requesting-user-name (name)	
Create-Job	requesting-user-name (name) job-name (name)	Job Attributes
Get-Job-Attributes	job-id (integer), requesting-user-name (name)	requested-attributes(1setOf keywords)
Get-Jobs	requesting-user-name (name)	my-jobs (boolean), requested-attributes(1setOf keywords), which-jobs (keyword)
Get-Printer-Attributes		document-format (mimeMediaType), requested-attributes(1setOf keywords)
Print-Job	requesting-user-name (name), job-name (name)	Job Attributes
Send-Document	job-id (integer), requesting-user-name (name)	document-format (mimeMediaType), document-name (name)

project. The library eases the process to adjust the packets in the required IPP encoding format.

By having all important requests available, the profiler connects to the scanned device uri and performs them. The responses received from the printer contain all valuable information that identifies that device and that I need to simulate.

When all responses are received, the data is stored in the profile following the format of the request-response approach. Before being saved, the data is analyzed and all parameters that need to be updated from the server are specified with a list of differences. Then, the ready profile is transmitted to the server which is now able to read it's content and simulate IPP inside the decoy.

Server The purpose of the server is to validate if the selected approach is working properly, with minimum analysis of the requests and responses that are specific for the given service. The IPP Manager follows this idea and automates the profile parsing and returning of the correct response. However, the server needs to properly match the incoming request with one that is saved in the profile. It performs two tasks to correctly achieve it:

identify important parameters from the request Important aspect about the IPP protocol is that it uses custom encoding for the transmitted messages. Each IPP message starts with a version number (2.0 is the most common), an operation id (IPP request) or status code(IPP response), a request number, and a list of attributes.

On Table3.2 is presented the format of an IPP message for every available attribute. We see that the length of the message is not fixed in size, but is dependant on the value of the data that is being transmitted.

The server needs to parse the important data of the incoming request. It reads any parameter, attribute and their values following the explained IPP format encoding. All this data is required when the server searches if the new request is presented in the profile.

When the request search is completed it could sometimes result that the incoming request could not be matched to any of the saved requests in the profile. Hence, I need to have a fall-back

Table 3.2: Single Value Attribute Encoding in the IPP messages

Field	Length
value-tag	1 byte
name-length(value is u)	2 bytes
name	u bytes
value-length(value is v)	2 bytes
value	v bytes

response, which will be returned in such situations. The most logical and easy to implement approach is to capture an error message and save it together with the remaining 10 operations. However, IPP supports several different error messages based on the type of error that occurs. Supporting different errors will increase significantly the complexity and the analyses that the server should perform. Hence, one error type is selected (client-error-bad-request) which has general looking description and could be a result of different errors. In order to receive that error response in the correct format and fingerprint as the scanned printer, I am performing one more additional request that always results with client-error-bad-request output.

find the appropriate response By comparing the extracted request parameters with the data for every stored request in the profile, the server identifies the response that should be returned.

Another specification of a printer running the IPP protocol is that there might be multiple printer uris for one device. These uris are the addresses at which the communication is happening. The available uris could differ in variety of fields, hence providing different functionalities. For example a printer can have two printer addresses, one of which uses encrypted communication, while the other is unencrypted. Due to the way the selected approach works, having encryption is not problematic for our profiler and the results does change the output inside the profile. I had to integrate some new functionality in the server to make possible having an encrypted communication. One of them is using wrapped sockets with SSL layer, that give us the possibility to support encrypted communication and still be able to read the clear data of the requests and responses.

After the connection is established and the IPP parameters are parsed from the new request, the server is using the operation id of that request and the obtained parameters to identify which is the correct response it has to return. If all of the necessary parameters are valid and they match with a given request from the profile, the server returns it in the sockets after the differences have been updated. If no request from the profile correctly matches the request parameters, the error response is returned.

This approach has several assumptions and limitations. For example, it supports only one default error response, while the protocol supports many other variations. However, it is designed in such way to deliver maximum coverage of the most used IPP features without many of the complications that would arise if the aim is to support every scenario. The behaviour of the decoy and the correctness of the profile would later on be evaluated and it will answer the question to what extent that approach functions properly.

3.3.3 HTTP

What is IPP

According to the official rfc of HTTP/1.1(RFC 2616 [13]), The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It's most common usage is for data communication for the World Wide Web (WWW) with hypertext documents. However, HTTP can also be used for other tasks such as name servers and distributed object management system. A feature of HTTP is the typing and negotiation of data representation, which allows systems to be built independently from the data being transferred.

HTTP is used from variety of protocols which inherits all of the HTTP streaming and security feature. An example of such protocols is the already discussed IPP.

In that document I refer to HTTP as a service. It is actually HTTP server which usually runs on TCP port 80. The HTTP server waits for a client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own. The body of this message is typically the requested resource, an error message or other information. Often these resources present information which can be observed from user who access them from a web browser. The combination of accessible resources that run on the HTTP server are referred as a Website.

Many IOT devices also have implemented a HTTP server. That website can give information to the user about the current status of the device, or present them with a way to easily control the device. The only required component is a web browser which will be used as the client communicating with the HTTP server. Since Web browsers are implemented for the majority of operation system running on different hardware, that website can be accessed without further requirements.

Analysing HTTP content

Except the transmitted resource, an HTTP response contains several headers used for a proper communication with the client. Next is showed an example HTTP response containing several headers.

```
HTTP/1.1 200 OK
Date: Mon, 6 May 2019 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 138
Last-Modified: Wed, 07 Jan 2019 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
  <head>
    <title>An Example Page</title>
  </head>
  <body>
    <p>Hello World, this is a very simple HTML document.</p>
  </body>
</html>
```

If we take a look at the headers we can easily identify information about the device running the HTTP server and the system it is using. In the presented example there is a server header field that contains the type and version of the HTTP server that has been used.

Almost all HTTP servers differ in the way they implement the HTTP protocol. When the HTTP request is legitimate, the response returned by all HTTP servers is more or less valid with the specifications described in the RFCs for HTTP. However, when the client performs malformed HTTP requests, these servers differ in their responses. Differences in the way HTTP protocol is handled by various HTTP servers constructs HTTP fingerprinting technique.

Saumil Shah in his paper "An Introduction to HTTP fingerprinting"[14] looks at different methods used for HTTP fingerprinting.

Table 3.3: Results of HTTP fingerprinting techniques

Server	Field Ordering	DELETE Method	Improper HTTP version	Improper protocol
Apache/1.3.23	Date,Server	405	400	200
Microsoft-IIS/5.0	Server,Date	403	200	400
Netscape-Enterprise/4.1	Server,Date	401	505	no header

HTTP header field ordering. Usually the order of the HTTP headers is not important for the proper handling of an incoming request. However, such inconsistency caused by the different HTTP servers, can be used to identify what server is being accessed even without observing the server header.

HTTP DELETE (forbidden operation) response HTTP/1.1 declares 8 possible request method types. They are: GET, HEAD, POST, OPTIONS, PUT, DELETE, TRACE and CONNECT. However, not all of them are allowed from every server. Most of the HTTP servers allow the usage of only several method types. An attempt to make a request with method that is not allowed can result in different responses from every type of HTTP server. For example, a DELETE request method can receive response as Method Not Allowed(405), Forbidden(403) or Unauthorized(401). By observing the response to such attempt, an attacker is able to recognize the type of HTTP server by knowing how it is responding to that specific request method.

Improper HTTP version response Usually the first 8 bytes of an HTTP request contain the version of HTTP that will be used. Constructing HTTP request with non existing version of the HTTP protocol could also lead to variety of different responses from the different HTTP servers. In his example, Saumil Shah is starting the request with HTTP/3.0. As a response he receives messages with codes: Bad Request(400), HTTP Version Not Supported(505) or even OK(200). Knowing how a given type of server responds to such request would reveal itself, even when the server header is not presented.

Improper protocol response Another possibility to craft a non standard request in order to analyze the output is just by sending a non proper request. For example this could be achieved by replacing the HTTP version, the type of the request or the uri destination at the beginning of the HTTP request. Some of the tested servers are capable to handle such request, while others respond with Bad Request(400) or even they could not return HTTP response header, but instead just returns an HTML formatted error message stating that this request is a not correct.

A summary of the results and the tested HTTP servers could be seen at Table. 3.3. The version of the tested servers are very old, but the principal that every server could return different responses to a non typical requests is still valid approach for determining the HTTP server type.

All fingerprinting methods described in the previous section are important aspect I want to cover in our profile. However, the observed requests are just a small part of all possibilities of strange requests that a HTTP server could receive. That is why just storing the responses to these specific requests would not be an optimal solution for our fingerprinting tool. A way better approach would be to analyze different tools to determine what requests they are performing, so I would be able to take into account all possible attack vectors that are being addressed.

Another information that describes how our HTTP service works and would identify the device that runs it is the actual content of standard HTTP request. This data is probably the most important one, because it follows what is the normal way to interact with an HTTP server. Such data contains of all HTML files which are needed and all other resources and files they use. The reason I am interested in them is because they would probably contain information about the manufacturer, vendor, the device itself or the current state it is in. Furthermore, without supporting the most important requests that could be performed, the attacker would easily determine that our generated honeypot does not work properly and would eventually conclude that it is not the actual device.

One important characteristics of how HTTP works is that the server always responds to a received request. The number of send requests and the number of received responses should always be the same and the correlation between the requests and responses is 1 to 1. The inconsistency of some other protocols, where there could be multiple requests without received response, or multiple responses to a given request, is not observed in typical HTTP communication. Hence, if I know all headers, parameters, and data that is contained in a given request, the response should always be the same. Here I should mention that when the server is using a backend with a database, the responses could change, but the performed commands are still not altered.

I will use this HTTP characteristics to create our profile. Capturing all important requests and the responses that the server returns would successfully mimic the behaviour of any HTTP server and should be indistinguishable from the original source.

I have already introduced the request-response scenario. Similarly to IPP, it is the approach I used to fingerprint a HTTP server. I will capture the responses to all significant requests and store them in the profile. The work of the server now includes just finding when an incoming request matches to a request that has been stored, and return the corresponding response. This simple interaction eliminated the need for the server to know how HTTP works, by implementing such HTTP server.

During the process of analyzing the request-response tuple I faced two important questions that I need to answer so I can make the whole communication between the profiler and the server more general and corresponding to reality:

Determining when a given request is matched as existing in the profile. A HTTP requests could contain different parameters and headers. Some of them are with significant importance for the HTTP server to manage and return a proper response. Others do not have such value. For example it could be the timestamp of the date when the request was performed. This field should be ignored by our Decoy server when it validates the collected requests, because it will never receive a request with the same timestamp. However, the collected response data would probably still be valid. Determining which parameters are crucial to match a HTTP request could not be done without previous analysis of the results. Unlike IPP where we have limited number of request to cover, potential scanning of an HTTP server could result in thousands of request and they will be different for every device. This lack of standardization makes impossible to analyze the important parameters in advance. Hence, this decision is delegated to the server, which will make customizing the results easier. In the profiler, I just have to ensure that all the parameters are presented so the server could determine which information is critical for the validity of a request, and which could be ignored.

Updating the response data before returning it. Some of the parameters that are being send in a given request are required to establish a proper connection between the two parties. For example the client can put in the request it's IP address. When the raw response is directly copied inside the profile,these parameters remain untouched. After that the server would use this raw data as a response when the same request is received. However, the server IP address would not be the same as the one that is stored. Hence, the data inside the response would not be valid and the attacker would easily spot such inconsistencies. It is even possible that the connection would not be established at all, or the it will be dropped when the wrong parameter is being processed.

To properly react to these problematic fields, I have implemented the differences module. For every request I am storing the information that need to be updated accordingly.

Integration in the Project

The request-response approach that I am using strongly depends on the fact that I can identify which requests are important for the scanned HTTP server. I should also cover significant amount of requests, which are usually used from many vulnerability scanners or just some malicious request send from an attacker. That is why identifying the requests to cover has become of a major importance for the completeness of the profile for HTTP and for the proper functioning of the generated decoy that will use that profile.

All three methods to identify valuable request are used when HTTP is scanned. They have been integrated in the following way:

Information Stored inside the Profiler Figuring out which files are mostly used for HTTP is more difficult than protocols like IPP, where the protocol itself declares the possible commands that the server should support. There are techniques that are used to copy the content of a Website. One such technique is Web scraping[15]. It usually implements a bot or web crawler that searches for specific data which is gathered and copied from the web. That approach generates a file content which then can be copied inside a HTTP server which will act as the original server.

The method that is being used is similar to Web scraping, however it collects not the files themselves, but just the requests which are needed for the proper loading of the only page that all servers supports - the index page. The profiler triggers automatic loading of the index page inside a browser(from the profiler it is achieved using the selenium web-driver). Meanwhile, it sniffs the generated traffic. Then it distinguishes all necessary requests called when that index page is loaded. Next, all captured requests are performed again directly from the profiler and the responses are matched and stored. This additional step avoids possible problems with handshakes or encryption that can be used from HTTPS.

Information obtained by following the behaviour of the device. The profiler initiates a phase where a Reverse Proxy is created between the IOT device and the used machine running the profiler. At that moment any interaction with the HTTP server is observed. The user can load any specific page they think should be included in the profile that contains important information for the fingerprint of the device, or is with significant value for the normal usage of the device.

Information obtained dynamically from a given penetration testing tool(vulnerability scanner) Many users would not like to communicate with the device during profiling, or they will not be familiar which requests are interesting for an attacker. That is why executing one or more tools which are interacting with the HTTP server is also integrated inside the profiler.

The use of a reverse proxy give us the possibility to not be bound to just specific tools that are integrated inside the project. Every tool that makes requests to the HTTP port could be used to identify potentially interesting requests. Such tools could either be started from the user when the profiling is activated, or they can be triggered from the project. For demonstrative purposes and for easier adaptation from unfamiliar users, the current version of the profiler uses two well known tools for HTTP service scanning(nmap script and Nikto). The results from the performance of the profiler and the usability of the generated profile will be presented in the evaluation chapter.

Comparison to HTTPS

Hypertext Transfer Protocol Secure (HTTPS) is an extension of HTTP. It uses Transport Layer Security (TLS), or formerly its predecessor - Secure Sockets Layer (SSL), for encrypting the communication channel between the client and the server. Encrypted communication allows only the participating parties to be able to read the content of the transmitted messages. It eliminates the ability of an attacker to sniff the traffic, because the captured data will not be readable without the encryption keys.

Using encrypted connection significantly improves the security of any HTTP Web browser. Nowadays it is considered a must for any major website, especially if its usage requires user authentication or any other user information.

