POLITECNICO DI TORINO

Master degree course in Ingegneria Informatica

Master Degree Thesis

Exploiting virtual networks for CPS security analysis

The Smart Home Environment

Supervisor
prof. Antonio Lioy
prof. Andrea Atzeni

Candidate
Paolo Trungadi

Academic year 2022-2023

Summary

Cyber-Physical Systems, also known as CPS, are nowadays widespread systems and consequently a widely researched topic. They are used to describe systems with decision making capacities integrated into the real world through a combination of sensors and actuators. Different types of CPS exist but we will focus on a particular one, the Smart Home, which is a fast-growing application of CPSs.

In this context, the aim of the thesis is to study the state-of-the-art of these systems in order to better understand them and then to use virtual networks simulation tools to recreate some test environments since a real-world creation of testbeds for every case study would be very resource consuming and expensive. Instead, being able to adapt and use a simulation tool to this scope would allow us to set up multiple simulations based on the device or protocol we want to test.

Smart Home systems and generally Smart Devices are used to increase the quality of life, comfort, and ability of remote controlling the home. As mentioned, the topic was firstly thoroughly researched and analysed in order to gain complete and extensive knowledge before moving to the simulation part.

Initially, an overview of Cyber-Physical Systems' scope, possible applications, and security was provided; CPSs can find application in many fields such as Automotive, Smart Grid, and the focus of our research: Smart Homes. The security concerns were analysed by looking at the main security threats, the most common vulnerabilities, and failures.

We then moved on to exploring how Cyber-Physical Systems can be specialized and applied to Smart Home environments. This was done by looking at the key and most useful functionalities provided Smart Homes, the most common types of devices, and the most used and widespread protocols which are used by devices to communicate and provide services. Some test case examples were also described to provide examples of use of Smart Devices.

Successively, the cybersecurity aspect of Smart Homes was taken into consideration by firstly reviewing previous research on the security issues, trends, and evolution of these Smart Environments and by mentioning the OWASP most common IoT devices vulnerabilities. There are many concerns regarding the security of Smart Home and generally the use of "Smart" devices which can end up being an easily exploitable piece of hardware with an internet connection and next to no embedded security. Then, we classified the different types of attacks dividing them by category and citing the most common ones for each category before examining some of the most notable and impactful past attacks such as the IoT devices malwares Mirai and BrickerBot. The possible impact of said attacks was also discussed and some possible countermeasures were mentioned.

We later moved onto the analysis of the available simulation software, listing all the evaluated or tested tools, and then discussing more in depth the ones actually used for the purpose of this research: GNS3, EVE-NG, OMNeT++, and ns-3. We encountered some limitations on the tools'

side, such as devices being proprietary and not having the possibility of being simulated, interesting older projects not being maintained or completely discontinued, and general compatibility problems. Nevertheless, researching and testing was still possible, so at this point the simulated work was presented.

OMNeT++ was used to reproduce a 802.15.4 network and evaluating the effect of a basic battery depletion attack. ns-3 was instead chosen to test network denial of service attacks, particularly generic DoS, Distributed DoS, blackhole, and wormhole attacks, with the last two targeting the AODV routing protocol. GNS3, being a different kind of simulator from the previous two, was employed for the purpose of creating a simulation as realistic as possible, so iso images of systems used on real worlds devices were used to create a functioning and internet-connected network.

Finally, the conclusions on this research are drawn by evaluating the results obtained, pointing out the advantages and limitations of the tools used and by making some hypothesis on future research work.

Acknowledgements

I'd like to acknowledge the Professors Antonio Lioy and Andrea Atzeni for the opportunity of working on this thesis covering such an interesting topic and for their help in the overall process.

I would also like to thank the following people, who have helped me in completing this research and in achieving this master's degree.

Thanks to my family for always believing in me and being understanding in the most stressful moments, my grandmother Vittoria, my sister Chiara, and my parents Vincenzo and Silvia for always helping with everything.

I am also grateful for my closest friends Andrea B., Stefano S., Edoardo, Stefano M., Andrea N., Carla, Luca, and all the others for always being there for me and for the many happy memories built together.

Special thanks to Elisa with whom I have shared basically every moment of my experience at Politecnico di Torino, she always supported me and I learnt a lot from her.

Thanks to Luca and Andrea, university colleagues with whom I have shared many projects and who have always being helpful.

Lastly, I'd like to recognize my university, Politecnico di Torino, even though it has been source of so many challenges and tough moments in the last five and a half years of my life and it required many sacrifices, it helped me grow and has given me great satisfaction and good experiences.

Contents

Chapter 1

Introduction

The amalgamation of technology into our everyday life has altered the way we live and interact with the world around us. One good example of this fusion is the Smart Home setting, which encompasses interconnected protocols, devices and structures that allow for remote monitoring and control of multiple aspects of the house. The widespread availability of the IoT devices used for the implementation of these systems has produced numerous benefits, such as amplified energy efficiency and augmented comfort and convenience.

Nevertheless, the interlinked nature of smart homes has also presented numerous security concerns. First of all, the increase in quantity of linked devices has resulted in a greater attack surface, thereby rendering smart homes more vulnerable to cyber-attacks. Secondly, having the Smart Home's network connected to the internet is another potential vulnerability if the proper precautions have not been taken. There is an old joke saying that the 'S' in 'IoT' stands for Security hinting at the, unfortunately, very common lack of security in IoT devices which opens even more doors for vulnerability exploitation and subsequently system impairment.

The goal of this thesis is to tackle these security concerns by exploring the potential of virtual networks in improving the security of Cyber-Physical Systems, particularly of one of their specializations: Smart Home environments. Through the deep analysis of both the standards currently in use and the known security vulnerabilities, the evaluation of possible simulation tools and the formulation of virtual network simulations we aim to highlight the most crucial areas and potential for their improvement.

Network simulation is a field that is growing consequently to the growth of the wireless and computer networks technologies since there is the need of testing protocols and devices before they can be handed over to real world users. Network simulation can be defined as a method of research of a computer network and different features of the environment can also be tweaked to evaluate how the network or protocols would behave and perform beneath different conditions. [1] More specifically, simulating network means using a software program to analyze the behaviour of a network by computing the interaction between the various devices and entities of said network. Additionally, the tool could be providing visualization, animation of flow packets, performance of the network, and cost effective technique to design this network. [2]

With our focus on Cyber-Physical Systems, particularly on Smart Home environments, these network simulation tools could potentially be used to simulate the behaviours and performance of modelled smart home systems to support designers and engineers. Such tools can be adopted both during the product design and first environment design and during environment validation and service design. [3] Our interest lies clearly in the potential of validating the system, specifically looking at the cybersecurity features of the simulated network.

Simulating for the purpose of analyzing the cyber safety and security of a system has been topic of research for a while now, for example in 2019 *H. Kavac et al* [4] reviewed the state of the art and future direction of simulation for cybersecurity. The research explains how cybersecurity is a notable challenge because it involves a mix of physical, software, and human systems; the studies on this systems of systems are then based on physical, emulated and simulated models. The article then classifies these models as follow:

- "*Physical* models are a mix of hardware and software connected via network and can be very costly."
- "*Emulators* act in the place of a real device, usually realized as software, and are used to provide greater flexibility to physical models."
- "*Simulation* models are purposeful abstraction of physical systems to explore how the interrelation between the human, social, software, and hardware systems might lead to vulnerability or resilience. Simulation models provide a means for examining complex interactoins and changes within the system over time."

The virtual networks can thus provide a secure and regulated environment for recreating, assessing, and evaluating the resilience of Smart Homes against cyber-attacks without having to physically buy and set up the devices which would surely requires more resources. On the other hand, working with the simulation tools can presents some difficulties and limitations compared to the real-world devices.

In conclusion, this thesis will deliver a comprehensive analysis of the security of Smart Home environments while also providing context on these systems such as the most common devices, protocols, technologies and vulnerabilities. Additionally, the benefits of using virtual networks to assess the systems in question and their resistance against cyber-attacks will be tested and discusses.

1.1 Thesis Structure

The thesis is organized in the following way:

- In chapter 2 an overview of the Cyber-Physical Systems is provided, along with a discussion on their security and an introduction to one of the possible CPS specialization: the Smart Home environment.
- In chapter 3 we analyze more deeply the elements that constitute a Smart Home in order to get full knowledge on how these systems and their devices work.
- In chapter 4 we extend the previously mentioned analysis to the cybersecurity aspect of the topic, reviewing the most well-known vulnerabilities and notable historical attacks that interested these systems.
- In chapter 5 the focus is moved to the simulation tools, mentioning which were considered and providing an analysis of the ones actually used or tested.
- In chapter 6 the simulations created are presented, with a discussion on what was the purpose and how it was implemented, along with an analysis of the result obtained.
- In chapter 7 we draw the final conclusions of the topic, discussing how we were able to use the simulation tools to our advantage, their limitations and possible future research works.

Chapter 2

Cyber Physical Systems overview and the Smart Home specialization

Where does the term Cyber-Physical System come from? It was first proposed by Helen Gill in 2006 at a National Science Foundation (NSF) workshop, and a CPS can be defined in different ways.

According to the NFS (National Science Foundation)

> *"Cyber-physical systems are engineered systems that are built from, and depend upon, the seamless integration of computation and physical components. Advances in CPS will enable capability, adaptability, scalability, resiliency, safety, security, and usability that will expand the horizons of these critical systems."* [5]

While the NIST (National Institute of Standards and Technology) defines a CPS in its glossary as

> *"A system integrating computation with physical processes whose behavior is defined by both the computational (digital and other forms) and the physical parts of the system."* [6]

So, we can identify a CPS as a network of embedded systems that interact with physical input and output. This combination of different interconnected systems and the main components being sensors, aggregators, and actuators provide the ability to monitor and operate real IoT-related object and processes.

From these definitions is clear that a CPS is a peculiar type of system where real-world objects and infrastructures are integrated with sensing, computation, control, and networking capabilities while also having access to the Internet.

This sophisticated blend of cyber and physical components allows for smarter devices and systems which can be more precise, responsive, and efficient than the traditional ones.

2.1 Cyber-Physical Systems scope and application

Nowadays a CPS finds application in industries from a wide range of domains, anywhere the functionalities mentioned in the section before could be helpful. Consequently, the scope of the system will change based on where it is being used.

Cyber-Physical Systems applications can be found in smart homes/buildings/cities, smart healthcare devices, smart grid, smart water networks, smart manufacturing, smart factory, autonomous vehicles, and more.

Some examples of applications are:

- **Autonomous vehicles:** In the recent years the number of autonomous vehicles is growing rapidly, and it's easy to see why CPSs are heavily used in these machines. They enable a smooth integration of the sensors of the vehicle (such as video feeds or lidar/radar) with the control algorithms necessary to plot a route and avoid obstacles.

- **Smart Grid systems:** A Smart Grid is an electrical grid with additional operations and energy measures capabilities used to control the flow of electricity. The purpose is improving the efficiency, reliability, flexibility, and sustainability of a standard grid.

- **Medical Devices:** Many healthcare wearable devices are improved by the CPS technology, for example peacemakers and insulin pumps are two crucial devices that have to monitor in real-time vital signs or other physiological measurements and act according to the values.

- **Smart Homes:** Smart Homes are going to be deeply discussed in the later sections, but once again it's clear how a CPS can bring benefits to a Smart Home environment where multiple devices and sensors need to be connected and communicate, with some devices taking decisions based on the information provided by others.

2.2 Cyber-Physical Systems security

Integrating with the research presented by *J.A. Yaacoub et al.* in the article *"Cyber-physical systems security: Limitations, issues and future trends"* [7] we can briefly classify the Cyber-Physical System threats, vulnerabilities, attacks, and failures.

2.2.1 Security threats

First of all, it has to be noted that in a CPS the security threats can be cyber, physical, or a combination of both. Cyber-attacks are often easier to deploy from any device, unlike physical threats that most likely require specific tools and physical access; cyber-physical security threats are also easily scalable and they can quickly spread through any unreliable domain.