The necessity of using HTTPS over HTTP in the IOT domain is smaller compared to any other global internet communication. The reason for that is because many IOT devices require the device to be included inside a home wifi network. This adds some level of protection against unauthorized access with the device, because the attacker should first gain access to the protected network. Hence, the usage of HTTP in the IOT domain is significantly bigger compared to a website communication over internet. The required certificates and their installation and usage is a slight overhead of the usage of an HTTP server, that is the reason many IOT manufacturers or vendors decide not to use encrypted communication when the usage is restricted inside the local network. However, the integration of HTTPS is strongly advised even in there is not communication over Internet.

Integrating new service inside our profiler requires having access on the content of all requests and response that this service uses. That is why using approaches like traffic sniffing is not possible for HTTPS or any other service that uses encrypted data. However, the described approach for HTTP does not require traffic sniffing. The essence of the reverse proxy that is implemented give us direct access to the clear messages that have been transmitted. Hence, the only step that have to be included for supporting HTTPS inside the proxy listening phase is to wrap the socket with a SSL module. Such task have already been performed with ipps schema and should be easily adopted here.

3.3.4 Telnet

What is Telnet

The first version on Telnet is proposed in the distant year of 1969. According to the RFC Specification of the protocol(RFC 854 [16]) the purpose of the Telnet protocol is to provide a fairly general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other.

Typically, Telnet is used to establish a connection to Transmission Control Protocol (TCP) port number 23, where a Telnet server application (Telnetd) is listening. However, the first versions of Telnet occurred before the creation of TCP/IP. It was originally running over Network Control Program (NCP) protocols.

After its creation Telnet was adopted as the primary protocol for remote administration purposes[17]. During that time most users of networked computers were in the computer departments of academic institutions, or at large private and government research facilities. In this environment, security was not nearly as much of a concern. After the exploding popularity of Internet in the 1990s, these security issues become problematic for the future of Telnet.

Today the use of Telnet is not recommended. Security experts, such as SANS Institute[18], describe the following security issues as the main reasons why Telnet should not be used:

- Telnet, by default, does not encrypt any data sent over the connection (including passwords), and so it is often feasible to eavesdrop the communication and use the password later for malicious purposes. Anybody who has access to a router, switch, hub or gateway located on the network between the two hosts where Telnet is being used can intercept the packets passing by and obtain login, password and whatever else is typed with a packet analyzer.

- Most implementations of Telnet have no authentication that would ensure communication is carried out between the two desired hosts and not intercepted in the middle.

- Several vulnerabilities have been discovered over the years in commonly used Telnet daemons.

Due to the these drawbacks, the use of Telnet protocol dropped rapidly, in favor of the Secure Shell (SSH) protocol. Created in 1995, SSH has practically replaced Telnet. SSH provides much of the functionality of Telnet, with the addition of strong encryption to prevent sensitive data such as passwords from being intercepted, and public key authentication, to ensure that the remote computer is actually who it claims to be. Alongside SSH, there now exist several extensions to the Telnet protocol which provide Transport Layer Security (TLS) security that address the above concerns.

Despite the explained vulnerabilities, the recommendations from security experts, and the existence of newer and better protocols(like SSH), Telnet is still in use today. Many of the devices that still support Telnet are exactly IOT devices. A quick search in the search engine for IOT devices - Shodan[8] reveals that there are more than 160 thousands of active devices running Telnet that are found from the website.

A relatively recent example of exploiting devices using the Telnet protocol is the Mirai Botnet[2]. In the late 2016 worm based botnet called Mirai(Japanese from "The future"') infected more than 600k infected IOT devices by primary using brute-force login attempts to establish a Telnet connection. It initially used 10 username and password pairs selected randomly from a pre-configured list of 62 credentials. Other services that use credential verification were also incorporated to increase the coverage of potential victims. After the first successful login, Mirai infects the device and adds it to a growing number of bots. Those networks were later used for Distributed Denial of Service(DDOS)[19] attacks on more than 600 thousand hosts all over the world.

The example of Mirai and other similar cases, show us that Telnet is still popular protocol in the IOT domain and is suitable to be integrated in our Profiling tool. The insights of it, will be very helpful in future integration of other protocols that have integrated a login phase.

Telnet phases and analysis

In order to figure out how I can incorporate Telnet inside the profile and to what extend, I had to research how the protocol works in details. After capturing the traffic of a Telnet communication with Wireshark[20], analyzing it and verifying my observations with the help of the standard protocol description[16]. I differentiated three phases of the communication. On Fig. 3.7 are presented these stages. I need to analyze every one of them to what extend it should be simulated from the decoy so it supports a medium interaction.

The negotiation phase. This is the first phase where several packets are transmitted between the server and the client. The main purpose of that phase is to elaborate the upcoming way of communication. Both parties are transferring data about themselves and negotiating about different predefined options. Some examples include: terminal size, terminal speed and etc.

This is very important phase, because both the terminal and the server are exchanging information what they need and how the rest of the communication will follow. Based on the selected and verified options, the data that has been transmitted is changing in order to be adapted to the needs of both parties.

Some of the information that has been exchanged can be device specific and can be used in the profile creation. However, this phase is just initial step to reach the actual usage of the service. The big variety of options, sub-options and the combination of them expands the number of request and responses that need to be covered. Evaluating the possible benefits of covering all possibilities does not outweigh the resources and time that have to involved in it. Hence, this phase will not be included in the profile at this initial phase of the project.

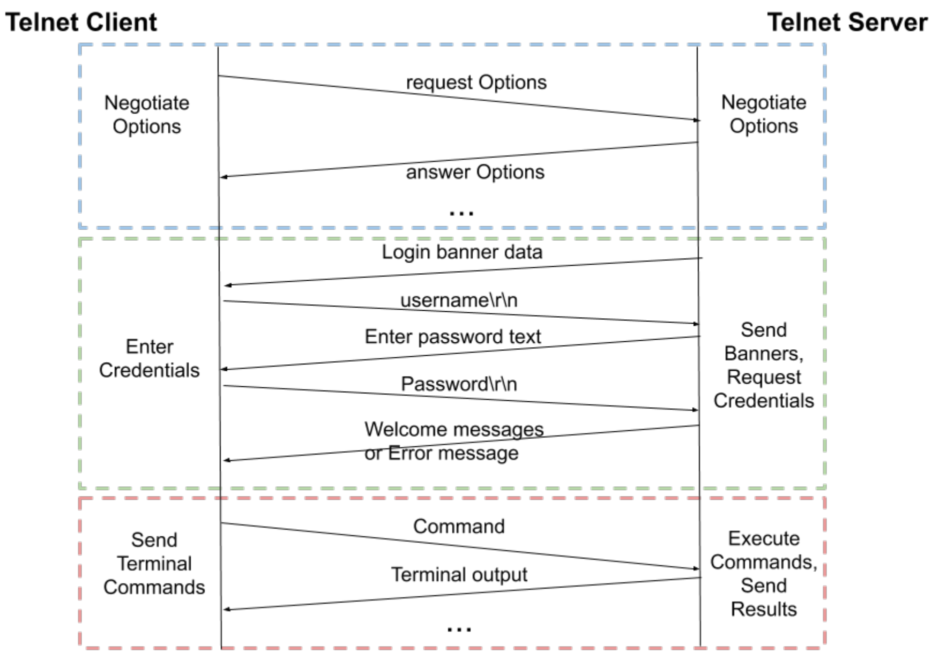


Figure 3.7: Telnet phases

However, in order to reach the second phase, the client or the server will be expecting for the options exchange. Therefore, the profiler and the server should take part of that exchange. Our way to pass to the next phase is by covering minimum interaction by responding to every request of option packet by declining the usage of that option. This approach is also used from different tools that analyze Telnet.

Every new option is located on 3 bytes. The first one is always "\xff" which identifies the beginning of an option data. The second byte defines the option type that is transferred and the last one consists of a code that informs the other party if the current option will be used. To automatically respond to negotiation message containing option data, that byte order will be used. The first byte will be used to identify the end of the first phase, while the third one will be used to automatically respond to a received negotiation request.

The authentication phase. After the negotiation is done, the server requests from the client to authenticate by providing username and password. Most of the penetration testing tools(like Ncrack[21]) operate on this phase. They use predefined lists of usernames and passwords and try to find a pair of correct credentials. This method is called a dictionary attack and is mostly used to identify not secure authentication credentials. Unfortunately this attack is often successful targeting an IOT device. The reason is that the users are not aware they need to change the default passwords embedded from the manufacturer which leaves their device on high risk of unauthorized access.

The described situation is actually the most common attack over running services that require initial authentication. Hence, our generated decoy should be able to respond properly to all these attack requests by keeping all device specific information intact.

The control phase. The second phase ends when the client provides correct username and password and receives welcome messages from the Telnet server. This means that the phase of user verification is successful. The client is then prompted with a terminal connection. They can send every command and perform every operation on the server having the same permissions as the logged in user. Often this user has admin privileges and the attacker receives unlimited control on the device.

Simulating terminal connection is highly complex task and is not integrated in the current state of the project. Covering such communication goes beyond a medium interaction honeypot, but will make it high interaction. Hence, profiling the third phase of Telnet is postponed for future version of the product. The easiest and best performing way to support terminal communication is to install a full Telnet server software on the virtual decoy. It will use the data stored inside the profile to adjust all possible settings and messages so it will keep all device specific information during that communication. Having fully running Telnet server would make that decoy service way more believable and less prone to mistakes. Hence, this is a recommended future direction for Telnet integration and any other service that can be installed and adjusted easily with existing software. Such software is currently available for Telnet and it is supported on all popular Operating systems. An example for Unix based OS is the Telnet daemon(telnetd). However, the steps and requirements for such integration are not further researched in this document due to the company specification that the server should be service independent and the fact that further research is needed to suggest what virtual machine will be used to host the actual decoy.

Integration in the Project.

The main focus of our project will be to profile and to simulate the second Telnet phase - the authentication. In both software components(profiler and server) reaching the authentication phase is a primary goal. To do that the negotiation phase is handled automatically with minimal analysis. It starts when the decoy transmits a don't echo message. This message is required since some tools expect the server to initiate the communication. After that, to every received option request the decoy responds with declining that option.

After all option messages are transmitted, the second phase begins. The main idea in it and the main part of Telnet that is included in the profile is to simulate the User/Password authentication. The profiling tool requests a set of correct username and password from the user and sends them to the server. A correct Telnet communication would result in either receiving a terminal control, or with error messages. The profiler captures the responses received with it sends the correct and some wrong credentials that Telnet server responds with. These messages contain valuable information about the running service and the device it is running on. The messages also are the only data from the device that is visible to an attacker. That is why these messages are forming the fingerprint profile of the scanned device. The only concern that the server needs to handle is to determine when it should return the correct or wrong messages.

The principal that Telnet messages are transmitted is different from the request-response strategy I have used in profiling IPP and HTTP. It is possible to receive multiple requests, before the server responds with one, or more messages. The response that the server should return does not depend only on given parameters in the incoming requests, but also depends on certain set of previous requests. This behaviour creates a sequence of requests that the server should use. It should respond with a given correct messages when the sequence of incoming request are validated and with other error messages when the sequence is different. The main example here are the username and password requests. In order to receive the messages for success login, the user should send the correct username following the correct password.

There is another characteristics of the way Telnet messages are transmitted that complicates profiling it. Based on the options that are negotiated in the first phase, the credentials can be transmitted in different number of packets. For example the username can be send together, or in separate packets letter by letter. Such inconsistencies of the way Telnet works is another reason why request-response strategy used before is not applicable here.

To handle this way of communication and to avoid the necessity to fully implement Telnet server running on the decoy, I use an approach that handles sequences of request and responses. As part of the profile I am saving the result of every sequence I need from the incoming requests. The profile contains other fields that provide information to the server when the sequence should start, when it should end, what information should be compared, what are the correct and wrong data that should be returned and other fields ensuring the correctness of the process

The result of running a decoy that supports Telnet is proper a telnet communication including option negotiation and authentication phases. The client receives correct banners and messages originating from the scanned Telnet server. The correctness and agility of the process are tested with different client softwares and are presented in the evaluation chapter.

Comparison to SSH

Telnet is being around for a long time. Although it is still popular among IOT device, any new system that wants to implement a remote shell access functionality is primary using the Telnet successor - Secure Shell (SSH)[39]. The encryption used by SSH is intended to provide confidentiality and integrity of data over an unsecured network, such as the Internet.

SSH is more complex protocol than Telnet and is used for different purposes. SSH is typically used to log into a remote machine and execute commands, which is the same purpose as Telnet. However SSH also supports tunneling, forwarding TCP ports and X11 connections. It can be used for file transferring using the associated SSH file transfer (SFTP) or secure copy (SCP) protocols. Similarly to Telnet, SSH uses the client-server model.

Another big difference between the two protocols is that SSH(especially version SSH-2) has well-separated layers which should be covered in the binary packets that are exchanged. Due to the involved encryption an SSH server needs to take care of key management, data validation and integrity. Such tasks are important obstacles when service like SSH should be integrated in a profile using the current approach. However, one possible direction for SSH coverage would be to simplify possible usage of the service by selecting just a specific use case(primarily a shell connection to a remote host). Similarly to IPPs and HTTPs a socket wrapper could be used to delegate the encryption task outside of the profile scope and focusing only on the plane communication packets that are exchanged. This would easy the process to identify SSH phases and then select which parts of the whole communication would be covered.

3.3.5 RTMP

What is RTMP

RTMP(Real-Time Messaging Protocol) is a messaging protocol for streaming audio and video files and data between a Flash player and a server. There are different variations of the protocol.(RTMPT - tunnelled to avoid firewalls, RTMPE - encrypted stream, RTMPTE - tunnelled and encrypted). The "plain" protocol works on top of TCP and uses port number 1935 by default. To deliver streams smoothly and transmit as much information as possible, it splits streams into fragments, and their size is negotiated dynamically between the client and server.

RTMP is one the most used protocols for streaming audio and video data. It is primary being used in situations where a secure transport connection is required. Unlike other streaming protocols like RTSP which use UDP, RTMP is running over TCP and it ensures more robust communication of packet losses. Since the current project focuses on TCP services and since most IP cameras have web page(running http server) that can run the camera stream inside a Flash client, I decided that RTMP is a good candidate for analyzing and integrating a streaming service.

Protocol phases

Similarly to TELNET there are 3 stages in a typical RTMP communication[22].

1. **Handshake.** The exchange of 3 packets from each side.: C1,C2,C3(Client messages) and S1,S2,S3(Server messages). The format of these messages is always the same, containing protocol version, date and time information of the request and random data which will be repeated by the other participant.

   From the point of view of device fingerprinting this phase looks straightforward and not valuable. The content of the messages inside is not of a big interest regarding the profiling of the device(except the protocol version that is being used). There is no data that describes a specific device behaviour which should be profiled. The main goal for the profiler and the server would be to successfully reach the next phase.

2. **Connect command.** The client and the server are negotiating some important parameters by exchanging Action Message Format(AMF) encoded messages[23]. This phases usually consists of 2 packets. The Client sends a connect message including parameters about itself and stream parameters needed for the connection. For example: the url address of the streaming server, version of the flash player and more. The server responds with result message describing the connection status and similar parameters regarding server capabilities and stream connection settings. These parameters describe the current status and capabilities of the RTMP server and so, they should be included in the device profile.

3. **Start the stream.** The client sends a "createStream" invocation and receives a result message.

   The next packet is a ping message, or a message defining the buffer size that will be used to transmit audio and video packets. Some of the cameras using RTMP also implement security features by inserting additional payload data to this message. For example it could be a channel number and a transfer of an authentication token. This behaviour is strictly device specific and should be part of the device profile.

   If the previous packets contain data that successfully validates the client(correct channel or token), the server sends several controlling packets and metadata information. After that the server starts sending audio and video data packets with the streaming content.

Important exchanged parameter

Some of the most interesting data fields exchanged in the explained three phases can be differentiated in 3 types of categories[25]:

Network Parameters: These options define how to connect to the media server.

> **socket data** host address and port number

Connection Parameters These options define the content of the RTMP Connect request packet. If correct values are not provided, the media server will reject the connection attempt.

> **app=name** Name of application to connect to on the RTMP server.
>
> **tcUrl=url** URL of the target stream. Defaults url has the following format: rtmp://host[:port]/app.
>
> **pageUrl=url** URL of the web page in which the media was embedded.
>
> **swfUrl=url** URL of the SWF player for the media.

flashVer=version Version of the Flash plugin used to run the SWF player.

Security Parameters These are optional fields that can be required to handle additional authentication requests from the server.

token=key Key for SecureToken response, used if the server requires SecureToken authentication.

jtv=JSON JSON token used by legacy Justin.tv servers.

Analysis and integration in the project

The primary goal of profiling an RTMP server would be to successfully trigger the process of receiving stream data and the other packets of the described communication that contain fingerprinting information.

Due to the inconsistency of the sequence of request and responses in a standard client-server communication, it is very hard to determine what response should be returned to a given request. For example, there can be several following requests from the client without receiving any response. Then the server could respond to each of them and they will arrive also one after another without any request between them. Such race conditions can be explained with either a specific device implementation characteristics or due to network differences. Hence, the request-response approach is not applicable for that service.

After analyzing the way the protocol works and the parameters which should be used to generate the device fingerprint, I have differentiated 3 possible approaches to integrate that service inside the profiler. They posses different complexity and differ in the level of involvement the project should invest for that service, based on which parts of the communication should be covered.

- The first approach is limited on supporting only the first 2 phases of RTMP. Strict protocol implementation on the server for that first 2 phases is possible. Then, the server could use the obtained parameters values from the profile to return valid responses. This is a possible scenario, due to the small amount of packets that need to be covered and also because that exchanged communication is straightforward where every packet has defined format from the protocol specifications[22].
- Another solution is to use external client or server software that supports RTMP streaming. One popular project that could be used as a client is a VLC player. Another option would be to integrate a RTMP library inside the profiler. With the help of such software I will be able to mimic the first and second phases without the need to manually implement these protocol specifications. However, any security parameters that could be integrated inside the client packets will not be presented using this approach. For example it will be missing security token, which will be observed from the original RTMP server. Hence, the client will not receive any streaming data and the third phase could not be handled and included in the profile.
- The third approach is the most complicated and with the highest protocol coverage. It requires capturing the raw requests and responses between the original camera client and the RTMP server. To successfully capture the performed request from the original client I will integrate the listening phase in the profiler. This scenario has already been presented and it includes sniffing the traffic between the device and the host computer. It creates the possibility that the user can integrate any functionality they wish as part of the profile. The main advantage of using this approach is that all important parameters in the packets will be captured, since the original client software is being used. When the decoy is generated, the responses it returns will contain these parameters, which will increase the similarity with the original responses and will successfully mimic the behavior of the real RTMP server. For some of the previously researched

service(like TELNET) I assumed that the user knows security fields like username and password, because they are required to obtain every data where the user should authenticate. However, the possibility to use tokens and the lack of knowledge how they are calculated removes the variant to ask the user for valid authentication data. Therefore, in order to reach the third phase(streaming) during profiling, I should sniff the traffic when the user is using the device and integrate that communication inside the profile.

Although it requires user interaction with the original client software, the approach that I have integrated is the third one. The approach requires that the decoy should be able to determine when a given request is matching the current request of the sequence based on its content. If the request is valid it should be returned from the server. The implemented steps are:

1. The profiler sniffs the traffic between the client and the server. I collects all requests that have been performed from the server.

   The implemented procedure in the RTMP Manager of the profiler is made abstract from the client that needs to be scanned. It is a general approach that scans RTMP without having any knowledge of the specific client implementation. The Manager listens to the traffic and saves all RTMP packets that have been transmitted between the client and the server. At this step no further actions are performed. The only goal is to obtain the packets which contain the parameters needed to start the actual stream.

   Sniffing the traffic requires that the correct commands to start the stream are performed. For HTTP this additional commands were restricted to load the index page. This process is identical for every HTTP server and it helped us automate it. However, it is not possible to do the same with a RTMP stream, because of the existence of security parameters in the RTMP packets. The user is not familiar how the used token is generated and the correlation with the authentication credentials is unknown. Therefore, our general approach assumes that the proper RTMP traffic will be triggered directly from the client by using the provided software from the camera manufacturer.

   Cybertrap has close relations with their clients. Many times the distributed product is adapted to the specific needs of that client. Hence, it will be possible to understand how the used RTMP client software work. In case there is a web page for that purpose, I can automate the process of triggering the RTMP traffic by using a testing software like a selenium webdriver[26]. It give us the possibility to perform every action the user needs to do in order to start the camera stream. Hence, that process could be used for every device that uses the same client software. For example all devices of a given manufacturer that use the same software would be automatically controlled when the necessary steps are saved once. It will make the process easier eliminating the need of the user to interact with the camera directly.

2. In the second step, the profiler will perform every of the stored request with some timeout between them. Doing so, I avoid any race conditions and ensure that the sequence of requests and responses is valid. When these requests are performed for a second time it is easy to determine when the server is returning the same responses. In the situations when the server is not responding or returns given error I can determine which information from the requests should be modified. Then, I adapt the information in the profile accordingly to the changes that should be performed for every request.

3. All received responses will be matched to the last performed request from the sequence and they will be saved inside the profile. When the responses are received after the final request from the sequence, they will actually be the streaming data of audio and video packets.

4. For every captured and performed request I determine if it is an important request of the protocol that should be saved in the profile. Requests which results without any received responses, may

be with less significance for the sequence and are marked as optional. The other requests are marked as important and for them some specific data is assigned that would help the server identify when a new request is valid and from the same type. The format of the profile that represents RTMP looks like this:

```
RTMP: [
{
        requestData: "Some data for the given request",
        validation:{
        listSameType:[
            {...}],
        listIsValid:[
            {... For example: token = 123456}]},
        responses:[
           {data:"data of the response", delay: 1234},
                  ...]
},
...
]
```

5. The information inside the profile will be used from the server to mimic the behaviour of the RTMP server. The analysis when a given incoming request is found in the profile and if it is valid or not, is handled from the server. When a new request is received, it is analyzed and compared with correct request from the stored sequence that should be following. Using the fields from the profile, the decoy concludes if the request is of the same type and if it is valid. When both requirements are satisfied, the decoy returns the stored responses for the current request in the sequence. Then, the current request is updated accordingly.

   Since this request verification is done on the server, it gives us the possibility to dynamically update the result of that analysis and the responses that should be returned. An overlook of the performed server verification is shown on Fig.3.8

The described approach for RTMP is very similar to the one used in HTTP. However, there are two important changes that make it possible to use this scenario with RTMP. First, the sequence of the stored requests is critical for the normal service functioning, and it should be followed from the decoy. Second, there could be multiple responses to a given request. This is critical foundation for every streaming protocol, because once the configurations are completed and the necessary commands are set, there are no more requests from the client, while the server is sending multiple video or audio packets. These two changes are determining which approach will be used to profile a service. Hence, a general solution that incorporates all specification could be used for all services.

3.4 Summary of the services

Besides the non service specific information that is stored in the profile, I have analyzed and integrated four application level services. These are: IPP, HTTP, Telnet and RTMP. They have been selected based on the three main components:

- primary interesting IOT devices for Cybertrap
- popularity and usage of the given service
- diversity in the analyzed services so more knowledge and approaches are covered

For every one of the selected services, I have analyzed the way they work, what information is important to identify the host running the service and how this information can be extracted and

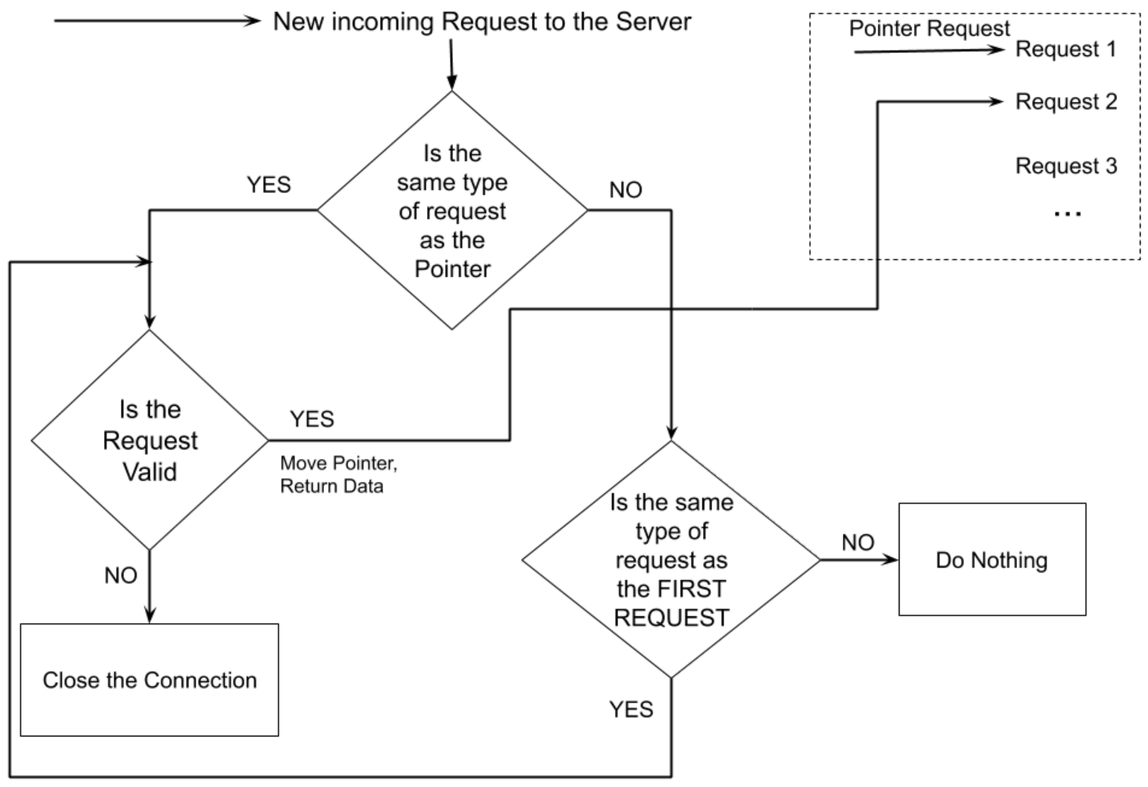

New incoming Request to the Server
Pointer Request
Request 1
Request 2
Request 3
...
Is the same type of request as the Pointer
YES
NO
Is the Request Valid
YES
Move Pointer, Return Data
NO
Close the Connection
Is the same type of request as the FIRST REQUEST
NO
Do Nothing
YES

Figure 3.8: RTMP Server verification procedure

saved in a profile, so that profile can be used from the server to generate a decoy of the scanned device.

Based on every analyzed service characteristics, I have adapted an approach which best suits the service characteristics and is used to be profiled. The level of involvement in every service differs based on which information is interesting for our medium interaction honeypot and how complicated it will be to cover specific feature inside the profile. All of the used approaches are designed to be not service dependant, but so they can be adapted from any other service that works in similar manner. This multi-service adaptation is also valid for the analyzed services and some of them are using the same approach. For example both IPP and HTTP use a request-response based solution.

Another important aspect of the integration of a given service is how to obtain the required information that will be stored with some of the approaches explained above. The way that data can be received can be divided in four groups:

automatic Automatically replying to a given request with the goal to reach another level of the communication. This method usually ignores some parameters and could lead to potential restrictions with the profiled behaviour. It is used in phases of the service which are not interesting for profiling.

static This is highly service specific data that consists of several specifically crafted packets that are stored in the Manager class for that given service. This packets are usually a set of messages that the server is expecting and they create big part of the service coverage we focus to profile.

semi-dynamic This approach is using service specific way to start a communication and later on it receives additional data for that service. The new data is used to obtain more valuable insights of which requests should be used to properly profile that service.

dynamic By using using external tools I identify potential important behaviour that are scanned and

Table 3.4: Summary of used approaches for every analyzed service

Methods\Service	IPP	HTTP	Telnet	RTMP
Approach to save the data in the profile	**request-response** For every request the corresponding response without considering request dependencies	**request-response** For every request the corresponding response without considering request dependencies	**sequence-end** Covering the authentication phase by saving the sequence of responses and the end result when the sequence should end	**request-sequence** For every important request the list of received responses is saved. The order is important. The request type is analyzed.
Approach to obtain important request	**static** Saving all important IPP type of request inside the Profiler	**semi-dynamic** Obtaining all request needed to load the start page **dynamic** Saving all requests performed from a tool that does not return given error	**automatic** Automatically replying to negotiation packets so the next phase can be reached **static** Performing the standard Telnet communication by saving them in the Profiler	**semi-dynamic** Passively finding the original performed request Triggering the original request is either done from the user, or automated for the given device

included in the profile. This approach help us to cover different attack vectors which are specific for the scanned service and which would be investigated from an attacker.

On Table.3.4 is shown a summary of the four covered services and which approaches are used to obtain their proper fingerprinting.

Chapter 4

Evaluation

The analyzed services have been integrated inside the profile. I have used several approaches to adapt and capture the specifications of ever service and at the same time trying to make these approaches as general as possible. Since I am targeting a medium interaction honeypot, the services are not cloned fully covering all their functionalities. Based on the research findings I focused on covering the most significant information and simulating the most valuable and used part of the protocol. For some phases of a proper service work, the complexity to cover given task would increase the global complexity multiple times. Such task for example is covering terminal control in Telnet.

Based on the approach I used and the level of integration that have been implemented the usability of the generated decoy will vary. To reach maximum amount of usability and to verify the correctness of the results I used several methods. In this chapter I will present all the methods I have used. I will look why they are valuable and how their results should be considered. After that for all of the integrated services I will explain which of these approaches are applicable. Then I will run these tests on several devices I have full control with. Some of these approaches are also applicable with limited access to the device. After the tests are performed and the results are presented, I will analyze them. I will also compare the results and the approaches with several partial or full competitors.