Cyber threats aim at jeopardizing the classic functionalities of a cyber system:

- Confidentiality
- Integrity
- Availability
- Reliability
- Authenticity
- Non-repudiation
- Accountability

2.2.2 Vulnerabilities

They can be divided into Network, Platform and Management vulnerabilities and usually are caused by:

- **Increasing Connectivity:** Which means increasing the attack surfaces, thus more vulnerabilities can be found.
- **Heterogeneity:** With a CPS being a heterogeneous mix of products from multiple vendors each device is prone to different vulnerabilities.
- **USB Usage:** Even though USBs are not as common nowadays, the infamous Stuxnet attack comes to mind, where the malware was planted into an offline system by plugging in an infected USB.
- **Bad Practice:** As every other system, bad practices such as leaving default credentials or bad coding standards are a source of vulnerability.
- **Spying:** CPS are prone to surveillance attacks, also thanks to their distributed structure where a spyware on a device could go unnoticed for a long time.

2.2.3 Attacks

A Cyber-Physical system is at risk not only from physical threats but also from cyber-physical threats; to be more precise in modern systems we can find four potential types of attack:

- Physical only attack,
- Cyber only attack,
- Physical-enabled cyber-attack,
- Cyber-enabled physical attack.

With a physical-enabled cyber-attack being an attack in the cyberspace that can be deployed because of a physical vulnerability and vice versa a cyber-enabled physical attack is a real-world attack happening because of a cyber vulnerability.

Nonetheless, as we will discuss in a later section any of the modes of attack just mentioned can have repercussion in both the cyber and the physical space.

Attacking a CPS can involve attacking sensor devices, actuators, computing components, communication, and feedbacks.

A few examples of physical attacks can be wire cutting/tapping, physical breach, introduction of infected items, key-card hijacking, and social engineering. On the other hands cyber attacks can include eavesdropping, cross-site scripting, SQL injection, password cracking, phishing, replay, malware infection, and obviously DoS/DDoS.

2.2.4 Failures

A failure in a Cyber-Physical System can be of different types, with the main ones being:

- **Content Failure:** happens when the content of the delivered information is inaccurate, and a function system failure could ensue.
- **Timing Failure:** means that the timing of information delivery is delayed or interrupted, and this could influence the decision making process.
- **Sensor Failure:** failure indicates that a sensor is not functioning properly and this could again affect the decision making process.

- **Service Failure:** occurs when an error propagates through the service and influence its normal performance ability; this could evolve into a partial or full CPS system failure.

- **Consistent/Inconsistent Failures:** a failure can be consistent if identically perceived by all CPS users, or inconsistent if the users differently perceive an incorrect service.

2.3 A Cyber-Physical Systems specialization: the Smart Home system

We are now going to focus on the Smart Home application of a Cyber-Physical System, but what exactly is a Smart Home?

The first appearance of Smart Home technology was the creation of the X10 protocol in 1975. X10 is a wired communication protocol that take advantage of a home's AC wiring to allow communication between devices and control modules.

In 1997 the Media Laboratory of the MIT (Massachusetts Institute of Technology) published an article called "Smart Rooms, Desks, and Clothes" [8] where they talked about their work towards developing smart networked environments that could help people in their homes, offices, cars, and when walking about.

Then, with the arrival of 21st century the development of wireless technologies started to unlock the full potential of smart homes, opening up the possibility of using stand-alone devices or even DIY (Do-It-Yourself) systems.

Today, according to the Oxford Languages dictionary a Smart Home is:

> *"a home equipped with lighting, heating, and electronic devices that can be controlled remotely by smartphone or computer: you can contact your smart home on the internet to make sure the dinner is cooked, the central heating is on, the curtains are drawn, and a gas fire is roaring in the grate when you get home"* [9]

More specifically, integrating with the definition proposed by B. Dvorsak et al [10], it's a residence where an organized home automation system connects all the electrical devices to manage lighting, heating, air conditioning, ventilation, security alarm system, audio and video system, call devices, energy control equipment, presence automation (door, windows, blinds, gates), technical alarms, and more.

Therefore, a Smart Home is created when separate installations of the house get connected into a common system. The main benefit coming from this automation is the reduced need of human interaction and an increase in quality of life, comfort and safety; additionally, since you can optimize the devices and their interactions, a Smart Home system is usually able to reduce energy waste and be more efficient.

A Smart Home does not have to be a full-scale system where the house is designed and built to be smart and every appliance is connected, even though these kinds of systems are the ones that provide the biggest improvements from every point of view, aside from costs and set-up time. A homeowner buying just a few smart devices and linking them to work together is also making his house "smarter" and integrating a Cyber-Physical System to his advantage. A simple example being the use of a smart thermostat which controls the heating based on data received from temperature sensors or user inputs remotely given through a mobile application.

Having different home control systems connected with each other provides great flexibility while managing the house, and it results in almost every device being remotely monitorable and controllable.

2.4 Chapter recap

In this chapter we defined what a Cyber-Physical System and mentioned some of its possible scopes and applications. Additionally, an overview of the CPS' security was given by analysing the possible and most common security threats, vulnerabilities, attacks, and failures. We then introduced and defined the Smart Home, the Cyber-Physical System specialisation on which this thesis focuses.

Chapter 3

The Smart Home Environment

In order to fully understand the Smart Home environment that we are studying, we need to take a closer look at its element and their functions. The following section will explore in more detail the key functionalities provided by a Smart Home as well as the most common components and technologies used.

3.1 Smart Home functionalities

- **Heating**: The heating (or A/C) system is one present in every house, and there are multiple devices you can use to improve the control over the home's temperature; as mentioned you can set specific rooms to higher or lower temperatures and you gain the ability to remotely monitor and command the system.
- **Lighting**: It's another universal apparatus with a key role since we are mostly at home when the natural light goes down outside, so there is the need to use artificial light. The functionalities that can be implemented with smart lights are many, some examples are lights turning on automatically with presence sensors, outside lights (e.g., porch lights) turning on at night and off at sunrise, slowly dimming out the lights in a baby's room while he's falling asleep and many more.
- **Energy Management**: Many smart system can now monitor the energy consumption of the device and display it in real time; this alone could help a user reducing his energy consumption by making him more aware of which devices are more power draining. Then all the other additional functionalities of the devices (e.g., self-shutting down after some idle time or when the user leaves the room/house) can help reducing the energy waste.
- **Home Automation**: With the devices being "smart", or at least connected to a smart controller, and interconnected the possibilities of tasks automation are endless and go further the simple action taken at a pre-set time, or remote activation.
- **Safety & Security**: Two fundamental functionalities to have for the well-being of the house owner who can greatly benefit from the integration of these devices with real-time monitoring and notification systems through software programs.
- **Weather control**: A more specific application, but it has some very useful use cases; the house central system can get fed information from a weather module and automatically close the windows if it starts raining, or the sunshade in the event of strong wind.
- **Use of Smart Appliances:** Even without having a complex interconnected system installed the use of stand-alone smart appliances is very simple yet useful; nonetheless, if you have multiple smart devices and appliances they can interact and provide more functions, and it's relatively easy to link a new one.

- **Remote control**: Since basically every subsystem mentioned is connected to a software controller, they can all be managed remotely through a mobile application; a very simple example is logging in the application to check if you turned off the lights or the television while leaving the house, and eventually being able to do so from your smartphone.

3.2 Devices

We will start by showcasing the different types of devices, briefly going over their basic functionalities and then making some examples of use.

3.2.1 Smart Speaker

One of the most common devices probably thanks to the ease of use and the "plug and play" behaviour, some examples are Google Nest, Amazon Echo, and Apple Homepod. It can be used through vocal activation to interact with the proprietary vocal assistant (respectively Google assistant, Alexa, and Siri) for basic tasks such as asking for the weather forecast, setting up an alarm, or reproducing music.

Additionally, if a smart device is connected you can use voice commands to control it, for example if you are using Amazon Echo and some smart lightbulbs you can just say "Alexa turn off the kitchen lights".

3.2.2 Smart Hub

A Smart Hub is also used to control devices but instead of working with voice activation it requires an app to setup and use. It works as a master controller for all devices and offers more complex and complete home automation with the peculiarity that it is able to translate between the different protocols used by devices. This enables the usage of routines or reactionary events such as turning on the lights when the front door is unlocked. Between the most common ones we have the Samsung SmartThings Hub, the Google Nest Hub, and the Aeotec Hub.

3.2.3 Smart Thermostat

A Smart Thermostat does exactly what the name suggest: it allows an user to control the temperature of his house from anywhere since it's app-enabled, with additional features such as having different temperatures for different rooms or programming the heating to start/stop at a specific time.

Its decision making is based on information given by temperature sensors spread around the house from which he frequently receives the updated measurements. To detect whether the heating (or A/C) is needed the thermostat could also be integrated with occupancy sensor or gather information directly from the user's phone location. Some of the most well-known are the Amazon Smart Thermostat, the Google Nest Learning Thermostat, and the Ecobee Smart Thermostat.

3.2.4 Smart Lightbulb

Smart LED lights are probably the most popular and easiest device to get into the Smart Home world. The available products range from LED strips to dimmable light-bulbs and coloured table lamps. Philips is the most established brand with its Philips HUE line, but there are others such as Wyze and TP-link.

3.2.5 Smart Plug and Outlet

The key thing about Smart Plugs and Outlets is that they have the capability of transforming into a "smart" device any common appliance or electric device that is plugged in, because the user gains the ability of turning it on/off through the phone, setting up a schedule, or even monitor the energy consumption.

There isn't really a reference brand for this kind of products as almost every technological vendor started selling them, but some of the best could be the one produced by Wemo, TP-Link, and Wyze.

3.2.6 Smart Smoke or Carbon Monoxide detector

Smoke or CO detector are certainly not a new security measure, but them being "smart" brings us some advantages:

- Primary advantage as for every smart device is the connection to a mobile application, which means that if a fire was to start you will know about it immediately even if you are not home which enables a faster response to the emergency.
- Another advantage to the detector being interconnected with other ones in different rooms/floors and then to the application is the better and more specific information that it will provide: instead of just hearing a loud sound you will get a notification telling you in which room the problem is.
- Bonus feature is the ability of the devices communicating the low-battery level "silently" through the application instead of the old method used: the sound alarm going off no matter what time it is.

3.2.7 Smart Door Lock

A Smart Door Lock is a pretty straightforward device, it allows a homeowner to enter their home or grant access to others without the use of a traditional key. Instead, a smartphone or a key fob is sufficient to wirelessly verify and unlock the door; some additional features are for example remote control, entry/exit logs monitoring, creation of temporary entry codes, and auto-locking. Among the best reviewed we have the August and the Ultraloq Smart Locks.

3.2.8 Smart Doorbell

A Smart Doorbell consist of a camera, microphone and speaker that lets the homeowner watch and talk to whoever is at the door through a smartphone application. Every activation of the camera is recorded and stored for a certain period of time so that you can go back and watch any interaction you had or anyone that stopped by the door that you may have missed.

Amazon Ring is one of the most famous (and sponsored) brand, then we can find again Nest and Wyze in the Doorbell vendors group.

3.2.9 Smart Security System

A Smart Security System is a highly customizable group of components used to improve the security of your residence that can range from a "do-it-yourself" kit to large scale setups provided with professional installation and monitoring. The peculiarity of a Smart security system, compared to a classic one, is it being connected to the internet so that the user can monitor and manage everything from a mobile application (e.g., watching the live security feed or receiving notifications when any alarm goes off).