4.1 Experiments

I will look at four methods that have been used to test the correctness and completeness of the created profile and decoy. These methods thy to address different usage scenarios so I can obtain full impression from the decoy. They tackle scenarios from the normal usage of the device, automated tools used from hackers and some specific attacks for a given service.

4.1.1 Simulating normal usage of the service

The first method I will use to evaluate if the generated profile is sufficient and if the generated decoy is working properly is most intuitive. It contains simulating a normal usage scenario for that device and the corresponding services. This approach is based on our informed understanding of the original device. Since I am aware of what is the device that have been profiled, I know how it works and what it requires to run it properly. Then I manually interact with the created decoy and compare the behaviour of the original device to the one from the decoy.

Since this method is based on the targeted device it is difficult to be explained in a general scenario. The actions that are taken can use different tools or client software in order to properly interact with the given service. This approach is focused on the perspective of the user. I assume that in order for a given honeypot to successfully fool an attacker(even for some time) I need to be able to successfully fool a normal user. The approach will help us identify what are the features and commands that the decoy is capable of handling. It will also give us clear view for limitations or errors in the process. Hence, it will be easier to spot if there are other features that are not properly handled, but I want them to be inserted. Thanks to the dynamic approaches that are integrated in the profiling step, I can trigger new scan that will generate better profiler that covers these desired features.

4.1.2 Profiling the real device(ground truth) and the generated decoy

The main idea behind the profiling phase is to capture the identifying data for the scanned device and store it in useful format(the profile). Furthermore, the main idea of the server is to use the data stored in the profile to repeat that identifying data when necessary. Looking to evaluate the work of the decoy I will use the profiler again, scanning the decoy itself. If the decoy is implemented properly it should respond to the profiling request from the profiler the same way as the original device. Hence, I should be able to generate a new profile(market with F) which should be very similar to the one generated from the original device(market with O). This process could be seen as making a copy of the copy. On Fig. 4.1 we see that process which leads to comparing the two generated profiles.

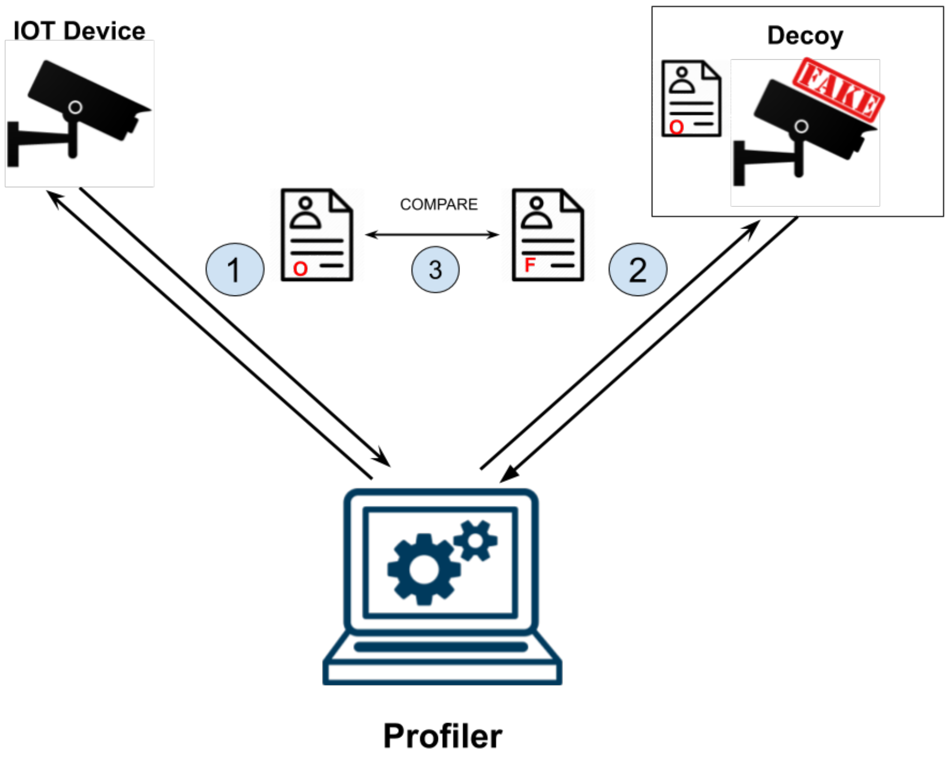


Figure 4.1: Evaluation method to compare the created profiles from the original device and the decoy

4.1.3 Compare the results of different tools on the real device and several variations of the decoy.

Except interacting with the device the attackers are using variety of tools to help them breach that device. These tools can be used to identify which is the tested device or can be directly searching for vulnerability by sending maliciously crafted request. Therefore, the next evaluation technique I have performed is using such tools to analyze the original device and the generated decoy. The outputs of that tools should be similar in way that they represent the same vulnerabilities if presented, the same information that could be gathered and the same behaviour for some specific actions.

Since most of the tools for scanning are focused on a specific service, or a small set of services, I will look at the analyzed services individually. For all of them I will present which tools I have used for the evaluation(if any) and will compare the results from the original and virtual devices.

For some of the analyzed services I have integrated the approach to run a reverse proxy and re-transmit every request to the original device. This approach give us the possibility to run any tool and use the results inside the profile. This approach corresponds to the evaluation technique described here. Therefore, there could be the situation where the same tool is used for both the profiling and the evaluation of the results. Such case implies that the results of the tool during the evaluation should be definitely part of the profile and the tool outputs should be very similar as the results to the original device. Hence, for the services that use the reverse proxy for profiling I will present comparison between these three type of sources: the original device, the decoy when the same tools is used in the profiling and the decoy when that tool is not used.

4.1.4 Tests for specific problematic attacks for a given service

Although there are numerous tools for scanning and testing the results of them are not always focusing and explaining every attack scenario that could be interesting for that specific service. There are a couple of reasons for that. First, the tools I have used are not focusing on that specific attacks and so the results they output do not contain information about them. Second, many of these tools use techniques with thousands of performed request and their output is designed to show result only when a vulnerability is discovered. However it is still possible that there is not vulnerability presented, but the decoy is responding differently than the original. Due to these situations I have integrated another approach for some services where I will perform some crafted requests to observe the results and to identify if a given attack scenario has the same results in both the original and virtual devices.

4.2 Devices

During the performed research I have used several devices that I have been provided from Cybertrap. I will address these devices as the local devices. For them I have full access and the knowledge of any credentials, security parameters or any device specific information. For most of the analyzed service, this information is crucial to obtain a proper profile. Without it, it would not be possible to reach the point of the device work where the most significant functionality is reached. In the majority of the attacks this information is the primary goal of the hacker and it is what they want to reach. Hence, without having full access on the device, I would not be able to support these attacks from the decoy, or at least not to the amount that would be possible other way.

These local devices are necessary to fully test the desired functionalities. However, the short number of local devices that I have been provided with is not sufficient to generalize the desired approach and to fully test the behaviour of the decoy knowing that there is no such device specific information that have been used to design the profiling approach. Therefore, I will use other devices which I have limited access to in order to observe the behaviour of the profile and the decoy in these restricted situation.

4.2.1 Local Devices

The local available devices used for the research and the evaluation tasks are based on the main interests of Cybertrap for first prototype goals. These devices were mostly printers and cameras. The list of local devices consisted of:

- Printer. HP Color LaserJet MFP M477fdw
- Printer. TASKalfa 2552ci. Host Name : KM647584
- IP Camera. Reolink. 5mp PoE Cam RLC-41005mp

On Table.4.1 We can see which of the analyzed services were available from the local devices.

As we can see the see in Table4.1, there are no available local devices that run a Telnet service. In order to implement that service I have been running a Telnet server on a virtual machine with specifications:

Table 4.1: Service coverage in local devices

Service \Device	Printer HP	Printer TASKalfa	IP Camera Reolink
IPP	✓	✓	✗
HTTP	✓	✓	✓
Telnet	✗	✗	✗
RTMP	✗	✗	✓

OS: Kali Linux
xinetd version: 2.3.15.3 libwrap loadavg

4.2.2 Other devices

Tto avoid the limitations of having a short number of tested devices I have used the profiler to scan multiple other devices available over the Internet. I have used the shodan website[27] to find such devices that run the desired services. It is a search engine for IOT devices that makes it easy to find any running services based on the open ports of the collected hosts.

4.3 Expectations

Before looking at the actual results I would state what are the expectations that I have for the performance of the profiler and the decoy for every of the executed experiments. The results would be compared with these initial expectations to determine whether the project successfully mimics the scanned device or not. These comparison would be useful to determine what are the advantages and disadvantages of the project and respectively which components should be improved for the next future versions of the profiler project.

The starting expectations for the four main types of experiments can summarized as:

Simulating normal usage of the service The generated decoy should be able to mimic the most important features of the real device. This means that the decoy should reply in as similar as possible way to any interaction with it, that is considered as a standard and crucial communication. That communication could be performed from different tools and applications that are specifically targeting the normal usage of the tested service. A successful decoy responses would be anything that handles proper interaction with that software and which makes that software "believe" in the correctness of the responses.

Profiling the real device(ground truth) and the generated decoy This experiment tests both the correctness of the decoy responses and the consistency of the profiler scanning. Since the decoy responses are based on replaying the data stored in the profile, the profiler should be able to create almost the same profile when the decoy is being scanned. The only differences should be based on the differences that are updated from the server before the response is returned. The comparison between the two profiles could include the number of stored requests and responses, the data inside and a check if the differences are properly updated. If there is a significant difference between the two profiles, this means that there is problem when the decoy is replaying the data, or the profiler is not performing the same scanning requests.

Compare the results of different tools on the real device and on the decoy The penetration testing tools are the most common automatic way that an attacker is scanning a given unknown device. These tools output results should be as close as possible to the output results obtained when the tool is executed on the decoy. Any significant differences suggest that the decoy is not able to replicate the responses of the actual device. The reason could be that these responses are not presented in the profile at all, or they are not properly matched or updated.

Since the profiler could include dynamically obtaining of important requests, some of these penetration testing tools can actually be used during profiling. The following results when the decoy is scanned, should be identical as the one when the original device is scanned(assuming the the tool has deterministic behaviour). When the tool is not used for profiling, I expect that the results would be limited, but they should still be valid and covering at least the base device behaviour.

Another important characteristics that should be compared is the time needed for these tools to complete their scanning. The results will show if there is significant delay of the decoy responses which will be a result of slower analysis on the profile. It is also possible that the decoy is replaying faster than the original device which would also help an attacker to identify that the scanning device is actually not a proper IOT device. Hence the expected time for execution should be as close as possible to the average time needed to scan the IOT device.

Tests for specific problematic attacks for a given service The deeper analysis of the integrated services let us find out how their results could be used to identify the scanned device. Since most of the tools are working as a black box it is difficult to find out if these requests have been performed and to properly analyze the results. Hence, running these services manually on the decoy would reveal if these attack vectors are covered from our profiler. These tests are primarily focused not on the the decoy correctness, but on the extended work of the scanning phase. The results would help us evaluate the fullness of the generated profile and if additional requests should be included in the profiling.

4.4 Results

In this subsection I will present all of the analyzed services individually. I will explain which of the evaluation experiments are suitable for that service and which can not be used. Then I will look at the results from the performed tests and I will discuss their correctness by comparing them with the initial expectations.

4.4.1 IPP

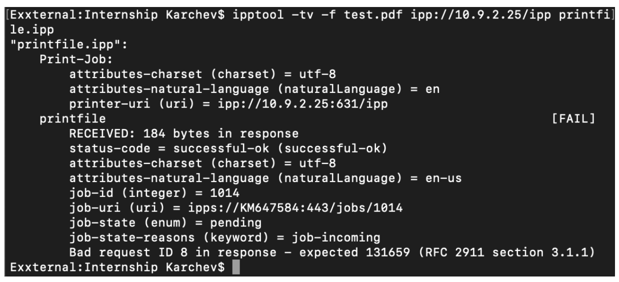
Not all proposed type of experiments can be executed for IPP. I will focus the evaluation on the first two scenarios:

- Simulating normal usage of the service
- Compare the Profiles of the real device and the generated decoy

The other 2 types of experiments are not presented, because at the moment of writing this document there are no available tools for testing IPP protocol that I am aware of. Furthermore, no other attack vectors are known that can not be instrumented as part of the simulating of the device. Therefore, I believe proving that the normal behaviour of the service compared to the behaviour of the original device will be sufficient test for the correctness of the IPP service decoy.

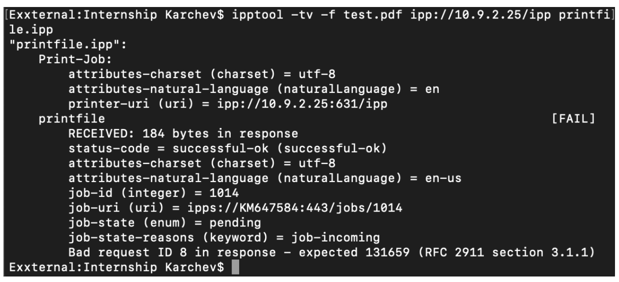
Simulating normal usage of the service

ipptool The first way I will evaluate the created printer profile for IPP and the generated decoy is by triggering a print command from a tool called ipptool[28]. It is used to perform internet printing protocol requests. The tool uses a text file that defines one or more requests, including



```
[Exxternal:Internship Karchev$ ipptool -tv -f test.pdf ipp://10.9.2.25/ipp printfi]
le.ipp
"printfile.ipp":
    Print-Job:
        attributes-charset (charset) = utf-8
        attributes-natural-language (naturalLanguage) = en
        printer-uri (uri) = ipp://10.9.2.25:631/ipp
    printfile                                                              [FAIL]
        RECEIVED: 184 bytes in response
        status-code = successful-ok (successful-ok)
        attributes-charset (charset) = utf-8
        attributes-natural-language (naturalLanguage) = en-us
        job-id (integer) = 1014
        job-uri (uri) = ipps://KM647584:443/jobs/1014
        job-state (enum) = pending
        job-state-reasons (keyword) = job-incoming
        Bad request ID 8 in response - expected 131659 (RFC 2911 section 3.1.1)
Exxternal:Internship Karchev$
```

Figure 4.2: ipptool failed operation output



```
[Exxternal:Internship Karchev$ ipptool -tv -f test.pdf ipp://10.9.2.25/ipp printfi]
le.ipp
"printfile.ipp":
    Print-Job:
        attributes-charset (charset) = utf-8
        attributes-natural-language (naturalLanguage) = en
        printer-uri (uri) = ipp://10.9.2.25:631/ipp
    printfile                                                              [FAIL]
        RECEIVED: 184 bytes in response
        status-code = successful-ok (successful-ok)
        attributes-charset (charset) = utf-8
        attributes-natural-language (naturalLanguage) = en-us
        job-id (integer) = 1014
        job-uri (uri) = ipps://KM647584:443/jobs/1014
        job-state (enum) = pending
        job-state-reasons (keyword) = job-incoming
        Bad request ID 8 in response - expected 131659 (RFC 2911 section 3.1.1)
Exxternal:Internship Karchev$
```

Figure 4.3: ipptool successful operation output

the expected response status, attributes, and values.

I created such test file containing a print-job request which executed on the original device successfully prints the provided document.

The content of the file looks like this:

```
 {
    OPERATION Print-Job
    GROUP operation-attributes-tag
    ATTR charset attributes-charset utf-8
    ATTR language attributes-natural-language en
    ATTR uri printer-uri $uri
    FILE $filename
}
```

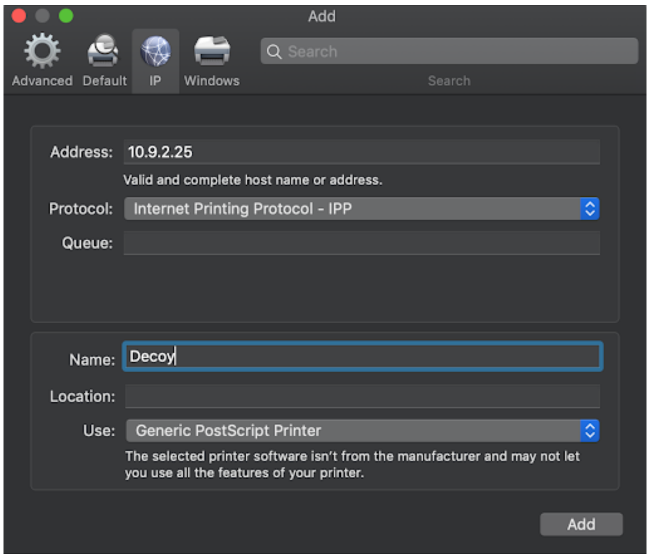
I also used a test pdf file(called test.pdf) which will be transmitted for printing. The terminal command to run the tool is:
ipptool -tv -f test.pdf ipp://10.9.2.25/ipp printfile.ipp
where:

- -tv declares the desired output format,
- -f test.pdf is the file we want to print
- ipp://10.9.2.25/ipp is the decoy printer-uri.(10.9.2.25 is the local ip address of the host machine running the decoy.)
- printfile.ipp is the file containing the print-job request shown above.

On Fig. 4.3 we can see the output result of ipptool executed on our decoy.
I have used ipptool during my research and implementation to help me identify what is needed to successfully simulate IPP. It gave me valuable insights of how to support a given request and what other modifications are needed. For example, before changing the request ID stored in the response inside the profile, with the correct one coming from the new request, ipptool was informing for a failed execution due to mismatch in the request ID. On Fig.4.2 We can see the result of running ipptool on our decoy, before updating the request ID in the returned response.

The result of running ipptool over all remaining ipp addresses is similar and returns successful PASS output status. Here I have to mention that based on the created profile data, some of the created decoys returned other messages besides 0 (successful-ok). Such message for example is 1287 (server-error-busy). This operation result is based on the current situation of the printer when the profile is created. However, the goal of the profiler is to capture such specific outputs of the time of the scanning. Hence, receiving messages of type 1287 (server-error-busy) is considered as correct result and it is also marked from ipptool with PASS status.

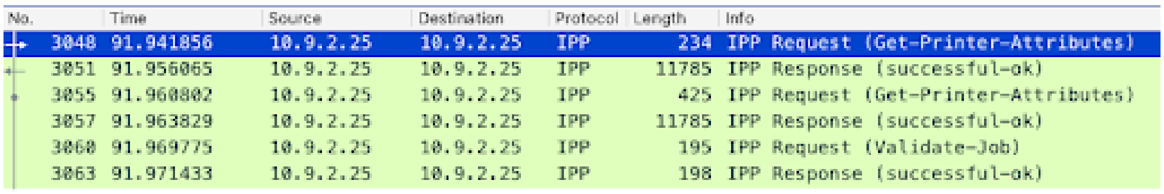
Printers and Scanners The second method I used to test the correct responsiveness of the ipp decoy is by using the integrated software to work with printers in a MacOS running machine(macOS



Address: 10.9.2.25
Valid and complete host name or address.
Protocol: Internet Printing Protocol - IPP
Queue:

Name: Decoy
Location:
Use: Generic PostScript Printer
The selected printer software isn't from the manufacturer and may not let you use all the features of your printer.

Figure 4.4: Standard dialog to Add new printer in Mac OS



```
No.   Time       Source     Destination  Protocol Length  Info
3048  91.941856  10.9.2.25  10.9.2.25    IPP        234   IPP Request (Get-Printer-Attributes)
3051  91.956065  10.9.2.25  10.9.2.25    IPP      11785   IPP Response (successful-ok)
3055  91.960802  10.9.2.25  10.9.2.25    IPP        425   IPP Request (Get-Printer-Attributes)
3057  91.963829  10.9.2.25  10.9.2.25    IPP      11785   IPP Response (successful-ok)
3060  91.969775  10.9.2.25  10.9.2.25    IPP        195   IPP Request (Validate-Job)
3063  91.971433  10.9.2.25  10.9.2.25    IPP        198   IPP Response (successful-ok)
```

Figure 4.5: Wireshark IPP traffic output when adding new printer

Mojave Version 10.14). The software is available in System Preferences - PrintersScanners. It gives all functionalities to a user to integrate and control a printer device.

Two steps are needed to test the Decoy:

1. Adding the decoy as a new printer.
2. Sending a file for print.

Adding the decoy as a new printer. In the provided window shown in Fig.4.4 I have inserted the ip address of the decoy and selected IPP as the protocol to use.

Pressing the button Add, sends several IPP messages to our decoy. The captured traffic is shown on Fig.4.5:

As we can see on Fig. 4.5 all requests send to the decoy return response with status successful-ok. Overall, the process of adding the decoy as new printer is successful and it is now available in the list of printers. Shown in Fig.4.6

For the second step I opened the print queue of the decoy and I sent a file for printing. That step resolved in the following requests that have been sent to the Decoy:

- Get-Printer-Attributes
- Validate-Job
- Create-Job
- Get-Job-Attributes

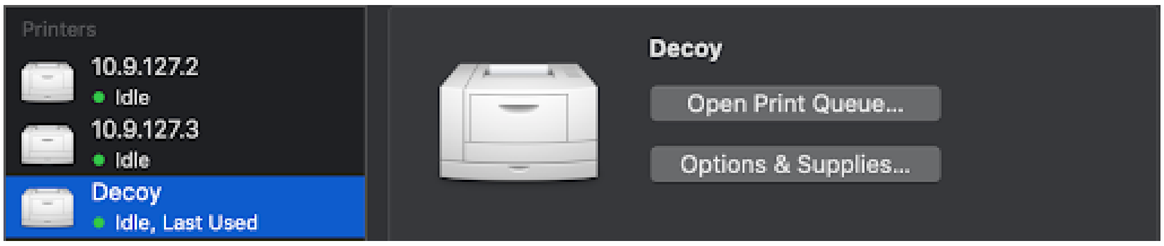


Figure 4.6: List of available Printers included in the system, including our Decoy.

- Send-Document

The last request is Send-Document, where the actual file for printing is being transmitted. Based on the created profile, the decoy returns successful-ok message, or other status like server-error-busy or client-error-bad-request.

The results illustrate that our decoy successfully mimics the original printer by being able to properly interact with the ipptool and the Printers and Scanners software of the Mac OS, which I have used. I am aware that when a specifically crafted requests are performed with parameters that would lead to a given error inside the original device, would be considered as a valid request at the current version of the server. However, the approach used to match and update requests could be easily adapted to such scenario and these requests would be covered properly. For the current goals of the project it is important and successfully verified that the decoy is able to perform as the original printer for the most common use cases.

Compare the Profiles of the real device and the generated decoy As I have already explained the IPP protocol consists of a limited number of request types that are possible during a communication between a client and a server. I have selected 11 requests to be part of the profiler which I use to obtain the printer responses. Hence, for a successful scanning, the profile should consists of all these 11 requests and the corresponding data about them.

By running another scanning on the generated decoy I can answer to several questions about the correctness of the approach I am using:

- Verify that the responses are correctly matched to a given request.
- Verify that the Server correctly identifies the incoming request and matches it to the correct request in the profile.
- Verify that the procedure to update differences in the stored responses works properly.
- Verify that the procedure to compact the http and ipp data in one response works correctly.
- Identify new missing differences that need to be inserted, or any other incorrectness of the results.

On Table.4.2 are presented the results of the scanning on both the real device and the decoy. For all of the available printer addresses I run the profiling 10 times and compared the results.
From the results on Table.4.2 I can conclude that the created profiles of the generated decoy are successfully matching the original device responses and have been properly modified for the differences that are specified in the profile itself. I have been able to identify other potential differences that could be supported in the profile requests, which will improve the validity of the information returned inside the responses.

Table 4.2: Comparison of IPP profiles for original and decoy devices

Test \Device	Printer 1 ipp original	Printer 1 ipp decoy	Printer 1 ipps original	Printer 1 ipps decoy	Printer 2 ipps original	Printer 2 ipps decoy
number of request	11	11	11	11	11	11
responses with status code successful-ok	4	4	4	4	5	5
matching responses	-	11/11	-	11/11	-	11/11
correctly updated requestIDs	-	11	-	11	-	11

4.4.2 HTTP

HTTP is the most popular service around IOT devices. The big variety of devices and usage scenarios give us the possibility to evaluate our approach using all described experiments:

Simulating the behaviour of the service The most common way of using HTTP is by having a web browser as a client that connect to a HTTP server. During the profiling phase I have spend specific attention on this usage scenario by capturing and repeating all of the request that are performed when the device IP address is loaded. After the profile was created and the decoy was executed I have used the server address to load it in a browser and compare the output to the original content of the website from the scanned device. The results showed that the website was successfully loaded by being identical to the original content. I am aware that any other further interaction with that website, which will result in performing new requests which are not part of the profile, would lead to missing file or a content. The decision how extensive a site need to be cloned and which requests should be covered can be postponed to a future step, because of the dynamic approach used for capturing requests from the device interaction or a given tool. The results of them are presented next and would show us how effective that approach is.

Compare the results of penetration testing tools on the real device and the decoy There are many different tools which are used to scan the http service. I have performed several scans after the decoy is created. I have compared the results with the obtained when scanning the original device. The comparison has been done on both scenarios: when that tools has been used during the profiling phase and when it has not been used.

The first test I have performed is using a nmap script called http-enum[7]. It is used to enumerates directories used by popular web applications and servers. The script uses advanced pattern matching as well as having the ability to identify specific versions of Web applications. The typical output of that script shows all found pages that result with return status 200 OK or 401 Authentication Required. The terminal command to start this script is:
nmap localhost -sV –script=http-enum -p 80
where:

- localhost is the address of the target. When we are running the decoy on our local machine, we can use localhost.(localhost = 10.9.2.25)
- -sV is nmap flag that probes open ports to determine service/version info
- –script=http-enum is the script name I want to execute
- -p 80 specifies the http port number that I want to restrict the scan on.

Table 4.3: Outputs of running http-enum and Nikto over every profile

Tested device \ Performed Test	Number of results found by http-enum	Time to load http-enum	Number of results found by Nikto	Time to load Nikto
Original	26	20.43 seconds	53	53 seconds
Profile 1	1	11.28 seconds	8	22 seconds
Profile 2	26	11.91 seconds	15	32 seconds
Profile 3	9	15.54 seconds	53	38 seconds
Profile 4	26	16.03 seconds	53	39 seconds

The second tool I have used for testing is Nikto[7]. It is an Open Source web server scanner which performs comprehensive tests against web servers for multiple items, including over 3500 potentially dangerous files/CGIs, versions on over 900 servers, and version specific problems on over 250 servers. The terminal command I used to execute the scan is:
nikto -p 80 -host localhost
where:

- -host localhost is the address of the target. As explained with the nmap script, I am am targeting the decoy on the local machine,so I can use localhost.
- -p 80 is specifying the http port number that we want to restrict the scan on.

Due to the possibility to use these tools in the profiling phase, I will present the results of nmap http-enum script and Nikto on 5 different profiles for every device. These profiles are a way for differentiation whether some of the tools are used during the profiling or not. Hence, the five different profiles are:

Original: The original device

Profile 1: Profile of the original device without using any tool in the scanning.

Profile 2: Profile of the original device using nmap http-enum script in the scanning.

Profile 3: Profile of the original device using Nikto in the scanning.

Profile 4: Profile of the original device using both http-enum and Nikto in the scanning.

On Table:4.4 are presented the output results of running nmap http-enum for every of the profiles generated for one of the tested devices(10.9.1.1 camera) including the time needed to be finished. The reason that this device is used is due to some specific interesting behaviour which will show us if they are transmitted to the decoy behaviour. First, for every .cgi request the device requires authentication even if there is no such file and second, for every request that contains the word ”stat” it returns a statistics page. Hence, this unusual behaviour could help us observe the results of the different tools over our decoys.

As we can see from the results of running http-enum script, the decoy using profile 1, is returning the server version, but the other device specific information is missing. The decoys using profile 2 and profile 4, where http-enum was used during profiling, successfully return the exact output as the original device. The decoy using profile 3 performs better than profile 1(both are not using http-enum during profiling) is able to return some of the outputs regarding the requests which contain ”stat”.

Table 4.4: Scanning data for HTTP service profiles

Scan type \Device	Device 1(10.9.1.1)	Device 2(10.9.127.2)	Device 3(10.9.127.3)
Profile 1	82	15	27
Profile 2	128	29	40
Profile 3	348	327	318
Profile 395	341	15	334

The time to perform the test is slightly increasing with every decoy. The reason for this is probably due to the sizes of the profiles and the number of requests in each of them. The responses from the original device are a bit slower than all responses from decoys, since it is located on different machine and the additional delay comes from network operations.

We can observe similar results from running Nikto. The decoy using profile 1 results with correct output for all tests regarding the index page and the http headers that are transmitted. The decoy of profile 2 has additional results that are found since some of the requests are performed also from http-enum during profiling. The decoys with profile 3 and profile 4 return the same results as the original device. The reason for this complete match is that the Nitko script was also used during profiling and the same requests were stored in the profile. Similarly to the results of http-enum, Nikto takes slightly more time with every following decoy caused from the bigger profile that is handled. Nikto needs more time to finish its scan on the original device. This delay is caused from the fact that unlike the decoys, the original device is not located on the same host machine. Hence there is a transmission time which multiplied by the number of performed requests increases the time of the full scan.