The main components are the following:

- **Security Cameras:** They enable the remote live-stream of the house and the reception of notification when any activity is detected. They can be indoor or outdoor cameras, with the additional possibility of having infrared or night-vision capabilities, and the feed can be stored either locally or on the cloud.
- **Sensors:** There is a big variety of sensor that can be installed with the main ones being motion sensors that are the ones installed in rooms or hallways, entry sensors that can detect the opening of a door/window, and the glass break sensor which can pickup the glass breaking sound.
- **Siren:** This is pretty much self-explanatory, a siren will sound whenever an alarm goes off with the intent of scaring off intruders.
- **Smoke and CO detectors:** As already mentioned as stand-alone devices the smoke and carbon monoxide detectors are a recommended tool to control the safety of the environment and many professional setups include them.
- **Keypad:** Often a simple control point mounted on a wall, it could also include a screen or directly be a touchscreen, from where you can insert the codes to activate or deactivate security measures.
- **Panic button:** Again as evident from the name a panic button is there in case of emergency to easily and quickly alert emergency services, whether it's going to be calling the police, an ambulance, or the fire department.

3.2.10 Smart Appliances

This section has the purpose of briefly mentioning some of the other possible appliances that could be added to a Smart Home system such as clothes washers/dryers, automatic vacuums, ovens, televisions, coffee makers, fridges, scales, and more.

These are all devices that could benefit one way or the other from being connected to a mobile application, for example nowadays there are fridges that keep track of the items contained and their quantities, coffee makers that let you personalize the coffee mix from your mobile phone, or smart television that you can fully manage from your smartphone.

3.3 Protocols

Just as there is a big variety of devices to pick from for your smart home, the same thing goes for the protocols that can be used: some devices use proprietary protocols developed by their vendors while other devices are more flexible and can work with different protocols. For example, central devices such as smart speakers and smart hubs (e.g, Amazon Echo, Google Home Voice Controller...) are able to communicate to different end devices through different protocols.

We will now more closely consider the most common protocols being used in the smart-home environment, starting from the more widespread ones (e.g., Wi-Fi) and working our way to the more context-specific ones.

3.3.1 Bluetooth

Bluetooth is used for short range communications and it's very good at avoiding interferences since it uses a technology called *'FHSS'* Frequency Hooping Spread Spectrum which divides the frequency bands in eighty 1MHz channels; the transmission then makes use of a combination of these channels, changing frequency up to 1600 times per second.

Additionally, the low power consumption variant 'BLE' Bluetooth Low-Energy (also known as Bluetooth 4.0) can be frequently found, since IoT devices are often built with minimal hardware.

3.3.2 Wi-Fi

Wi-Fi is a name that everyone has heard at least once in his life, but what does Wi-Fi actually stand for?

As stated by the founders of 'The Wi-Fi Alliance' and by the IEEE [11] "Wi-Fi is a short name for Wireless Fidelity" and it generally refers to any type of IEEE 802.11 Wireless Local Area Network (WLAN); more specifically it's the industry standards for products conforming to the IEEE 802.11 standard.

The 802.11 standards are under the family of IEEE 802 standards that include LAN and MAN (Metropolitan Area Network) standards with the purpose of defining over-the-air protocol needed to support networking in a local area. It uses radio frequencies to transfer data and allows high-speed Internet connection without the use of cables.

Nowadays it's hard to find a house or a building without at least one Wi-Fi network, so consequently there has been an exponential growth of "plug and play" devices that only need to be connected to a Wi-Fi network with internet access in order to provide their full services.

Furthermore, devices can greatly benefit from wireless networks even if there is no internet access but rather utilizing it as a local way of communication.

3.3.3 802.15.4

Also coming from the IEEE 802 family of standards we have 802.15.4 which according to the IEEE Standards Association definition provides "the physical layer and MAC sublayer specification for low-data-rate wireless connectivity with fixed, portable, and moving devices with very limited batter consumption requirements." [12]

802.15.4 is used to define the operations of a LR-WPAN Low Rate Wireless Personal Area Network, where lower data rates are traded off in favour of lower complexity and battery drain, fitting properties for IoT devices.

3.3.4 6LoWPAN

6LoWPAN is a standard created by the Internet Engineering Task Force (IETF) to approach the usage of IPv6 routing in Low-power Wireless Personal Area Networks.

The idea behind the creation of 6LoWPAN was introducing the benefits of standard IP networking to low-power mesh and sensor networks which usually contain low-processing capability devices and are often limited by proprietary technologies. Using 6LoWPAN we obtain a low power wireless mesh network where every node has its own IPv6 address.

As said the general concept is providing wireless connectivity at low data rates with a low duty cycle but 6LoWPAN finds application in many fields such as Automation, Industrial Monitoring, Smart Grid, and clearly Smart Homes with the key thing being able to connect the home IoT devices via IPv6, which can bring notable advantages compared to older IoT systems.

3.3.5 ZigBee

ZigBee is an open standard for low-data rate, low power application designed for machine-to-machine (M2M) and IoT networks. Being an open protocol ZigBee should theoretically grant the ability of mixing implementations from different manufacturers but, in reality every ZigBee product is customized by vendors, rendering compatibility harder to obtain.

Compared to Wi-Fi networks, where the endpoints are connected to an high-speed network, ZigBee works with lower data rates while using a mesh networking protocol to create a self-healing architecture and to avoid the need of a central device.

3.3.6 Z-Wave

Z-wave, similarly to ZigBee, is a wireless communication protocol that provides a lower-power alternative to Wi-Fi and a longer-range alternative to Bluetooth for smart devices connection and commands, it's also a bit slower than ZigBee, but requires less energy to cover the same range.

A Z-wave network contains a primary controller, known as smart hub, which is usually the only appliance connected to the internet, and other IoT devices which refers to the central hub. Using a source-routed mesh network technology, Z-wave signals can hop trough devices up to four times to reach the device that the user wants to control.

3.3.7 KNX

KNX is an open protocol, more specifically a KNX system is a bus system for building control which means that all devices in said system see the same transmission and are able to exchange data via a common bus network.

One of the main characteristics is the usage of a decentralized topology: there is no need for a central control unit since every unit connected to the system is itself smart, so the "intelligence" of the system is spread across all its devices. A clear advantage that this decentralized structure brings is not having a central SPF (Single Point of Failure): if one device was to fail the others can continue undisturbed and only the applications depending on the device that failed will be interrupted.

3.3.8 Thread

Thread is a relatively newer IPv6-based networking protocol designed specifically for low-power Internet of Thing devices in an IEEE 8025.15.4 wireless mesh network, commonly called WPAN (Wireless Personal Area Network).

The main features advertised are:

- Simplicity;
- Security obtained from the authentication of all devices in the network and encryption of all communications;
- Reliability given by the self-healing mesh networking, absence of a SPF, and spread-spectrum techniques employed to provide immunity to interference;
- Efficiency since devices can sleep and operate on battery power for years;
- Scalability up to hundreds of devices.

3.3.9 Matter

Matter is an open-source application-layer connectivity standard created to ease device manufacturers and developers in connecting and building reliable and secure ecosystems while increasing compatibility among devices. It was introduced by an association of big companies: Amazon, Apple, Google, Comcast, and the Connectivity Standards Alliance (former ZigBee Alliance) which is the group of companies that also maintain the ZigBee standard.

The standard was created following the idea that the devices used in a smart home should be reliable, secure, and have an easy and seamless integration. The protocol was modelled using IP and is compatible with Thread and Wi-Fi networks; by building over IP, Matter enables communication across different smart home devices, mobile apps, cloud services, and characterizes a set of IP-based networking technologies for device certification.

3.4 Use cases

We will now provide three brief scenarios of implementation of Smart Home systems, to give some examples of how the devices can be used and can interact with others.

3.4.1 First use case

When Alice moved to her new house, she bought some smart devices to monitor and manage it and is now using the Samsung SmartThings application on her phone to control all the devices. The SmartThings application allowed her to add her different devices: a Ring Doorbell camera, a smart thermostat, and some Philips Hue Lightbulbs.

One thing that she can do with this setup is turning down the heating of her house while she is away and turn it back on just before coming home so she will still find a warm house while saving on the bills and reducing the energy waste. This is clearly a more flexible solutions compared to standard programmable thermostats since those can only be set to start heating at a certain time and could not account for days when Alice comes home earlier from work or goes unexpectedly out with friends, coming home later than usual.

Another example is controlling the doorbell camera from her phone, which means that Alice can check who is ringing from anywhere or look at past "events" happened by her front door. She can even talk with whoever is ringing her door if she ever needs to communicate for example with a courier delivering something while she is not home.

3.4.2 Second use case

Bob is less of a smart devices' enthusiast, but since he received a smart Speaker (e.g., Amazon Echo) as a gift he learnt that it's helpful to do various tasks while he's busy with different things around the house. For example, he can start downloading the latest episode of his favourite show on the TV as soon as he gets home, he can ask for traffic information while getting ready to leave and a lot of more.

He also bought some smart lightbulbs that he spread all over the house and since they easily bond via Bluetooth with Alexa he can now turn on/off or just alter the lights to his preference from his phone or with his voice without having to move.

3.4.3 Third use case

Susanna is more tech savvy than the other two so when she renovated her house, she went a step further and installed smart actuators that operate electric motors all over her house so that she has control of almost every part of the house. Then she set-up a smart hub to bridge together every device and control the communications among them even when they work with different protocols.

The usage of this smart hub allowed her to automate lot of things around the house, for example the information sent by the smart thermostat is used by the heating system and by the smart plugs where the A/C system is plugged so that her house is always at the perfect temperature.

Susanna also established specific times for the window blinds to open and close based on her preferences and the sunrise/sunset times, while having the ability to open or close them by pressing a button on her smartphone.

She got herself a smart front door lock that automatically unlocks when she gets in front of it by recognizing her phone and that she can also unlock it remotely if she ever need to let someone in while she's not at home.

Additionally, she put around the house some smart plugs that she uses to easily automate the charging or activation of non-smart devices by programming when the plug is turned on and off.

3.5 Chapter recap

In this chapter we analysed the Smart Home environment more in depth by overviewing its key functionalities, which are the main advantages that "smart" devices bring compared to a standard household. Then, we moved on analysing the most common types of the just mentioned smart appliances dividing them by category and citing their basic operations. Furthermore, we surveyed the main protocols used by smart devices to operate and communicate in a smart home network.

Chapter 4

Cybersecurity in Smart Home Systems

4.1 Smart Home security overview

Although a lot of benefits are gained from using Smart devices in residences, we must keep in mind that these Smart Home systems are susceptible to many different, and even new, attacks.

For example, now that houses are not entirely offline spaces but rather turning into more complex Cyber-Physical Systems, we need to consider the possibility of cyber enabled burglaries and residential crimes, with cyber-enabled crimes being defined as:

> *"traditional crimes, which can be increased in their scale or reach by use of computers, computer networks, or other form of information communications technology (ICT). Unlike cyber-dependent crimes they can be committed without the use of ICT"* [13]

IoT devices are known for being unsafe and, being built on minimal hardware, they often lack built-in security which means that vulnerabilities are often found. Therefore, it's clear that the rising number of IoT devices and Smart sensors being used in smart home applications leads to the systems being more vulnerable.

The security level of Smart Home devices and systems has been object of discussion for a long time, with the general sentiment being that the number of smart devices bought by users would keep rising. On the other hand, lately there have been growing security and privacy concerns coming from potential new users after multiple vulnerabilities exploitation made the headlines because of their severity and impact.

As previously mentioned in section 2.2, the most common and relevant security goals considered are Confidentiality, Integrity, Availability, Authenticity, Authorization, and Non-repudiation.

One example is the paper "Emerging Trends in Smart Home Security, Privacy, and Digital Forensics" in 2016 [14], in which researcher underlined five major aspects concerning the security and privacy of Smart Homes:

- Potential for remote security breaches
- Risks for smart home devices
- Privacy violations
- Infrastructure vulnerabilities
- Digital forensics challenges

The paper also identifies the pressing need for regulation and certification of smart home devices minimum level of security and data protection.

Additionally, due to the quick evolution of new technologies, policy makers are not able to identify IoT issues and propose adequate viable solutions quickly enough. This mean that often the improvement of user security and privacy is either absent or a trade-off for user experience.