All other devices running HTTP have similar results to the one presented for the camera running on 10.9.1.1. The following conclusions are confirmed about the work of our decoys when tested with penetration testing tools:

- Profile 1 successfully loads the index page in a browser.
- Profile 1 is not capable to simulate deeper analysis from any tool.
- When a given tool is used during profiling, then the decoy is producing identical results as the original device.
- Increasing the number of requests inside the profile improves the results even when different tools are used for testing.
- Increasing the number of requests inside the profile increases the time for response and so for the tool to complete its scan.

Compare the Profiles of the real device and the generated decoy Unlike IPP where there are limited number of requests which could simulate big part of the service usage, HTTP does not have such standard requests. Hence, covering as many valid requests as possible would increase the similarity with the original website. On Table.4.4 we can see the results of comparing the profiles across several observed devices that run HTTP for every of the five profiles. These comparison covers the number of requests inside every of these profiles and the time needed to complete the tools scans

Testing additional attack vectors which are not covered from the used tools

Table 4.5: Comparison of returned HTTP headers order

	Device 1 Original	Device 1 Decoy	Device 2 Original	Device 2 Decoy	Device 3 Original	Device 3 Decoy
Result	HTTP/1.1 200 OK Server Date Content-Type Content-Length Last-Modified Connection E-Tag Accept-Ranges	HTTP/1.1 200 OK Server Date Last-Modified Connections E-Tag Accept-Ranges	HTTP/1.1 200 OK Server X-Frame-Options X-Content-Type-Options Cache-Control Content-Type Expires	HTTP/1.1 200 OK Server X-Frame-Options X-Content-Type-Options Cache-Control Content-Type Expires	HTTP/1.1 400 Bad Request Content-Length Content-Type Upgrade Accept-Encoding	HTTP/1.1 200 OK Content-Length Accept-Encoding Date Cache-Control Expires Set-Cookies

Table 4.6: Comparison responses to forbidden operation

	Device 1 Original	Device 1 Decoy	Device 2 Original	Device 2 Decoy	Device 3 Original	Device 3 Decoy
Result	HTTP/1.1 405 Not Allowed	HTTP/1.1 200 OK	HTTP/1.1 400 Bad Request	HTTP/1.1 200 OK	HTTP/1.1 400 Bad Request	HTTP/1.1 200 OK

Since all of the stored request have been obtained dynamically(by either sniffing which requests are used to login the index page, or by capturing the request performed from a given tool) I do not know what requests are stored and how sufficient these requests will simulate the service on a given attack vector. I have performed manually several requests that I described as a way to identify a HTTP server and which can be used for fingerprinting the http server based on the responses. I compared the results with the one returned from the real device.

HTTP header field ordering. Performed request is HEAD / HTTP/1.0\r\n\r\n

From the results on Table.4.5 we can see that the order of the HTTP headers is not changing. Due to the simplicity of the sent request and the fact it is partly different from the one performed during profiling, in Device 1 we observe two more HTTP headers returned. However, the order of the headers is not changing. For Device 3 the request was not recognized as valid request from the original device, while the limited validation from the decoy results with a matching request and returns a 200 OK response.

HTTP DELETE (forbidden operation) response: DELETE / HTTP/1.0\r\n\r\n

The results in Table.4.6. confirm the idea that the different HTTP servers could return different responses to a DELETE request. Here we see that Device 1 returns error 405 Not Allowed, while Device 2 and Device 3 return 400 Bad Request. The results from the previous request are confirmed following that the decoys are not able to identify that these requests should not be proceeded, since they are not part of the profile.

Improper HTTP version response GET / HTTP/4.0\r\n\r\n

On Table.4.7 we see that Device 1 and Device 2 respond with 400 Bad request, while Device 2 responds with 505 HTTP Version not supported. Here we see again that the decoys are not able

Table 4.7: Comparison of response to improper HTTP version

	Device 1 Original	Device 1 Decoy	Device 2 Original	Device 2 Decoy	Device 3 Original	Device 3 Decoy
Result	HTTP/1.1 400 Bad Request	HTTP/1.1 200 OK	HTTP/1.1 505 HTTP Version not supported	HTTP/1.1 200 OK	HTTP/1.1 400 Bad Request	HTTP/1.1 200 OK

Table 4.8: Comparison of responses to improper protocol responses

	Device 1 Original	Device 1 Decoy	Device 2 Original	Device 2 Decoy	Device 3 Original	Device 3 Decoy
Result	HTTP/1.1 404 Not Found	HTTP/1.1 200 OK	HTTP/1.1 400 Bad Request	HTTP/1.1 200 OK	HTTP/1.1 505 HTTP Version not supported	HTTP/1.1 200 OK

to determine that HTTP version is wrong and reply with status 200 OK, due to the matched request at the same address uri.

Improper protocol response GET / JUNK/1.0\r\n\r\n

On Table.4.8 the three original devices respond with different error messages when improper protocol name is send in the request data. Similarly to the other crafted requests, the decoys reply with status 200 OK as a matched request is found.

The results of this testing method clearly showed one of the limitations of the selected request-response approach. The search in the profile for correct response was strictly based on the path that has been requested and on the parameters that are transmitted. There was not deep analysis of the other information transmitted in the request. After this limitation was discovered I have inserted additional check for proper protocol name, HTTP method and protocol version. However, in order to return the correct responses as the original, every one of the problematic requests should be included in the profiling phase, by saving all of them manually in the profiler HTTP Manager source code. This solution might fix the observed differences, but any other requests which are not covered would still be problematic, because of the way the request-response method works and the desired minimum knowledge from the server how that given service works.

Even with the described limitation, the observed and tested performance of the created decoys for HTTP looks very promising and I believe that it covers the requirements of medium interaction honeypot. The possibility to dynamically increase the coverage of available requests, makes it possible that the generated approach can be easily adjustable to the desires of any potential customer.

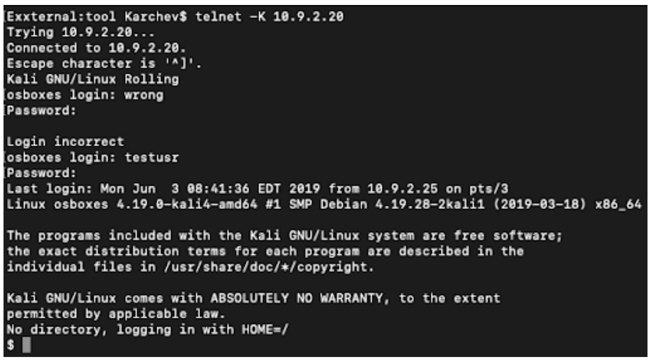
4.4.3 Telnet

Telnet is very popular service target for hackers from many years. Their primary goal is to obtain the credentials needed for further control of the server host machine. Our goal is to evaluate how the decoy is performing using these three types of experiments:

Simulating the behaviour of the service
Most Telnet communications are performed over a terminal or with specific client program(for example PuTTy). I will trigger a new telnet connection with the decoy from a terminal client. I will look at the responses returned from the decoy and if they are correctly provided. For the Telnet machine where I have access to the credentials, I will observe the transmitted data when these credentials are provided. For the other servers, I do not have a correct pair of credentials, so I will observe the behaviour just for proper responses to incorrect login verification.

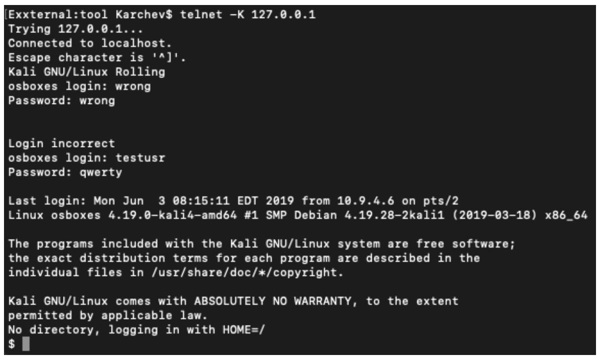
The client I will use to connect to a Telnet server is called "telnet" and the terminal command looks like this:
telnet -K address
where:

- -K is a flag that indicates not automatic login attempt
- address is the address of the Telnet server. In our tests it could be 10.9.2.20(real Telnet server), or 127.0.0.1(localhost when the decoy is executed on the local machine)



```
Exxternal:tool Karchev$ telnet -K 10.9.2.20
Trying 10.9.2.20...
Connected to 10.9.2.20.
Escape character is '^]'.
Kali GNU/Linux Rolling
osboxes login: wrong
Password:

Login incorrect
osboxes login: testusr
Password:
Last login: Mon Jun  3 08:41:36 EDT 2019 from 10.9.2.25 on pts/3
Linux osboxes 4.19.0-kali4-amd64 #1 SMP Debian 4.19.28-2kali1 (2019-03-18) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No directory, logging in with HOME=/
$ 
```

Figure 4.7: Telnet communication with the original Telnet server



```
Exxternal:tool Karchev$ telnet -K 127.0.0.1
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Kali GNU/Linux Rolling
osboxes login: wrong
Password: wrong


Login incorrect
osboxes login: testusr
Password: qwerty

Last login: Mon Jun  3 08:15:11 EDT 2019 from 10.9.4.6 on pts/2
Linux osboxes 4.19.0-kali4-amd64 #1 SMP Debian 4.19.28-2kali1 (2019-03-18) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No directory, logging in with HOME=/
$ 
```

Figure 4.8: Telnet communication with the Telnet server from the Decoy

A normal Telnet communication from the authentication phase usually consists of three types of exchanged messages: Server banners, incorrect credentials and error response, and correct credentials and control prompt. This communication for the tested Telnet server and the generated decoy are shown on Fig.4.7 and Fig.4.8:

As we can see from the two Telnet connections with the scanned Telnet server and the Decoy, the results are relatively identical. There is only difference between the two communication snapshots. It is the fact that the password characters are not displayed on the terminal when the test is on the original Telnet server. This difference is observed, because the decoy does not fully support the negotiation phase, where that server preferences are set. Hence this option is not negotiated and the terminal client is showing the characters when it is communicating with the decoy.

Compare the Profiles of the real device and the generated decoy
After the profile is created and the decoy is running, I have profiled it again by scanning the decoy. I have compared the results of the profiles.

The generated two profiles are almost identical. All of the valuable information is kept and not changed in any way. This includes all of data, all correct and error banners, all fields identifying when a sequence should end and fields identifying if the result of the sequence is correct. Hence, the second generated profile(used by scanning the decoy) is as valid as the one from the original device and it can also be used from the server to create a decoy.

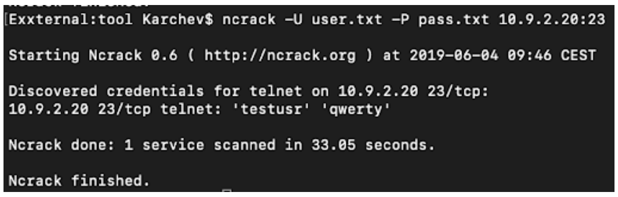
Compare the results of penetration testing tools on the real device and the decoy
In order to verify the correctness of the decoy and the responses it returns, I wanted to test it in a real scenario similar to how an attacker would try to connect to it. For that purpose I have used a tool called Ncrack. It is used to perform dictionary based attacks on different services, including Telnet. The main interests of an attacker usually is to obtain the credentials and gain control over the device. To ensure that the credentials would be found from Ncrack, I inserted them inside small dictionary files used from Ncrack during scanning.

The command to start the scan is:
ncrack -U user.txt -P pass.txt localhost:23
where:

-U user.txt : It is a text file that contains possible username that will be tested. The correct username is inserted inside to verify successful scan.

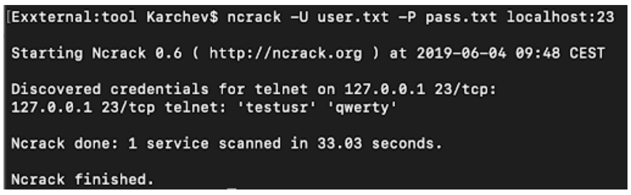
-P pass.txt : The same as the username file, this is a file that contains possible passwords that will be tested. The correct password is included inside.

address:23 : This is the address of the Telnet server and the port it is running on.



```
Exxternal:tool Karchev$ ncrack -U user.txt -P pass.txt 10.9.2.20:23

Starting Ncrack 0.6 ( http://ncrack.org ) at 2019-06-04 09:46 CEST

Discovered credentials for telnet on 10.9.2.20 23/tcp:
10.9.2.20 23/tcp telnet: 'testusr' 'qwerty'

Ncrack done: 1 service scanned in 33.05 seconds.

Ncrack finished.
```

Figure 4.9: Running Ncrack on the original server



```
Exxternal:tool Karchev$ ncrack -U user.txt -P pass.txt localhost:23

Starting Ncrack 0.6 ( http://ncrack.org ) at 2019-06-04 09:48 CEST

Discovered credentials for telnet on 127.0.0.1 23/tcp:
127.0.0.1 23/tcp telnet: 'testusr' 'qwerty'

Ncrack done: 1 service scanned in 33.03 seconds.

Ncrack finished.
```

Figure 4.10: Running Ncrack on the Decoy

Table 4.9: Average results of running the Ncrack on the original device and the decoy.

	Original Telnet Server (10.9.2.20)	Decoy Telnet Server (localhost)
Times credentials found	20	20
Average time of execution	23.72 seconds	30.02 seconds

On Fig.4.9 and Fig.4.10 are presented the results of running Ncrack on the real device and on the decoy.

As we see from the results, the two outputs are almost identical. Both of them successfully found the correct credentials. The time needed for Ncrack to finish scanning is respectively 30.05 seconds and 30.03 seconds. It also shows that the appearance of the decoy would successfully mimic the original Telnet server.

To have better and more general observation of the tool results I have executed the tool 20 times on both, real device and decoy. The average results are shown on table.4.9

The results show that the credentials were found all 20 times. There is increase of 26% of the time needed for execution. At this stage of the project this is considered as good performance.

Another test that I have executed is using other Telnet servers found with the Shodan search platform. Since I do not have access to their credentials and permissions to log in, the tests are performed observing the remaining responses over a Telnet authentication process. I have picked 10 random Telnet servers which were checked to be active by contacting them through the terminal client. The scan resulted in:

- 10 scanned Telnet servers
- 9 negotiation phases passed. One of the servers was combining the negotiation and authentication phases in the same packet. This resulted in incorrect responses send from the negotiation phase and not able to reach the next phase.
- 9 successfully generated profiles. All of the servers which passed the negotiation phase was able to generate a profile.
- 8 correct telnet sessions. One of the sessions was not properly created since the server was blocking any requests when a wrong credentials were transmitted.

These results showed that the majority of servers are properly scanned and the decoy is capable to simulate a normal authentication session. However, there are some servers which differ in some parameters during their communication. In such cases their scanning is not properly matched and the created decoy is not fully correct. These variations should be considered for the next version of the project, which will increase the robustness and adaptation level of the profiler and the server.

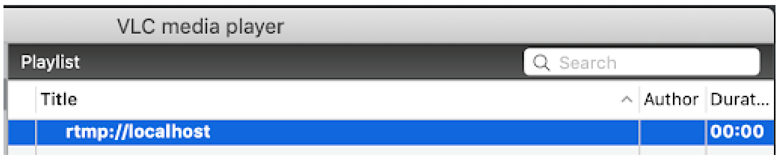
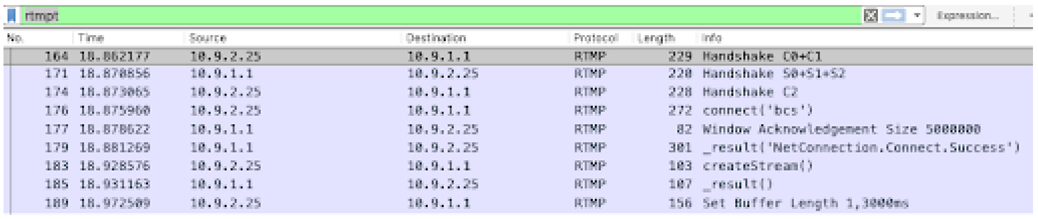


Figure 4.11: Loading decoy address in VLC player



```
rtmpt
No.   Time       Source     Destination  Protocol  Length  Info
164   18.862177  10.9.2.25  10.9.1.1     RTMP      229     Handshake C0+C1
171   18.870856  10.9.1.1   10.9.2.25    RTMP      220     Handshake S0+S1+S2
174   18.873065  10.9.2.25  10.9.1.1     RTMP      228     Handshake C2
176   18.875960  10.9.2.25  10.9.1.1     RTMP      272     connect('bcs')
177   18.878622  10.9.1.1   10.9.2.25    RTMP       82     Window Acknowledgement Size 5000000
179   18.881269  10.9.1.1   10.9.2.25    RTMP      301     _result('NetConnection.Connect.Success')
183   18.928576  10.9.2.25  10.9.1.1     RTMP      103     createStream()
185   18.931163  10.9.1.1   10.9.2.25    RTMP      107     _result()
189   18.972509  10.9.2.25  10.9.1.1     RTMP      156     Set Buffer Length 1,3000ms
```

Figure 4.12: RTMP packets from VLC player to the original device

4.4.4 RTMP

As we have already seen, a RTMP communication between a client and a server consists of several specific types of packets that need to be exchanged between the two parties before the actual stream begins. The tests I have performed to evaluate RTMP contains of the same type of methods used on the other services. They are now adjusted to the specifications and the goals an attacker could have by targeting RTMP. The performed methods are:

Simulating the behaviour of the service.
The proper communication of RTMP usually contains some security parameter(usually token) which is generated from the manufacturer based on their own criteria like device id, username and etc. To be able to obtain the tokent, the client should have also access to the credentials and the client software that is distributed from the manufacturer. I have already discussed how this process could be automated so triggering of stream commands could be started directly from the profiling tool.

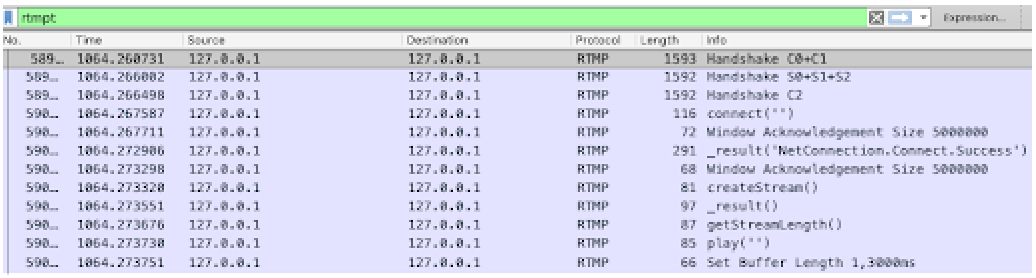
Our tests for proper work of the decoy and the level of service simulation consists of profiling the Reolink camera located the address 10.9.1.1. During scanning the profiler specifies the stream time that will be captured inside the profile. Then it saves all stream responses inside the profile which are received during that time.

The server reads the profile and runs the decoy on the host address(localhost). To generalize the process how an attacker could interact with a given RTMP server I could not use the client software that is specified for that device. Therefore, I have used a well known application software that is able to play many different audio and video protocols, including RTMP. That software is the VLC player [29].

The process starts by loading the RTMP address of the decoy inside the VLC player list. On Fig.4.11 this step is shown.

The lack of a token parameter results is stopped communication from both the real camera server and our decoy. The performed requests and responses for both stream attempts are shown on Fig.4.12 and Fig.4.13.

In both Figures we see how the RTMP communication is evolving in a similar way and reaching the point where in the last packet the client should send the token field. Since it is missing, both communications are terminated. This results show that the generated decoy is successfully mimicking the camera RTMP server.



```
rtmpt
No.     Time          Source      Destination  Protocol  Length  Info
589…  1064.260731   127.0.0.1   127.0.0.1    RTMP      1593  Handshake C0+C1
589…  1064.266002   127.0.0.1   127.0.0.1    RTMP      1592  Handshake S0+S1+S2
589…  1064.266498   127.0.0.1   127.0.0.1    RTMP      1592  Handshake C2
590…  1064.267587   127.0.0.1   127.0.0.1    RTMP       116  connect("")
590…  1064.267711   127.0.0.1   127.0.0.1    RTMP        72  Window Acknowledgement Size 5000000
590…  1064.272986   127.0.0.1   127.0.0.1    RTMP       291  _result('NetConnection.Connect.Success')
590…  1064.273298   127.0.0.1   127.0.0.1    RTMP        68  Window Acknowledgement Size 5000000
590…  1064.273328   127.0.0.1   127.0.0.1    RTMP        81  createStream()
590…  1064.273551   127.0.0.1   127.0.0.1    RTMP        97  _result()
590…  1064.273676   127.0.0.1   127.0.0.1    RTMP        87  getStreamLength()
590…  1064.273730   127.0.0.1   127.0.0.1    RTMP        85  play("")
590…  1064.273751   127.0.0.1   127.0.0.1    RTMP        66  Set Buffer Length 1,3000ms
```

Figure 4.13: RTMP packets from VLC player to the Decoy

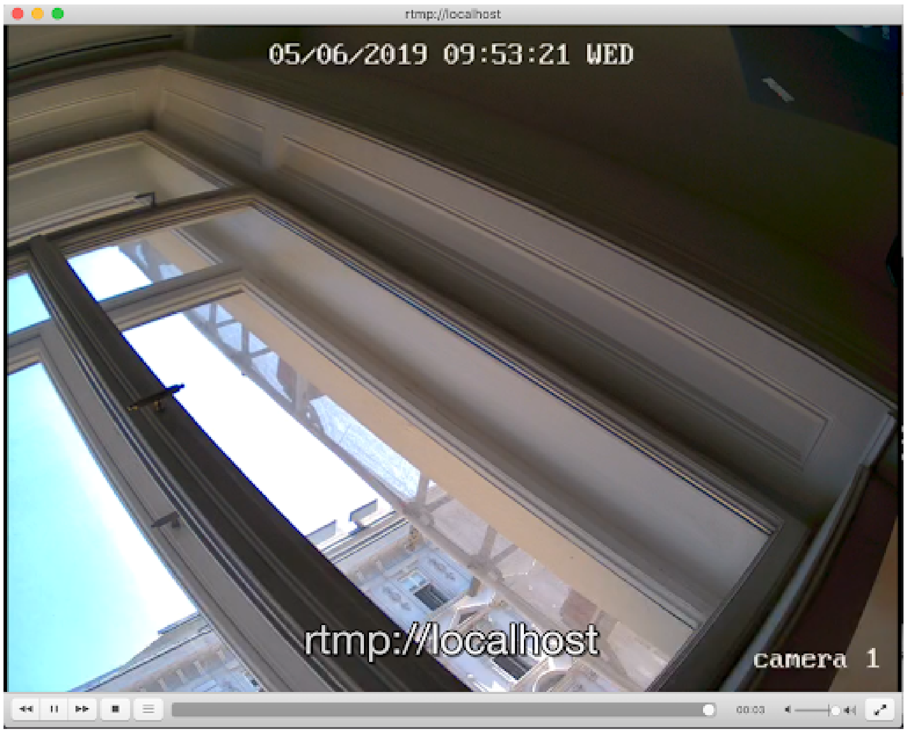


Figure 4.14: VLC player loading the responses from the decoy

In order to observe the decoy behaviour after this step, I have turned off the check for correct token validation inside the decoy. This way, the it should continue the stored sequence of responses from the profile even when the token is missing. These responses are the actual stream data stored in the profile during the specified time from the profiler. On Fig.4.14 we see a screenshot of the result shown on VLC player after the stream have been returned from the decoy.

As seen on Fig.4.14 VLC player loads the responses received from the decoy. The result is an image successfully displayed and originally captured from the real camera. VLC also indicates the duration of the received stream data. It is visible as the number of seconds from the stream start. This time is relatively equal to the time set in the profiler for capturing stream data, where any difference is caused because of the time needed to receive the first stream after the last request has been sent.

Using the shodan platform I have identified multiple other RTMP servers. However all of them are using some type of security parameters and does not allow free live stream access. Hence, the communication between a client and that server is terminated when that packets is received without having the correct token. Having this in mind, I have scanned 10 such RTMP servers found in shodan. The result of the scanning is a profile that contains the valid communication returned from the profile, without any actual stream data. This minimum coverage of the service is still useful for the purposes of a medium interaction honeypot. It will correctly fool an attacker that the RTMP server is actually

Table 4.10: Comparison of the profiles generated from profiling RTMP service on the camera and the decoy

Compared Parameters \Scanned device	Reolink Camera(10.9.1.1)	Decoy(localhost)
Number of open ports	7	7
Number of stored request-responses sequences	4	4
Number of responses after request 1	2	2
Number of responses after request 2	2	2
Number of responses after request 3	1	1
Number of responses after request 4	837	820

presented and any interaction with it would still be useful to determine how an attacker is trying to obtain access of that stream.

Compare the Profiles of the real device and the generated decoy

To verify that the decoy answers in the same way as the original RTMP camera server I have used the technique copy of the copy. I executed the profiler again using the running decoy as a target. I have used the copied requests from sniffing the camera traffic to begin the RTMP communication with the decoy. On Table.4.10 is shown the comparison between the profiles generated by scanning the camera and the decoy.

The results from the comparison indicate that the profile generated by scanning the decoy has similar parameters and is responding in a similar manner when it is being profiled. The only difference is the number of responses received as a result to the last request of the rtmp communication sequence. This is actually the streaming data that is captured for the time specified in the profiler(10 sec). The difference of 17 packets is relatively small and is observed because the decoy is delaying every response. The duration of the delay is equal to the amount of time passed from the first received request when the scanning is executed. Hence, there could be some minor increase of the delay when running the decoy analysis. The timeout of 10 seconds for capturing the stream data is completed before these 17 packets are handled. However, I believe that the decoy is properly mimicking the scanned RTMP server and the profile comparison proves that.

4.5 Result analysis

The global observation of the performed evaluation tests prove that the decoy successfully represents a medium interaction honeypot of the scanned IOT device. Most of the expectations for the correctness of the project results are validated. All services properly react when the normal usage of the device was tested. The results of Telnet clearly showed that when I am covering a specific phase of the service work, it could result in differences based on the client that interacts with the decoy. All decoys were successfully scanned from the profiler and the two profiles have showed very close results, which proves the consistency of the scanning phase and the decoy responses. The results of running other tools for testing showed almost identical results. For HTTP the results vary based on which profile is scanned. When the tool is used during profiling, the output is complete. When it is not used, the decoy is still performing as a proper device, but the specific device vulnerabilities are not observed. Having another tool during scanning, slightly increases the response correctness of the second tool. This implies that further analysis of the profile responses from the server could lead to better global performance of that service. A slight increase of the required time to complete a tool scan was registered. However this increased time corresponds to a normal execution deviation and should not be a factor when an attacker is performing that scan. Targeting specific attacks which are not covered in the profile resulted in differences from the original response. In such cases, these attacks can be specifically targeted during scanning, so it is assured that they are presented in the profile.

In conclusion, the evaluation results verified our expectations and proved that the decoy creates a proper virtualization of the IOT device. They helped us identify some problems and gave us ideas for further improvement of the project.

4.6 Competitors

4.6.1 Fingerprinting

The idea to create a fingerprint of an IOT device by scanning it is not new. It has been used for different purposes, but primarily to identify the type of the device that is scanned or the type of service that it is running. An attacker is then able to search for specific vulnerabilities related to this device or service and can proceed with exploiting it. Hence, the process of creating the fingerprint can be compared to the profile that is generated in the proposed project. Any tool that creates such fingerprint is potential competitor of the explained profiler and the correctness of the results in it.

nmap One tool with diverse functionalities is nmap. Nmap is highly popular for network scanning. It is also used in our profiler to find which ports are open in the IOT device and to execute some nmap scripts in the dynamic proxy phase. Besides that, nmap can be used to discover service types and versions. Having an accurate version number helps dramatically in determining which exploits a server is vulnerable to. Two other fields that version detection can discover are operating system and device type. The Nmap version scanning subsystem obtains all of this data by connecting to open ports and interrogating them for further information using probes that the specific services understand. This allows Nmap to give a detailed assessment of what is really running, rather than just what port numbers are open. Based on the obtained results nmap generates a fingerprint of the found characteristics. It recognizes more than one thousand service signatures, covering more than 180 unique service protocols. If nmap gets data back from a service that it does not recognize, a service fingerprint is printed along with a submission URL. This creates the possibility that the service database could be extended from community.

While nmap is very popular choice for service detection, operating system and device type identification, the fingerprint it generates differs in the way it is created and used from our profile. While nmap tries to identify and match specific outputs to results in a database, our profiler aims at much more. It uses the findings to store all data that identify what is the scanned device and how it works. This process should also be done in way and format that the results can be easily reproduced from the server. Hence, even that nmap generates a device fingerprint, the potential usage of it is different and very limited to what is performed from the profiler.

In the scientific area there have been different attempts to research the area of IOT fingerprinting with many different goals of potential usage. In some like [30] and [31] the goal is to create a development platform for the scanned device. Others like [32] aims at new cryptographic methods using such fingerprint. The approaches to create a fingerprint are similar across every cited solution. Many of them are also integrated in the profiler when a scanning is performed. However, the desired ending result does not allow us to use any similar method to save the obtained fingerprint. The reproducibility I focus on is only easily obtainable when the majority of the communication is obtained and registered in the generated profile.