4.2 OWASP Top 10 IoT Vulnerabilities

OWASP, the Open Web Application Security Project, is a non-profit whose focus is on improving software security. One of its more known projects is the OWASP Top 10, an up-to-date list of the ten most critical web application security risks, but in 2014 they also started the OWASP IoT project which aims at raising the awareness on the security issues associated with the Internet of Things to manufacturers, developers, and consumers.

This IoT project include an IoT version [15] of the OWASP Top 10, and in the last update from the 2018 these were the worst vulnerabilities identified:

1. Weak, guessable, or hard-coded password;
2. Insecure network services;
3. Insecure ecosystem services;
4. Lack of secure update mechanisms;
5. Use of insecure or outdated components;
6. Insufficient privacy protection;
7. Insecure data transfer and storage;
8. Lack of device management;
9. Insecure default settings;
10. Lack of physical hardening.

As we can notice a lot of the points on the OWASP's IoT vulnerability list match what previously mentioned when talking about the security concerns regarding Cyber-Physical Systems and Smart Homes. For example the use of default credentials or weak passwords, insufficient privacy protection, network insecurities, and lack of physical hardening.

This issues are often caused by a mix of negligence from both the end-users and the vendors in using or setting up the devices and flaws in the product design itself.

4.3 Attacks classification

One of the most common classification of attacks is the distinction between passive and active attacks.

The passive attacks can be described as attacks that attempt to gain information (and eventually make use of them) from the system without interfering with its functions and resources, in order to learn something on the system without being noticed. The active attacks are instead those whose purpose is actually affecting the system's resources and operations so they can imply modification of data or introduction of fraudulent data into the system.

We can further classify attacks based on the level in which they take place:

4.3.1 Physical attacks

Physical attacks involve gaining unauthorized physical access to the Smart Home system and its devices which could lead to theft or tampering of said devices. Some example are stealing a smart lock key fob, physically disabling a security camera, or messing with the wiring of the house.

4.3.2 Network-based attacks

Network-based attacks refers to attacks that jeopardize the network to which the Smart Home devices are connected, for example using a DoS attack to interrupt communications or hacking into the network to gain control of the connected devices.

Examples of network attacks are:

- **Denial of Service attack:** Commonly called DoS attacks, it's any attack that can disrupt the network services or devices; this can be accomplished by flooding the network or by crashing the services themselves.
- **Distributed Denial of Service attack:** A specific type of DoS attack that occurs when multiple actors are coordinated to attack a single target at the same time; an example is a botnet of infected nodes targeting the victim. The distribution of the malicious nodes allows greater volumes of traffic and more difficulties in stopping the attack.
- **Masquerading:** is an attack where an unauthorized entity is able to gain access to a system and its functions by posing as an authorized entity.
- **Eavesdropping:** as it can be deduced from the etymology of word, eavesdropping means being able to secretly listen to private communications happening in the network which allows the attacker to gather information and potentially capture sensible data.
- **Blackhole attack:** involves a malicious node using its routing protocol to aggressively publicize itself for having the shortest route to the destination node of a transmission; this allows him to set up a route where the source node sends him the packet and the malicious node can choose whether he wants to drop or forward this packet.
- **Wormhole attack:** it's an attack where two malicious nodes with better communication capacities position themselves strategically on the opposite sides of a network. The malicious nodes establish a connection and forward data from one end of the network to the other basically creating a tunnel over the existing network; as a result they can manipulate the routing algorithm since legit nodes may be tricked into thinking they are closer to other nodes than what they actually are.

4.3.3 Device-based attacks

Device-based attacks are those that have an individual device as the starting point. The most common scenario is finding and exploiting an appliance-specific vulnerability (e.g., a firmware vulnerability), but you could also capitalize on a malware-infected device to spread through the network.

Consequently, we can identify the main attacks as:

- **Malware:** when the device is infected with malicious software to gain control or steal sensitive data.
- **Physical Tampering:** as mentioned earlier, an attacker with physical access could directly target a specific device to compromise it or bypass security measures.
- **Rogue Device:** a rogue device attack happens when a malicious user is able to introduce a fake device into the network.

- **Ransomware:** it's a common attack where a device's contents and data are encrypted and a payment is requested as ransom.
- **Permanent DoS:** also called *Phlashing*, is an attacks that aims at incapacitating a device to the point it's rendered unusable until it gets either repaired or replaced.

4.3.4 Application-based attacks

Application-based attacks are similar to the device-based ones just mentioned, but a vulnerability in the mobile application or software used to control the device is exploited instead than one on the device itself.

We can therefore easily point out some well-know attacks such as:

- **SQL Injection:** notorious attack where malicious code is inserted through a SQL statement.
- **Buffer Overflow:** happens when an attacker sends more data than an application can handle, possibly making it crash or execute arbitrary code.
- **Remote Code Execution:** a remote code execution attack takes place when an attacker is able to exploit a vulnerability that allows him to execute arbitrary code.
- **Cross-Site Scripting (XSS):** it's the injection of malicious scripts into benign and trustworthy devices.
- **Cross-Site Request Forgery:** is an attack that forces the victim user to execute unwanted actions on a web application.

4.3.5 Social engineering attacks

Social engineering attacks include a vast range of malicious activities achieved through human interactions, used to trick a user into providing direct access or information useful to gain unauthorized access to the Smart Home or its systems.

Some examples of social engineering attacks are phishing, baiting, pre-texting, and any other strategy that can be used to deceive or influence people to involuntarily disclose sensitive information such as login credentials or access codes.

4.3.6 Supply Chain attacks

Also known as a third-party attack, supply chain attacks focus the logistics network behind the Smart Home systems; furthermore, the attacker tries to damage or gain unauthorised access to an organization by targeting less secure elements in the supply chain of other vendors.

This means that the attacker might be tampering with the firmware, hardware, or software of a device before it is even shipped to the end-users, for example placing a key-logger on USB drives will provide the attacker great amounts of sensitive information from the company that buy these USBs.

4.3.7 Side-channel attacks

As defined in the NIST glossary a side-channel attack is:

> *"An attack enabled by leakage of information from a physical cryptosystem. Characteristics that could be exploited in a side-channel attack include timing, power consumption, and electromagnetic and acoustic emissions."* [16]

To mention a few different vectors used to perform a side-channel attack, we have the following types:

- Electromagnetic
- Acoustic
- Power
- Optical
- Timing
- Memory cache

4.4 Historical attacks

4.4.1 Mirai

When talking about the lack of security in IoT devices it must be mentioned the Mirai malware that was used to attack the 'KrebsOnSecurity' website in September 2016, and subsequently in other attacks. The attack on Krebs' website exceeded 600 Gigabits of traffic per second in volume, while the later attack on the hosting company 'OVH' reached one Terabit per second.

These unprecedented traffic rates were obtained through the botnet that Mirai created, compromising hundreds of thousands of IoT devices. As we mentioned already, the advent of IoT has created a perfect playground for cyber criminals due to the lack of security on IoT devices, even the edge ones. Mirai infected mainly internet connected routers, security cameras, digital video recorders, and printers. After infecting a new device Mirai proceeds to scan the available networks looking for vulnerable devices trying a set of default credentials. Using this approach Mirai was able to infect over 65 thousand devices by the end of the first 24 hours of attack, eventually reaching two or three hundred thousand infections.

Even though this kind of DDoS attacks do not directly target the owners of the devices, using vulnerable appliances is something that anyone should try to avoid. Additionally, a compromised device that is actively being used in a botnet will at least appear slower to the user, if not entirely unusable.

4.4.2 TRENDnet Webcam Hack

The TRENDnet Webcam Hack is instead an example of an attack that directly affects the device owners by bringing to the table serious safety and privacy concerns. It all began when a user of the "consolecowboys" blog posted about a security vulnerability he had found out and successfully exploited on a TRENDnet Camera [18] that allowed him to watch the feed from the camera by making a web request to the IP address of the camera plus the fixed path *"/anony/mjpg.cgi"*.

There was no way to stopping this feed from being live-streamed, even after setting up security credentials for the camera; he was also able to take things a step further utilizing Shodan, a search engine for Internet-connected devices, where he was able to find the addresses and access more than three hundreds devices.

This vulnerability allowed anyone with the IP address of a camera to look at the feed and maybe even listen to the microphone. In an old press release by TRENDnet itself [19] we can see them admitting the vulnerability and taking action to correct and publish updated firmware, along with a list of all the cameras affected.

Furthermore, according to this FTC's case (Federal Trade Commission) TRENDnet transmitted user login credentials in clear, readable text over the internet, and stored users' login credentials in clear, readable text on their mobile devices.

4.4.3 ZigBee Chain Reaction

The "ZigBee chain reaction" [17] is the name researcher gave to the implementation of a new vulnerability that they found in the widespread Philips Hue Smart-Lightbulb which allowed them to create an IoT worm; this is a relatively new type of threat in which neighbouring IoT devices will infect each other. This means that in the current world where IoT devices are getting more and more common, if the density of compatible IoT devices exceeds a certain critical mass the malware will spread explosively over large areas. Additionally, the worm doesn't require a connection to an internet network but is able to spread from one lamp to adjacent ones using only the built-in ZigBee wireless connectivity and the physical proximity.

The Hue lightbulb contains a ZigBee chip made by Atmel, which uses multiple layers of protection to protect the lamp once it is connected to the controller, particularly it will ignore any request to reset or to change affiliation unless it is sent from a ZigBee transmitter in close proximity of the lamp. The first discovery made by the researchers was a major bug in the Atmel stack's proximity test that allowed any standard ZigBee transmitter to dissociate lamps from their current controller by starting a factory reset; once they are dissociated the transmitter can send additional instructions to fully take control of the lightbulb.

Due to the low power consumption, contained weight, and small dimensions of the transmitter and the automatability of the attack the researchers were able to create a portable autonomous attack kit. This kit was subsequently used to perform a 'war-flying' attack, meaning that they mounted the attack kit on a drone and executed and executed the attack starting 350 meters away from the target building by driving the drone close.

Then, looking to scale up the attack and cause permanent damage to the devices, the researchers focussed on reverse engineering the process used by Philips to perform firmware updates and within a few days were able to deduce all the secret cryptographic elements used (such as IV and key).

Having extracted the global keys Philips used to encrypt and authenticate new firmware, the researchers were now able to load any malicious over-the-air firmware updates, demonstrating how a low-budget attack could have a devastating effect. Plugging in a single infect lightbulb with modified firmware can start a chain reaction in which each lamp will infect and replace the firmware of all the neighbours. From this point the attacker can freely modify the lights to his liking, exploit them for a large-scale DDoS attack, or even permanently brick all the lightbulbs by disabling their firmware update process, effectively rendering the lamps useless.

4.4.4 BrickerBot

In April of 2017 the cyber-security firm Radware detected through their honeypot servers a new malware strain named BrickerBot and recorded almost two thousands PDoS attempts over a four-day period. Subsequently, they published an attack report with information on the discovery and analysis of this new Permanent DoS tool. [20]

BrickerBot targeted mainly Linux BusyBox-based IoT devices and spread to devices that had open Telnet ports and then, similarly to Mirai and Hajime (and other IoT malwares), tried to access the device configuration by trying a list of known default credentials. If successful in accessing the device the malware would then execute some commands to:

- Write randoms bits to corrupt the storage drives.
- Disable TCP timestamps to disrupt Internet connectivity.
- Set the maximum number of kernel threads to one (the default value for ARM-based devices is in the tens of thousands) which effectively cuts down the performance.
- Wipe all the files on the device.
- Reboot the device.

This results in the IoT device being "bricked" merely a few seconds after infection, resulting in the need of the device being reinstalled or replaced altogether as the malware could even rewrite the firmware. There is a second version of BrickerBot with extra commands to flush all the IPtables, firewall, and NAT rules and drop all outgoing packets.