4.6.2 Honeypot generation

The other main aspect of our research and the presented implementation is the generation of a honeypot(decoy). The concept of honeypots is not new and there are many solutions that are creating such deceptive systems for different technologies. As explained, the IOT domain differs of the approach that should be used, because of the huge diversity of devices.

Vdoo One approach that tries to overcome these problems and create a honeypot for an IOT device is by reading the firmware file of that device. One such solutions is Vdoo. It is a proactive threat detection honeypot that analyses the IOT device firmware binary to precisely mimic it. It's goal is to detect different attack methods which can be simulated. The generated honeypot then acts like real deployed devices as part of a real global network. Vdoo also provides web platform which is used for alerting on any tampering attempt to allow data utilization for immediate response and for deep forensic research.

Since Vdoo analyses the device firmware, it does not need to scan the device behaviour and create a profile of it. Another advantage is also that the performed analysis has a global view of the device behaviour and is not based on some specific services. These differences to the proposed solution in this document may be critical for a given user, however the Vdoo solution also create several limitations that should be considered.

Comparing Vdoo to the described approach is this document we can see that the desired end result is the same. Both solutions are capable to generate honeypot that mimics a real IOT device based on some characteristics it possesses. However, the used approach is totally different. Hence, the assumptions and requirements that Vdoo has in order to work properly are restricting the type and number of users that are capable to use their approach. Some of the assumptions Vdoo has are:

- The user has access to the device firmware
- The device firmware is of a type that Vdoo is able to analyze
- The firmware contains all valid information how the device works. Any possible situations where the behaviour is based on some server or cloud instructions are ignored.

The presented requirements could not be satisfied by many potential users for one of the following reasons:

- The device firmware is not available
- The device firmware could not be analyzed
- The generated honeypot is limited and could not be extended
- Any user specific behaviour could not be integrated in the honeypot

In conclusion, I can determine that both solutions have similar goal which they try to achieve with very different approaches. These differences present many advantages and disadvantages when the requirement of them are compared. For many users Vdoo could be an easier approach since no live scanning is needed, but the requirement of available firmware file that contains all valuable information is not always true. Hence, the proposed solution in this document will differ significantly of the obtained result and targeted customer groups. The dynamics of the profiling it creates would also lead to more detailed and adaptive honeypot.

Honeyd Another solution that comes together to our goals and purposes is Honeyd[33]. Similarly to the described document in this project, Honeyd uses service based honeypot generation. Honeyd mimics the network stack behavior of operating systems to deceive fingerprinting tools like Nmap and

Xprobe. The main difference is that Honeyd aims at low interaction honeypot. Honeyd does not simulate an application level services like I do, but it works on lower level with protocols like TCP, UDP and ICMP. Hence, the generated honeypot has limited abilities when it is approached by an attacker. It is able to fool scanning tools like Nmap, but further interaction with any of the available service will result in easy conclusion from the attacker that the contacted IP is not a proper functional device.

Another major difference between the proposed approach and Honeyd is the way that the honeypot is generated. The proposed solution from Honeyd includes configuring a template that mimics given device using set of commands that manage the traffic. This approach uses manual generation of this commands, while our project focuses on automatic and dynamic approach that is obtained with the profile creation.

The combination of automatically generated profile and medium interaction honeypot generation that can be easily extended and deployed to any IOT device is not covered from any of the researched solutions and competitors. The proposed project aims at obtaining the advantages of that combination which should result in successful application that will increase the security of any scanned IOT device by analyzing all potential attacks over it.

Chapter 5

Conclusion and future work

During my internship in Cybertrap I was introduced to the concepts of deception technologies. I got familiar with the product they provide and with the desired future implementation of a decoy for an IOT device. The diverse functionalities and device types make it impossible and impractical to create a fully functional honeypot platform for every device that should be mimicked. Therefore a research and first prototype implementation was required over the best approach to reach the desired outcome.

5.1 Contribution

In this document I presented my work to identify the best approach to profile and simulate an IOT device. The result solution is able to automatically and dynamically scan any IOT device by focusing on specific analyzed services. The created project consists of three main components. These are a profiler that scans the IOT device, a profile file that is the output of that scanning and a server that reads that file and works as a decoy.

The main focus of my work was the creation of the profiler. I have analyzed what information is needed to copy and reproduce the device behaviour in medium interaction way. The approach I have used is service based. It means that the behaviour of a device is considered as a combination of the services it is using and reproducing their behaviour will lead to reproducing the full outlook of that device. I have analyzed four application level services that are of primary interest for Cybertrap and differ in functionalities so more different approaches could be integrated with them. The analyzed services are: IPP, HTTP, Telnet and RTMP. The most appropriate approach that allows easy reproduction and dynamic extending is to store in the profile all important requests that I want to cover and the responses the device returns. Based on the way that exact service works, the selected approach is slightly modified. The two main modifications are based on whether the device responds with a single response to a given request(IPP and HTTP) or with multiple responses(Telnet and RTMP). Another important aspect is if the requests are independent of the stored order, or they are part of a sequence that should be followed for the correct reproduction.

The second biggest aspect of the proposed solution is the discovering of the important requests that should be integrated inside the profile. Several methods are proposed and analyzed. For some of the integrated services there are limited number of requests which identify the complete coverage of the service. Hence, including these services in the profiler is a sufficient way for its integration. Other services do not have such limited important requests and so they need to be identified. Some of the integrated ways inside the profiler for discovering these requests are: automatic loading or passive listening to the communication with device. This creates the possibility to give the user the option to dynamically adapt the comprehensiveness of the profile by interacting with the device, or by starting any penetration testing tool that analyses the device.

The next important component of the project is the profile itself. As part of the research different formats were considered and analyzed. The most appropriate format for the first project prototype is

JSON. It gives the possibility for faster software adaptation and modification from the server. Except all of the requests and responses, the profile contains information how they should be validated and updated accordingly to any change that should be fixed in the final response. Other non application level information is also integrated inside, which gives the possibility to adjust any low level packet fields of the simulated communication.

The last component that reads the profile and starts working as a decoy is the server. All of the integrated approaches are considered in such way that the server should have minimum analysis of the profile and what service it is currently simulating. This generalization is achieved with standard non service specific way to find the proper responses and update them accordingly.

In order to properly implement, test and evaluate many techniques were used. The goal of them is to test the decoy for correct behaviour in a standard device usage from their normal user, and with extreme situations of an attack with penetration testing tools. Some of the techniques are general and used for all services. Such example is the generation of second profile from scanning the decoy and comparing it with the original. Other techniques are adapted for the specification of the tested service and evaluate its correct usage. The results show that the generated decoy is successfully mimicking the scanned IOT device. Some limitations of the approach were discovered and they can be properly considered in future versions of the project. The solution is also compared with other projects that have similar approaches or results and it showed that none of them combines all desired characteristics having automatic and dynamic way to approach every tested device.

5.2 Future work

The performed research and the created project showed promising results that the approach to profile and then reproduce an IOT device behaviour is possible with the integrated methods. However, this is only the first step that was needed to proceed with future more direct solutions. Based on the findings of the research and the evaluation results of the project, I have identified several next steps that should be executed:

Generalization of the profile service format All of the used approaches like response-request, response-sequence and end-of-sequence approaches should be combined in a single general method. This will further reduce the level of analysis involvement from the server when reading the profile. That generalization could be obtained with additional fields and parameters included in the profile that will inform the server how exactly it should analyze the profile information and how to reproduce it, without knowing which method is used from the contacted service.

Enriching of the covered services The four integrated services are just initial stage that should lead to covering many more application level services. This will increase the potential usage of the profiler to many other types of devices. Some promising options are services like: HTTPs, SSH, FTP, RTSP and etc. Another potential service which is widely used from IOT devices is MQTT used for publish-subscribe-based messaging communication.

Non tcp service coverage All of the considered services so far are using TCP as a transport protocol. It creates more secure and reliable connection that is often the target of an attacker. However, in next of version of the profiler it should be able to simulate other services which use the other main protocol of the transport layer - UDP. Voice and video traffic is generally transmitted using UDP(unlike RTMP). Other services which are mainly accessed from routers are Domain Name System (DNS), the Simple Network Management Protocol (SNMP) and the Dynamic Host Configuration Protocol (DHCP). Another protocol that is primarily designed and used in the IOT domain is Constrained Application Protocol (CoAP).

Automatic non analyzed service coverage Based on my research and evaluation findings it is not possible to cover an application level service with abstraction from the way it works. There is always some specific data, request, or the order of them that is essential to be profiled.

Hence, analyzing and integrating the top services in the profiler is necessary for the medium interaction purposes that is my target. However, a more limited, but fully service independent approach is still needed to cover any non integrated service. Such service could also be any custom manufacturer communication running on that device. An example approach could be to passively or actively listen to a service traffic and automatically try to select the most suitable approach. It is a less precise and more prone to mistakes approaches, but a potential success would increase drastically the value of the profiler and would easy the process of deep integration of that service in a future step.

Standalone decoy platform In the current project, the role of the decoy is transferred to the server, which after analyzing the profile data is able to act as such. However, the decoy should be an individual virtual component that is not connected to any host machine. It will properly isolate the decoy from any host machine characteristics and communication. This virtualization is also needed for further integration into the Cybertrap platform that will take care of all logged events data reported from the decoy and will present it to the user in live and user-friendly manner. Choosing such platform requires additional research that will reveal which platform would be easily virtualized and the results from the logged events created from the decoy would not be coupled with the data generated from the platform itself.

Acronyms

API Application programming interface.

DDOS Distributed Denial of service.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IOT Internet of things.

IP Internet Protocol.

IPP Internet Printing Protocol.

JSON JavaScript Object Notation.

MISP Malware Information Sharing Platform.

OS Operating System.

PC Personal Computer.

REST Representational state transfer.

RTMP Real-Time Messaging Protocol.

RTSP Real Time Streaming Protocol.

SSH Secure Shell.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

WWW Worl Wide Web.

XML Extensible Markup Language.

Bibliography

[1] N. Eddy, 21 Billion IoT Devices To Invade By 2020, 2015
https://www.informationweek.com/mobile/mobile-devices/gartner-21-billion-iot-devices-to-invade-by-2020/d/d-id/1323081

[2] M. Antonakakis, T. April, Understanding the Mirai Botnet, August 2017
https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf

[3] R. Conoth, Malicious cryptocurrency miners, Jan 2019
https://arxiv.org/pdf/1901.10794.pdf

[4] Wikipedia, Honeypot (computing), May 2019
https://arxiv.org/pdf/1901.10794.pdf

[5] Medium Interaction Honeypots,
https://pdfs.semanticscholar.org/9d46/8fa983b844c76a07b1e3ea63d6f7a9cae294.pdf

[6] Wikipedia, Internet protocol suite, May 2019
https://en.wikipedia.org/wiki/Internet_protocol_suite

[7] Nmap scripts, http-enum
https://nmap.org/nsedoc/scripts/http-enum.html

[8] Wikipedia, Transmission Control Protocol, June 2019
https://en.wikipedia.org/wiki/Transmission_Control_Protocol

[9] Wikipedia, Internet Protocol, June 2019
https://en.wikipedia.org/wiki/Internet_Protocol

[10] P.Lippman, Passive Operating System Identification From TCP/IP Packet Headers, 2003
https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.114.194rep=rep1type=pdfpage=44

[11] Wikipedia, Network socket, June 2019
https://en.wikipedia.org/wiki/Network_socketRaw_socket

[12] Internet Printing Protocol/1.1: Model and Semantics, January 2017
https://tools.ietf.org/html/rfc8011section-5.4.15

[13] Hypertext Transfer Protocol – HTTP/1.1, June 1999
https://tools.ietf.org/html/rfc2616

[14] S. Shah, An Introduction to HTTP fingerprinting, May 2004
https://www.net-square.com/httprin

♭15♯ *Wikipedia‘Webscraping‘May*2019
https.◁◁en▷wikipedia▷org◁wiki◁Web_scraping

♭16♯ *TELNETPROTOCOLSPECIFICATION‘May*1983
*https.◁◁www▷rfc›editor▷org◁rfc◁pdfrfc◁rfc*854*▷txt▷pdf*

♭17♯ Wikipedia‘Telnet‘April2019
https.◁◁en▷wikipedia▷org◁wiki◁Telnet

♭18♯ Wikipedia‘SANSInstitute‘June2019
https.◁◁en▷wikipedia▷org◁wiki◁SANS_Institute

♭19♯ Wikipedia‘Denial›of›serviceattack‘June2019
https.◁◁en▷wikipedia▷org◁wiki◁Denial_of_Service_attack

♭20♯ Wireshark
https.◁◁www▷wireshark▷org◁

♭21♯ Ncrack
https.◁◁nmap▷org◁ncrack◁

♭22♯ H▷Parmer‘Adobe’s Real Time Messaging Protocol, 2012
http://wwwimages.adobe.com/www.adobe.com/content/dam/acom/en/devnet/rtmp/pdf/rtmp_specificatio

[23] Action Message Format, 2006
https://wwwimages2.adobe.com/content/dam/acom/en/devnet/pdf/amf0-file-format-specification.pdf

[24] Wikipedia, Secure Shell, June 2019
https://en.wikipedia.org/wiki/Secure_Shell

[25] LibRTMP, 2011
https://rtmpdump.mplayerhq.hu/librtmp.3.html

[26] SeleniumHQ Browser Automation, 2018
https://www.seleniumhq.org/

[27] Shodan search engine
https://www.shodan.io/

[28] ipptool
https://www.cups.org/doc/man-ipptool.html

[29] VLC media player, 2019
https://www.videolan.org/vlc/

[30] F. Pramodianto, IoT Link: An Internet of Things Prototyping Toolkit, 2015
https://ieeexplore.ieee.org/document/7306927

[31] Son N. Han, DPWSim: A simulation toolkit for IoT applications using devices profile for web services, March 2014
https://ieeexplore.ieee.org/document/6803226

[32] Q. Hu, Device Fingerprinting in Wireless Networks: Challenges and Opportunities, Sep 2015
https://ieeexplore.ieee.org/document/7239531

[33] N. Provos, A Virtual Honeypot Framework, 2004
https://www.usenix.org/legacy/event/sec04/tech/full_papers/provos/provos_html/

[34] M. Sweet, P. Zehler, How to Use the Internet Printing Protocol
https://www.pwg.org/ipp/ippguide.html

[35] J.Alet, pkipplib library, 2006
http://www.pykota.com/software/pkipplib/

[36] Owasp, Nikto
https://www.owasp.org/index.php/Nikto

[37] Vdoo, 2019
https://www.vdoo.com/honeypot-threat-detection

[38] H. Gupta, iFogSim: A toolkit for modeling and simulation of resourcemanagement techniques in the Internet of Things, Edgeand Fog computing environments, Oct 2016
https://onlinelibrary.wiley.com/doi/epdf/10.1002/spe.2509

[39] pkipplib, A python library for IPP requests. http://www.pykota.com/software/pkipplib/