The author of this malware explained that he created BrickerBot as a way of fighting the constant growing botnets made by IoT devices (increased further by the release of Mirai's source code online) by disabling vulnerable devices before they end up as part of a DDoS botnet. In December of 2017 the author announced the retirement of BrickerBot claiming to have bricked over 10 million devices since he started this project he self-described as "Internet Chemotherapy" in November 2016. [21]

4.4.5 Other Examples

There are many other examples of attacks on a large range of devices, we are now going to mention a few more:

- **Nest Thermostat Hack:** At the Black Hat 2014 security conference a group of hacker demonstrated how they were able to compromise a Google Nest Smart Thermostat. The attack consisted in a physical jailbreak, so they were not able to implement a remote hack yet, but the device completely lacked a chain of trust which meant that they could reprogram to the point that it would share data with both the attacker and the user without the latter knowing it.

- **Smart Refrigerator Threats:** As we read in the research published by S. De Bique [22] because of the lack of integrated security smart refrigerators are a common target for botnets. For example, in 2017 a vulnerability was found in the LG IoT product line called SmartThinQ which attackers could exploit to infiltrate by modifying the appliances' Linux application. In 2015 it was discovered that the Samsung's smart refrigerator technology did not authenticate the Secure Socket Layer (SSL) certificates when connecting to remote servers, so anyone on the network could freely access the users' credentials.

- **Smart TVs Hacks:** As a result of being internet-connected to be "smart" and provide services even TVs are vulnerable to remote attacks. In 2013 researcher from the University of Amsterdam studying a Samsung Smart TV were able to impersonate a Samsung's update server in order to perform an online firmware upgrade; additionally, they discovered some vulnerability in the TLS/SSL implementation. [23]

- **Ring Video Doorbell Hack:** In 2019 in the Anti-Malware research section of the Bitdefender blog was published a whitepaper analysing a vulnerability discovered in the Amazon's Ring Video Doorbell Pro. While in the configuration stage, the doorbell camera receives the home's network (Wi-Fi) credentials from the smartphone app in plain HTTP, so an attacker physically close to the device could capture the credentials and scale the attack to a larger one against the network. [24]

- **August SmartLock Hack:** A few month after the Ring Doorbell one, Bitdefender disclosed a very similar vulnerability this time affecting the August SmartLock. In this case during the configuration phase the device and the application exchange encrypted messages, but the encryption key is hardcoded into the application. The whitepaper references the CVE-2019-17098 which underlines the use of hard-coded cryptographic key vulnerability. [25]

- **Foscam IP cameras vulnerabilities:** In 2017 researcher from F-Secure Blog published a report in which they point out eighteen vulnerabilities found in multiple models of Foscam cameras, affecting thousands of internet-connected cameras. The vulnerabilities were the consequence of eleven CWE present in the devices, a remarkable number that as the report underlines allows attacker to gain full control of the cameras. [26]

4.5 Impact analysis

The impact of attacks on a Smart Home system can be subdivided in two main categories: physical impact and cyber impact. Additionally, the Quality-of-Life of the user is certainly going to be affected by attacks so we can also classify the impact on domestic life.

Integrating with some definition given by *Heartfield et al.* [27] we can define and briefly explain some attack consequences.

4.5.1 Physical Impact

From the following list we can see how the physical consequences of being the target of an attack are mainly loss of physical privacy and interferences with the system's operation.

- **Breach of Physical Privacy:** any kind of attack that results in the malicious actor gaining information on the system is a breach a of physical privacy, whether this mean just being able to log the homeowner habits or actually obtaining video/audio recordings or live feeds from the house.
- **Delayed Actuation:** it happens when an attack affect the availability of the network of an household which can cause delayed transmission on commands or queries and consequently delayed actuation.
- **Incorrect Actuation:** a small scale example of incorrect actuation could be an attacker interfering with the temperature readings from sensor so that the smart thermostat continuously tries to fix it, wasting energy and possibly overheating a room.
- **Prevented Actuation:** happens whenever the effect of a system's action is completely blocked either intercepting and dropping the packets or by tricking the controller into thinking that the action cannot take place.
- **Unauthorised Actuation:** any situation where the attacker is able to hijacking the controller of a Smart Home and for example switch the lights off or unlock a door.

4.5.2 Cyber Impact

- **Confidentiality:** it's often violated by any attack that is able to obtain access to secret information, such as passwords or pin codes.
- **Integrity:** many security threats will compromise the integrity of a smart home system since they involve unauthorised manipulation of data.
- **Availability:** it's commonly disrupted by DoS and jamming attacks, which can render devices incapable of transmitting and receiving data or information.
- **Non-repudiation:** it's linked to a user's ability of providing evidence that can legitimate his activities, compared to the activities of a malicious actor.

4.5.3 Impact on Domestic Life

Any of the beforehand mentioned threat can surely disrupt the normal course of domestic life, and the consequences can be classified in three main categories:

- **Direct:** it covers some of the most evident consequences such as inconvenience, loss of control, safety, invasion of privacy, and financial.
- **User Experience:** from the perspective of the user experience an interference could be instantly noticeable, noticeable over time, or non-noticeable.
- **Emotional:** which is any consequence that may emotionally affect an user after a violation of the cyberspace of his own home.

4.6 Possible countermeasures

The focus of this thesis is not researching how to secure a Smart Home system, but we are still going to briefly mention the key aspect of securing these kinds of systems in order to add some context.

To try and mitigate the numerous threats that we discussed in the previous sections one should starts from basic things such as not using default password or making the home residents aware of security risks; then you can move up the scale of best practices until you get to the more specific ones.

To cite a few of the suggested best practices we have:

- Use strong passwords
- MFA
- Access limitation
- Disable unnecessary features
- Keep devices up-to-date
- Isolation
- Encryption
- Security monitoring and analysis

For a more precise classification of security countermeasures by goal, we can refer to the review made by *Komninos et al.* in the *"Survey in smart grid and smart home security"* summarised in the following table 4.1. [28]

As it always happens in the cybersecurity field, it's almost impossible to obtain a solution that guarantees complete safety, but the mentioned countermeasures are a good way to reduce the exposure to threats and limit the risks and consequences.

4.7 Chapter recap

In this chapter we proceeded to analysing and research the Smart Home systems from the cybersecurity perspective. This wad done by firstly providing an overview of the possible security issues and of the OWASP Top 10 IoT vulnerabilities. Then, a classification of the different types of possible attacks was presented along with a list of notable past attacks and the explanation on how they worked and what effect they had. Additionally, we analysed the different kind of impact that such attacks could have on a Smart Home and its users, categorizing them by physical, cyber, and domestic life impact.

Confidentiality and Privacy
• Symmetric/Asymmetric encryption algorithms • Anonymization • Trusted aggregators • Homomorphic Encryption • Data obfuscation • Verifiable computation models
Integrity
• Cryptographic hasing techniques (e.g., SHA-3) • Digital watermarking • Adaptive cumulative sum algorithm • Load profiling • Time-stamps • Sequence numbers • Session keys • Nonces
Authenticity
• Keyed cryptographic hash functions (e.g., HMAC) • Physically unclonable functions • Hash based authentication codes • MAC-attached and HORS-signed (Hash to Obtain Random Subset) messages
Availability
• Anomaly based IDS • Specification based IDS • Alternate frequency channels
Authorization
• Attribute based encryption • Attribute certification • Attribute based access control system

Table 4.1. Review of Security Countermeasures by Goal. [28]

Chapter 5

Simulation Tools

After having looked thoroughly at the protocols used and some of the possible attacks and their real-world consequences it's easy to realize how fundamental security and reliability are in the Smart Home environment.

This brings us to the need of simulation tools to test the security of new devices or to verifying the compatibility and reliability without having to construct an entire real-world CPS which could be a great cost in terms of time and resources. Unfortunately, the tools for the Smart Home systems seem not to be as common and developed as other CPS such as the Industrial and the Mobility ones.

Nevertheless, we will go through some of the more generic simulators as well as some more context-specific ones.

5.1 Simulators overview

- **Cisco Packet Tracer:** is a Cisco proprietary networking software which aims at being a learning tool through a combination of simulation and visualization activities. It a strong tool for student to prepare for Cisco CCNA certifications. [29]

  Pros:

  - Very easy to learn and use.
  - Offers, although pretty simplistic, a native Wi-Fi simulation which is not a common feature in simulators.
  - Provides a good overview of the devices and their statuses, allowing centralised control (e.g., from the central hub).

  Cons:

  - It has been developed in an educational scope.
  - The only available devices and nodes are a subset of Cisco proprietary ones, so it's pretty restrictive.

- **COSSIM Framework:** COSSIM ("Novel, Comprehensible, Ultra-Fast, Security-Aware CPS Simulator") is an open-source framework that aims at simulate both the networking and processing parts of the CPS, perform fast simulations, and provide accurate results. [30]

  Pros:

  - Open-source tool.

  Cons:

  - Not recently maintained.

- **EVE-NG:** stands for Emulated Virtual Environment - Next Generation and it gives you tools for virtual devices and their interconnection with other virtual or physical devices. It provides a free community edition, or a commercial Professional license and it you can manage the virtualized environment via an HTML5 client, but the setup is not that fast and easy. [31]

  Pros:

  - Open-source tool.
  - Allows realization of complex topologies divided in multiple areas.
  - Allows Wireshark use to inspect packets.

  Cons:

  - You need to deploy a pretty heavy VM.
  - You need to manually add the image of every device you may want to use.
  - Setting up extensive topologies is not fast by any means.

- **GNS3:** is an open source, actively developed, free software that consist of a all-in-one GUI that can be used to create topologies and a VM that act as a server. [32]

  Pros:

  - Open-source tool.
  - Widely adopted, has a good base of appliances that can be installed as nodes.

  Cons:

  - The VM can be quite resources consuming.
  - Hard to add any type of node not available on the GNS3 appliances list.

- **IoTIFY:** is a cloud native IoT simulator designed to simulate and test Internet of Things devices and applications at a global scale. It is more suited for development testing than attacks simulation. [33]

  Pros:

  - Large-scale deployment support.

  Cons:

  - Not an open-source tool, need a license to be used.
  - Hard to represent real-world devices and their behaviour.

- **IoTSecSim:** is a framework for modelling and simulation of security in Internet of Things built entirely in python. It allows setting up IoT attacks and defences in an emulated IoT network and the subsequent evaluation of the impact through multiple security metrics. [34]

  Pros:

  - Open-source tool.
  - The network has different settings and configurations. (e.g., topology, n. of attacker, defence system)
  - Offers graphical representation of the simulation and statistics.

  Cons:

  - Does not aim at representing an exact type of device.
  - The devices node cannot be personalised.

- **MiniCPS:** is a framework for Cyber-Physical Systems real-time simulation built on top of mininet and provides support for physical process and control devices simulation and network emulation. [35]

  Pros:

  - Open-source tool.
  - Aims at precisely simulate CPS systems.

  Cons:

  - Scarce documentation, the only example is of a water treatment system.
  - Generally seems more aimed at industrial cyber-physical systems.

- **ns-3:** is an open-source software, actively developed, discrete-event network simulator for Internet systems used for educational and research purposes. [36]

  Pros:

  - Open-source tool.
  - Presents a decent amount of protocols/devices implementations along its modules.

  Cons:

  - Not many examples can be found as references.
  - Cannot create an exact virtual copy of a device, but you can set up nodes with similar protocols.

- **OMNeT++:** is an extensible, modular, component-based C++ simulation library and framework, used primarily for building network simulators. It does not natively support network protocols, but they can be added through frameworks such as INET, an open-source model suite for wired, wireless, and mobile newtorks. [37]

  Pros:

  - Open-source tool.
  - INET use provides many useful models, such as Wi-Fi, 802.15.4, and AODV routing.

  Cons:

  - Cannot create an exact virtual copy of a device, but you can set up nodes with similar protocols.

- **UCEF:** stands for Universal CPS Environment for Federation and is an open-source tool kit developed by the National Institute of Standards and Technology (NIST) that provides a collaborative experiment-development environment. [38]

  Pros:

  - Open-source tool.

  Cons:

  - Not well maintained.
  - More focused on Federated testbed architectures.

We are now going to take a closer and better look at the tools more extensively used during this research work.

5.2 Omnet++

Although OMNeT++ is not a network simulator itself, it can be used as a network simulation platform and so is one of most widespread tools. It is self-described as:

> *"..**an extensible, modular, component-based C++ simulation library and framework, primarily for building networks simulators**. "Network" is meant in a broader sense that includes wired and wireless communication networks, on-chip networks, queueing networks, and so on. Domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modelling, photonic networks, etc., is provided by model frameworks, developed as independent projects."* [37]

The main components are:

- C++ kernel simulation library
- The NED topology description language
- Simulation IDE based on the Eclipse platform
- Interactive simulation runtime GUI
- Command-line interface for simulation execution
- Sample simulations

During the years OMNeT++ has been available multiple simulation models and example frameworks have been created by researchers of different topics such as internet protocols, switched LANs, mobile ad-hoc networks, wireless sensor networks, etc.

Even though many of these projects are now discontinued or no more maintained the majority is open source and they can be found listed in the OMNeT++ models and tools webpage.

Among them we have the INET Framework which can be considered the standard protocol model library for OMNeT++. It's an open-source framework maintained by the omnet team suited for wired, wireless, and mobile networks by providing models for the Internet stack and many other protocols.

To build a simulation you firstly need a OMNeT++ model built by components which can exchange messages; modules can also be nested to form a compound module. The active components of the model have to be programmed in C++ using the simulation kernel and class library.

Then the module structure has to be described with the NED either through the source code or via the graphical editor. Finally a *omnetpp.ini* file is needed with the configuration and parameters of the simulated model.

5.3 ns-3

The ns-3 project aims at developing a preferred, open simulation environment for networking research and consequently, the ns-3 tool is described as a discrete-event network simulator publicly available for research, development, and use.

Ns-3 is used to create multiple virtual nodes on which you can install different devices, internet stacks, applications, and others.

Additionally, we have other features such as tracing of the nodes, network visualization through the NetAnim tool, generation of a pcap file (that can be analysed with Wireshark), and gnuPlot to plot data from the trace files.

In the ns-3 doxygen documentation webpage we can fin a list of all the modules that are implementable with the ns-3 library such as:

- 6LoWPAN
- AODV Routing
- CSMA Network Device
- Energy Models
- Internet (containing the basic IP protocols such as IPv4, IPv6, TCP, UDP, etc)
- Mobility
- Network Animation
- Point-to-Point Network Device

and many more.

5.4 GNS3

GNS3 is the shortened name for Graphical Network Simulator-3 and is an open source software that allows the emulation of complex network designs.

GNS3 is is helpful in designing, building, and testing a network in a risk-free virtual environment; some key features being advertised on the official website are:

- Real-time network simulation for pre-deployment testing without the need for network hardware
- Test 20+ different network vendors in risk-free virtual environment
- Create dynamic network maps for troubleshooting and proof of concept (POC) testing
- Connect GNS3 to any real network
- Customized topologies and labs within GNS3 for network certification training

and many more.

Differently from ns-3 and OMNeT++, which are script-based simulators, GNS3 allows the execution of actual firmware on simulated devices for example Cisco routers.

GNS3 consists of two main software components:

- The GNS3-all-in-one software (GUI)
- The GNS3 virtual machine (VM)

The first one is just the graphical user interface that can be used to install appliances, set up devices, and establish connections thus creating a topology. As for the servers on which the simulated devices runs there are three possible option:

- Local GNS3 server
- Local GNS3 VM
- Remote GNS3 VM

The recommended option is running a local VM so that you have full control over it but any additional process (e.g., Dynamips) will not be installed on the main host. This can be done with a virtualization software such as VMware or Virtualbox, with the former being the recommended one.

5.5 EVE-NG

EVE-NG stands for *"Emulated Virtual Environment - Next Generation"* and is an established simulation tool used by Network, Security, and DevOps professionals and researchers.

EVE-NG is considered one of the best tools for the design, efficiency, and flexibility offered when simulating a network, some of the highlighted features are:

- Clientless management, instead uses telnet, rdp, vnc over HTML5
- KVM hardware acceleration
- Local client Wireshark capture
- "Click and play" topology designer
- Custom Kernel Support for L2 protocols
- Memory optimization (Ultra Kernel Samepage Merging)
- CPU Watchdog
- Interaction with real network fully supported
- Supports up to 1024 nodes per lab

and many more.

EVE-NG has a free community edition and a professional paid edition. The community edition has only the very basic features but it's enough to test the tool and create some modest simulation models.

Simulating with EVE-NG can be really resource demanding, the minimal systems requirements for the EVE virtual machine are 1 core with 4 processors, 6Gb of ram, and 40Gb of disk space while the actual recommended requirements are 8 processors, 24Gb of RAM, and 200Gb of disk space. This means that the system that hosts the VM has to be quite powerful.

One of the advantages offered by EVE-NG is the possibility of deploying the VM remotely on the Google Cloud Platform and the installation process is well document in the official Community Cookbook. Obviously, Google Cloud is a paid service but it allows for a smooth deployment of the EVE VM on an adequately strong system.

Even though in the next section we are not going to present a simulation built using EVE-NG the analysis of the tool it was only fair to present the tool anyway as it was deeply researched and tested before ultimately being dropped in favor of GNS3.

The two tools are fairly similar from a conceptual perspective and EVE-NG provided the mentioned possibility of installing the EVE VM on Google Cloud which we were able to set up correctly and it's a big advantage to the local GNS3 VM in terms of the virtual machines resources. This was enough to offset, initially, the longer installation process needed to add device's templates which required to connect to the Virtual Machine via ssh or telnet console to upload the images files and correctly install them.

Unfortunately, it is harder to find free open-source devices for building the topology and we also ran into some problems with some installed nodes not running properly, perhaps for lack of maintenance or old deprecated images not having an updated version. For example we downloaded an older free version of Cisco's vWLC (virtual Wireless LAN Controller), followed the official EVE-NG instructions for its installation without running into any problem, but then the router would not boot, no matter the combination of physicals or virtual settings that we tried.

So, we decide to move forward with the GNS3 simulation.

5.6 Chapter recap

In this chapter we first provided an overview of the various existing simulators tools along with some *pros* and *cons* for each one of them. Then, we moved onto the ones that were actually used for the simulations implemented for this thesis and a deeper analysis was made, by looking at the key features and how they could be implemented.

Chapter 6

Simulated scenarios

6.1 Omnet++ simulation

This section is going to describe the simulation developed with the OMNeT++ tool: the version used was the latest one available, release *OMNeT++ 6.0.1* and it was installed on a native Windows 10 machine; the fundamental INET framework was also the latest one: *INET-4.4.1*.

6.1.1 Battery depletion attack

A battery depletion attack aims at draining the battery life of a device. One of the main issues of WSN (Wireless Sensor Technology) or generally speaking IoT devices is the limited physical resources, including the battery capacity.

This can leave the systems exposed to battery-exhaustion attacks, perhaps coming from active intrusions or protocol failures, and once some of the devices constituting the network fail because of low battery levels the entire system operation could become compromised.

Simulation

A good example for this attack in a Smart Home is a lighting or heating system which uses external battery-powered sensors, so the basic INET visualization model of a Smart Home network was modified to a topology matching our needs which contains a central controller node, multiple Smart-Lightbulbs around the house, and some sensors with a 1:N proportion to the lights. The controller and the lightbulbs are plugged into a power source, while the sensors are not.

A mock hypothetical communication model was set up, where the sensor periodically send data to the controller which process the information and sends control packets to the lamps. The simulation was run for a 100 seconds period and this caused an average drop of battery charge of 4% in the sensors.

Then we introduced the malicious node involved in this attack, which was assumed to be a sensor to which the attacker was able to gain access: this infected node was spamming one of the sensors with ping messages, and with the network being Wireless every sensor was receiving this message. We were able to observe how, over the same 100 seconds timespan, the battery of the target sensor dropped 23% while the other sensors, which didn't have to completely compute and answer to the ping message, lost 7% of battery charge.

We chose a single target to the Ping application to notice the difference in battery-exhaustion between the target and the "bystanders" of the network, yet the latter almost doubled in power consumption. Clearly an implementation of this attack where the attacker manages to compromise more than one node and starts targeting every node in the network would quickly drain the battery charge of every device.

Implementation

To implement the attack was firstly necessary to include the power consumption element which was done through the *power* INET model which was used to set up the sensors' energy storage, management and consumer elements. A new sensor module was created which used the *SimpleEpEnergyStorage* which maintains a residual energy capacity by integrating the difference between the starting capacity and the consumed power.

The starting battery charge of the sensors was randomized to be between 10% and 100%, and the consumption values were set up close to the standard ones, as we can see from this code snippet:

```
*.sensor*.energyStorage.nominalCapacity = 5J
*.sensor*.energyStorage.initialCapacity = uniform(0.1 * nominalCapacity,
    nominalCapacity)
*.sensor*.energyManagement.nodeShutdownCapacity = 0.1 *
    parent.energyStorage.nominalCapacity
*.sensor*.energyManagement.nodeStartCapacity = 0.5 *
    parent.energyStorage.nominalCapacityT
*.sensor*.wlan[*].radio.energyConsumer.sleepPowerConsumption = 0.1mW
*.sensor*.wlan[*].radio.energyConsumer.receiverIdlePowerConsumption = 2mW
*.sensor*.wlan[*].radio.energyConsumer.receiverBusyPowerConsumption = 5mW
*.sensor*.wlan[*].radio.energyConsumer.receiverReceivingPowerConsumption =
    10mW
*.sensor*.wlan[*].radio.energyConsumer.transmitterIdlePowerConsumption = 2mW
*.sensor*.wlan[*].radio.energyConsumer.transmitterTransmittingPowerConsumption
    = 100mW
```

On the attacker side, a new sensor model was created to describe an infected malicious sensor with an ideal infinite power source (plugged in) called *IdealEpEnergyStorage* and then his behaviour was set up using the INET *PingApp* model to flood the target sensor with a frequency of one packets every millisecond which was done through the following line of code: `*.infsensor.app[0].sendInterval = 0.001s`.

In figure 6.1 we can see the simulation environment before the attack starts and the specific battery level of three sensors. Then in figure 6.2 we have the same view at the end of short attack simulated and it's easy to notice the battery level drops as well as the devices marked by a red cross which have reached the minimal battery percentage for operating.

Analysis

Omnet++ proved itself as a good tool to run experiments on battery powered devices as it has different settings for the Energy Storage, Capacity, and Consumption models. Additionally, the physical details of Wi-Fi and other wireless protocols can be tweaked pretty extensively.

The main downside is perhaps not being able to customise the networks element after a certain point, in the sense that you can build a new "object" module from scratch or integrating other modules already provided by the Omnet++ and INET community, but you cannot insert the virtualization of real life device.

6.2 ns-3 simulations

This section is going to describe the simulations developed with the ns-3 tool. The version used was the latest one available when the research work started, release *ns-3.37*; it was installed on a Debian based Linux virtual machine.

Using ns-3 it's pretty straightforward, the simulations are implemented through C++ files used to describe the topology, the protocols in use, and the behaviour of the actors of the simulated

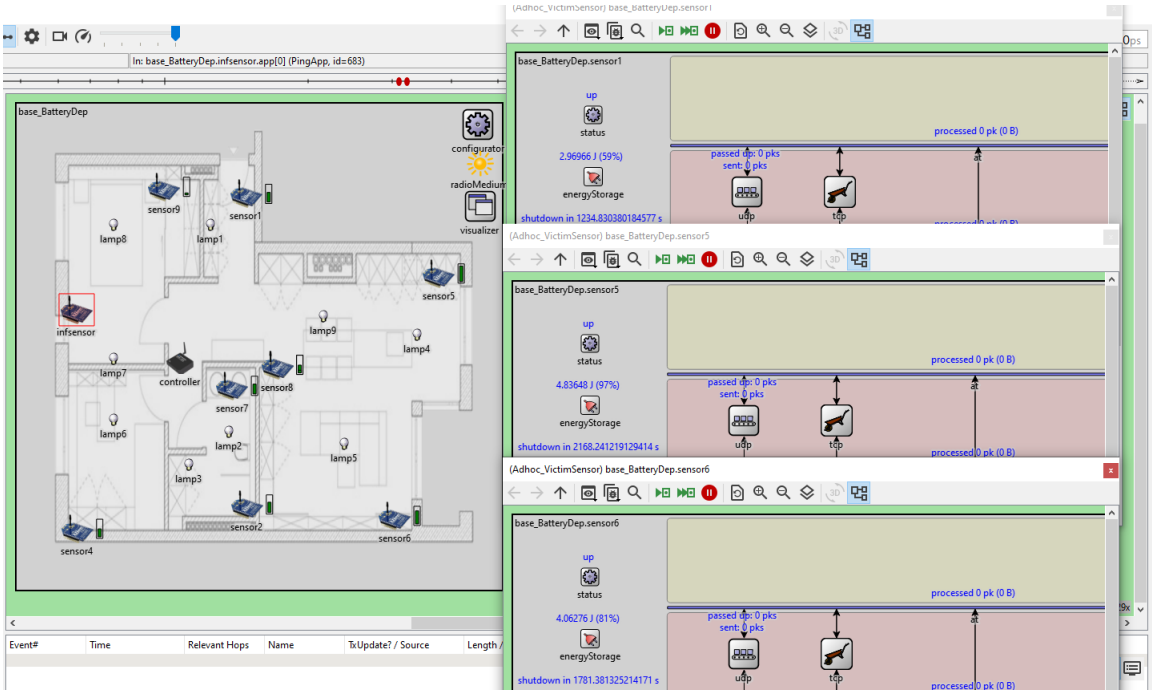

Figure 6.1. Screenshot of the battery levels before the attack starts.

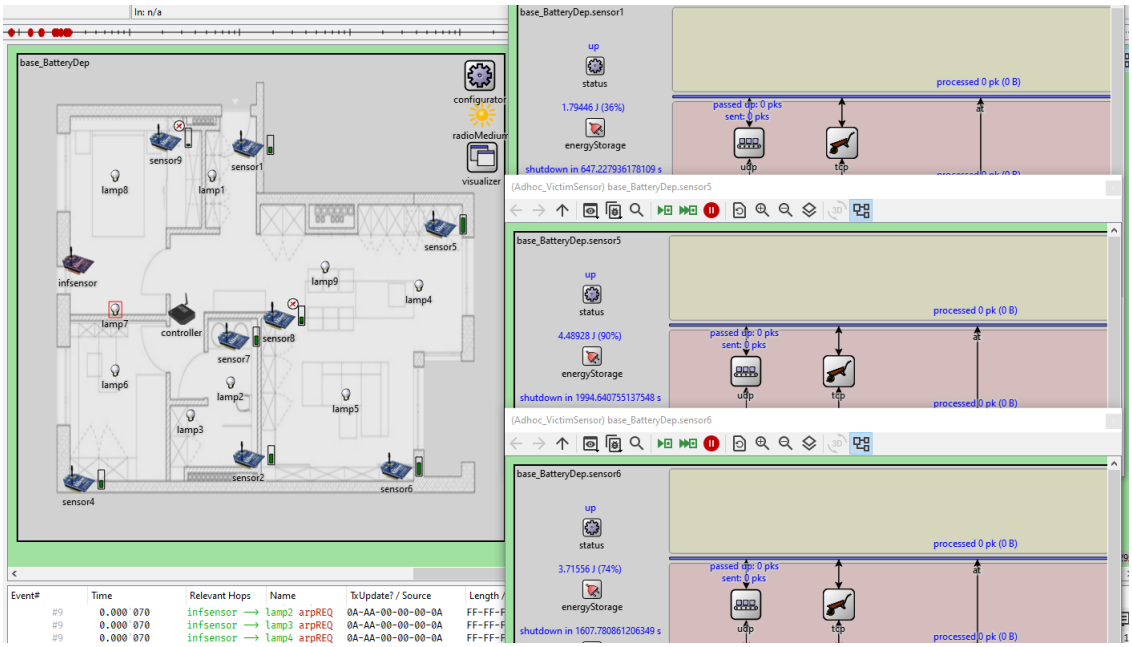

Figure 6.2. Screenshot of the battery levels when the attack ends.

environment. This C++ file must include all the modules, provided by the ns-3 tool itself, which describe the elements used by the simulation, for example this is one of the many includes used: `#include "ns3/wifi-module.h"`.

This C++ files then have to be placed in the *../ns-3.XX/scratch* directory so that we can now run it with the simple command

```
./ns3 run ExampleFile
```

If the program needs some argument parameters then the syntax is going to be

```
./ns3 run ExampleFile --parameterName=value'
```

with the command line parameters being parsed later during the program execution.

6.2.1 DoS attack

As mentioned in section 4.3 a DoS is an attack which aims at interfering with a system to the point that it's not able to operate normally nor be available for its legit users.
PaloAlto Networks, a global cybersecurity leader, defines a DoS as:

> *A Denial-of-Service (DoS) attack is an attack meant to shut down a machine or network, making it inaccessible to its intended users. DoS attacks accomplish this by flooding the target with traffic, or sending it information that triggers a crash. In both instances, the DoS attack deprives legitimate users of the service or resource they expected.* [39]

So we decided to implement a basic traffic flood DoS attack in ns-3. This requires to first build a simulation of a network of wireless devices, which for example could be a WSN (Wireless Sensor Network) implemented in a Smart Home, where multiple sensor provides information to a controller through the WiFi protocol. This controller then could have to forward this information to another central node such as the home gateway, perhaps if the processing stage happens in the cloud.

If then a malicious node is placed in this environment it can flood the controller node disrupting its operations; the malicious node could for example be a legit node of the system compromised by an attacker through the exploitation of a device vulnerability.

Simulation

This simulation includes six legit nodes communicating wirelessly with a node that acts as the controller (SmartHub) and is then connected to a central node (e.g., the Home Router or Gateway); we then have the additional malicious node connected to the Wi-Fi channel.

The legitimate nodes are spread around the network and periodically communicate with the SmartHub, which in this case relays the message to the central router through its channel; the malicious node is instead flooding the SmartHub node with UDP packets, causing delays in the central node answers reaching the legitimate nodes.

Implementation

To implement this topology the ns-3 *Node, NetDevice, InternetStack, and IPv4Interface* modules were used together with the Wi-Fi channel *YansWifiPhy* class which implements a model of 802.11a. Everything had to be set up properly including the WiFi MAC, ssid, interfaces, and address space.

To visualize the simulation the AnimationInterface tool was used together with a mobility model, specifically the *ConstantPositionMobilityModel* was set which means that the devices are static and then arranged with the *UniformDiscPositionAllocator* used to position the router at the center of the network with the SmartHub immediately close by and the other node spread around it.

To simulate the periodic communication between the legitimates nodes and the router a *UdpEchoClient* application was installed on every node with the destination being the *UdpEchoServer* installed on the receiving node. On this "router" node was also installed a *UdpSink* application to match and stop the flood of UDP packets sent by the malicious node which was instead using an *OnOff* application to constantly transmit trough the *ns3::UdpSocketFactory* to the corresponding socket on the sink.

As highlighted by figure 6.3 when the attack start the malicious nodes start sending the UDP packets to the SmartHub (node 0) and we can quickly notice the channel from the SmartHub and the router (node 0 to node 1) getting flooded; being a Wi-Fi channel we can see the packets also being transmitted to every other node in range on the wireless channel.

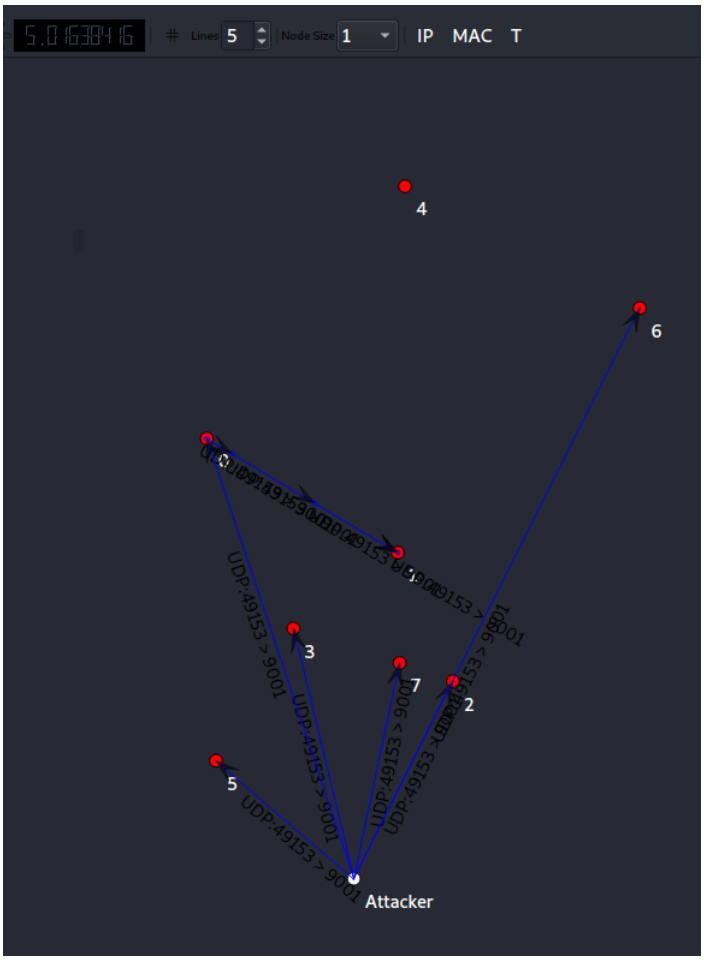

Figure 6.3. Screenshot of the attack simulation

Analysis

This example simulation allowed a precise representation of a WiFi network with a topology similar to what could be found in a home environment; it was also effective to visualize the attack flow and highlight the effects that a simple DoS can have on the legit communications of a network.

As previously seen in omnet++, there is no possibility of inserting in the simulation of an exact copy of existing devices, what it allows us to do is creating new modules with a behaviour similar to the one of the devices that we would like to simulate.

6.2.2 DDoS attack

A DDoS is a Distributed Denial of Service attack which occurs when a DoS attack is orchestrated such that multiple sources attack a common target in a simultaneously. Having this distribution of hosts that are used in a DDoS provide the attacker multiple advantages:

- Leverage over greater volume of machine to which can generate a more disruptive attack

- Source location of the attack is harder to detect due to possible distribution of attacking systems
- Harder to prevent and contain an attack coming from multiple machines
- The real attacking actor is harder to identify, as its disguised behind these distributed systems

and more. [39]

Simulation

This time the simulation aims at representing an attacker being able to compromise multiple nodes in a SmartHome system (i.e., an attack spreading as the one mentioned in section 4.4.3) and use them to completely block the operation of a system by flooding its central nodes.

The network is composed by three legit nodes which represents again a SmartHub, a home central router or gateway, and a normal node, respectively named node 1, 2, and 0. The standard node could be a control device such as a Smart Thermostat or a Keypad trying to communicate with the other two, whether it's to receive instruction from the Hub or to from the user through commands coming remotely via internet connection.

Once again this node tries to communicate while the malicious nodes act as sources of the DDoS attack flooding the connection channel between the Hub and the router.

Implementation

To implement this topology the ns-3 *Node, NetDevice, InternetStack, and IPv4Interface* modules were once again used but for the sake of visualization clarity, because of how the visualization tools represents WiFi packets, in this simulation we used point-to-point connections between every node (which is still a possible configuration for a Smart Home system) but the concept would work in the same way in a complete wireless network with the proper adjustments.

The legitimate nodes communication was emulated with TCP packet transmission so this time the end destination node (number 2) implemented both a *UdpSink* application and a *TcpSink* one. The *OnOff* application was once again used to flood the target with UDP packets and this time was installed on every single malicious node, starting at the same time.

To visualize the simulation the AnimationInterface tool was used again used with the *ConstantPositionMobilityModel* model, but this time the devices were arranged with the *GridPositionAllocator* and set up so that we have on the first row the three legit nodes and on a second row the malicious ones.

As we can see in figure 6.4 the legitimate node 0 is sending TCP packets to the destination (node 2) passing through the SmartHub (node 1) but the communication channel between node 1 and 2 has already been flooded by the attackers spamming UDP packets, causing a big delay in the second half of the TCP round trip from node 0 to 2.

Analysis

This was a basic example of a DDoS attack happening inside the Smart Home network itself and the simulation allows us to realize how much the system can be compromised if an attacker is able to exploit a vulnerable device and gain control of its functionalities, perhaps as described in section 4.4.3.

Furthermore, what usually happens is that vulnerable IoT devices are compromised and employed as part of botnet for a large scale DDoS attack such as the ones carried out by the Mirai tool (seen in section 4.4.1). So, if there is not any monitoring and limitation rules of egress traffic in place a Smart Home owner could be providing dozens of device for botnets used by hackers from all over the world.

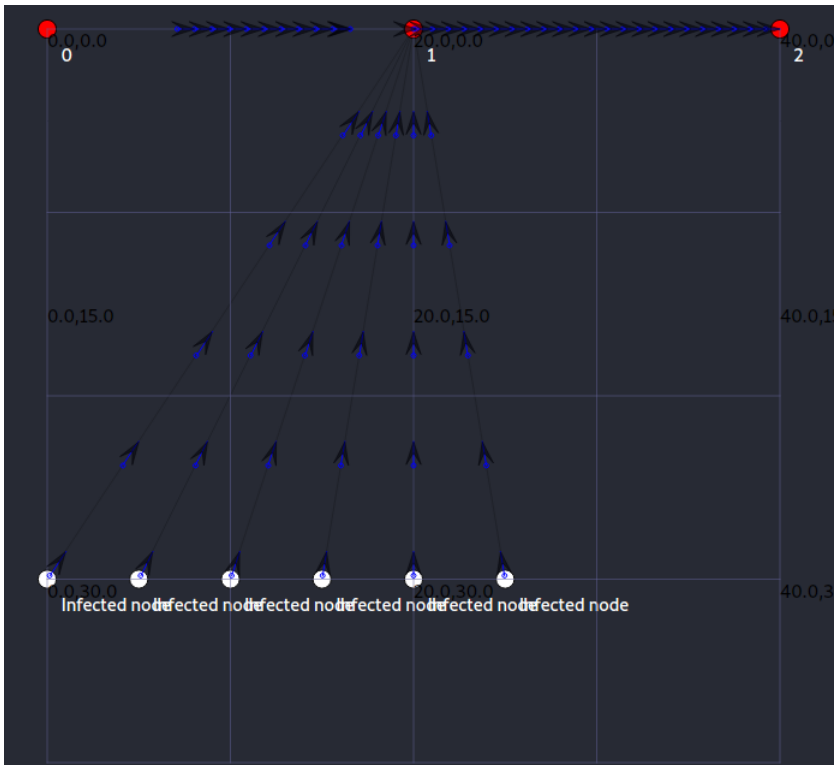

Figure 6.4. Screenshot of the attack simulation.

6.2.3 Blackhole attack

A Blackhole attack, also known as packet drop attack, is a specific type of DoS attack in which a node or router that is supposed to forward packets decides instead to discard them. This could be a consequence of a DDoS attack in which a legitimate router is under stress and drops the packets that is not able to able to handle, or a compromised malicious node/router that actively decides to to drop some specific packets at a casual or predefined rate.

This kind of attack is peculiar of wireless ad hoc networks which have an unconventional architecture and use different routing protocols such as AODV (Ad-hoc On-deman Distance Vector) since a host can falsely advertise itself as the shortest path to a destination and then drop the packets instead of relaying them.

The network is built with six nodes placed to form an elongated hexagon, so that we basically have the source and destination (respectively, node 1 and node 3) on the two lateral points of the shape, as can be seen in figure 6.5. Every node is only in range of his two adjacent perimeter nodes.

Of these six nodes, one of the two adjacent from the source is the malicious one so the source should theoretically have two equally valid routes to reach the destination, but because the AODV routing protocol is compromised the packet will be forced onto the attacker controlled node, which will break the forwarding chain.

Simulation

To simulate a Blackhole attack an example network implementing the AODV protocol was adopted, which is a common routing protocol in Smart Home networks, and more generally Wireless mesh networks.

The malicious behaviour from the compromised node was obtained by modifying the AODV model provided by ns-3 itself; this modified version of the module has a boolean status variable

which is set when installing the routing protocols on the nodes and indicates whether the node should or should not have a malicious behaviour.

Implementation

If the node is malicious it will advertise itself as shortest path whenever he receives a RREQ (Route REQuest) message, and this can be implemented by modifying the `RoutingProtocol::RecvRequest(..)` function so that a false routing table entry with higher sequence number (more attractive for AODV). This false route entry is then sent to the `RoutingProtocol::SendReplyByIntermediateNode(..)` where we also set to 1 the parameter *hopCount*, which indicates the hops needed to reach the destination by following this route.

```
void RoutingProtocol::RecvRequest(..){
        ..
        Ptr<NetDevice> dev =
            m_ipv4->GetNetDevice(m_ipv4->GetInterfaceForAddress (receiver));
        RoutingTableEntry falseToDst(dev,dst,true,
               rreqHeader.GetDstSeqno()+100,
               m_ipv4->GetAddress(m_ipv4->GetInterfaceForAddress(receiver),0),
               1,dst,m_activeRouteTimeout);
        SendReplyByIntermediateNode(falseToDst,toOrigin,
               rreqHeader.GetGratuitousRrep());
        ..
}
..
..
bool RoutingProtocol::SendReplyByIntermediateNode(..){
        ..
        if(EnableBHatk)
               rrepHeader.SetHopCount(1);
        ..
}
```

Once the malicious route is established and a packet is received the following lines of code placed in the `RoutingProtocol::Forwarding(..)` function ensure that the packet does not follow the regular relaying process by skipping the rest of the function entirely:

```
if(EnableBHatk){
        std::cout <<"Warning: a Blackhole attack is happening! Packet dropped
            . . . \n";
        return false;
}
```

In figure 6.6 two simulation runs are highlighted. In the first one the simulation script is launched without enabling the attack so we can see how the example packets are regularly transmitted and received using the AODV routing forwarding on this very basic straight line network; the flow monitor allows us to see how all the transmitted data is received by the destination.

Then the simulation runs again and this time the `enableAtk` parameter is set as *true* so we know that the node0, which is located between the source and the destination is going to drop any AODV transmission packet; this is evident once again by the flow monitor, which points out a 0Mbps throughput since no packet got to the destination.

Analysis

This is a basic implementation of a Blackhole attack but it's already effective in highlighting how much impact an attack like this can have on a system. In this case the attacker is not trying to be stealthy and but goes directly for the interruption of communications but the behaviour could

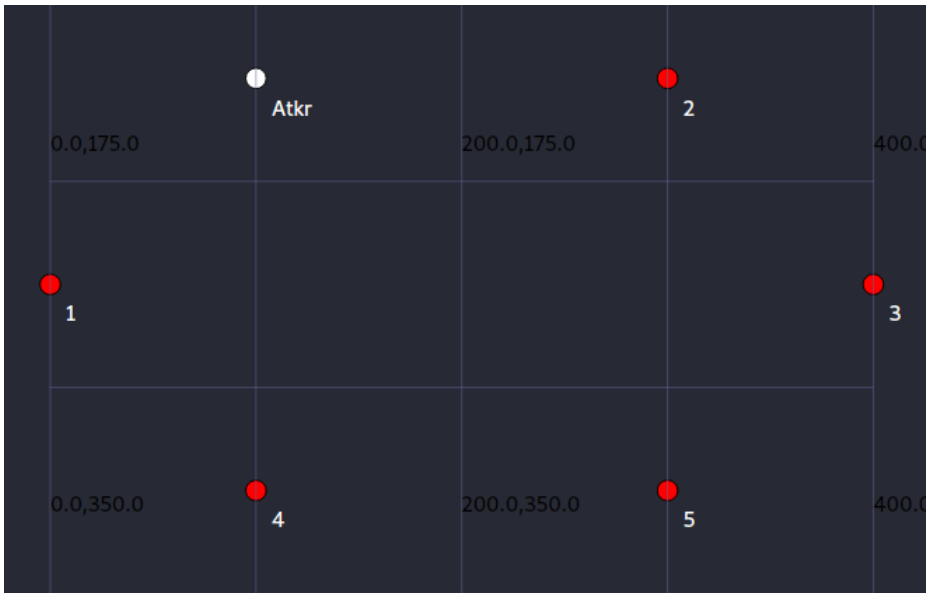

Figure 6.5. Screenshot of network visualization.

```
┌──(kali㉿kali)-[~/ns-allinone-3.37/ns-3.37]
└─$ ./ns3 run "blackhole_attack --enableAtk=false"
[0/2] Re-checking globbed directories...
ninja: no work to do.
Blackhole attack is not enabled.
Flow 1 (10.1.2.2 → 10.1.2.4)
  Tx Bytes:   10520
  Rx Bytes:   10520
  Throughput: 0.00891442 Mbps

┌──(kali㉿kali)-[~/ns-allinone-3.37/ns-3.37]
└─$ ./ns3 run "blackhole_attack --enableAtk=true"
[0/2] Re-checking globbed directories...
ninja: no work to do.
Blackhole attack enabled.
A Blackhole Attack is happening! Packet dropped . . .
A Blackhole Attack is happening! Packet dropped . . .
A Blackhole Attack is happening! Packet dropped . . .
A Blackhole Attack is happening! Packet dropped . . .
A Blackhole Attack is happening! Packet dropped . . .
A Blackhole Attack is happening! Packet dropped . . .
A Blackhole Attack is happening! Packet dropped . . .
A Blackhole Attack is happening! Packet dropped . . .
A Blackhole Attack is happening! Packet dropped . . .
A Blackhole Attack is happening! Packet dropped . . .
Flow 1 (10.1.2.2 → 10.1.2.4)
  Tx Bytes:   10520
  Rx Bytes:   0
  Throughput: -0 Mbps
```

Figure 6.6. Screenshot of simulation running without and with the attack being enabled.

be tweaked so that only a certain percentage of packet is dropped which means that the system gets slowed down without the attack being immediately obvious.

Additionally, further specifications could be added based on the context, such as only dropping packets coming from specific sources.

6.2.4 Wormhole attack

A Wormhole attack is another particularly grave attack in (wireless) ad hoc networks, hence we are going to study it using the previously mentioned AODV routing protocol.

This kind of attack has been known for a while but it can still be cause of troubles since if there is no specific countermeasure in place the attack can disrupt the network communication and the integrity of the transmissions while staying undetected. In 2006 *Yih-Chun Hu et al.* [40] defined the wormhole attack as:

> *"In this attack, an attacker records a packet, or individual bits from a packet, at one location in the network, tunnels the packet (possibly selectively) to another location, and replays it there."*

and further described:

> *"For tunneled distances longer than the normal wireless transmission range of a single hop, it is simple for the attacker to make the tunneled packet arrive with better metric than a normal multi-hop route, for example, through use of a single long-range directional wireless link or through a direct wired link to a colluding attacker."*

They also mentioned the following:

> *"Due to the nature of wireless transmission, the attacker can create a wormhole even for packets not addressed to itself, since it can overhear them in wireless transmission and tunnel them to the colluding attacker at the opposite end of the wormhole"*

which, combined with the what was already implemented for the Blackhole attack in section 6.2.3 to make a malicious node advertise itself as the best router, can be used by an attacker to force packets of the network to pass through the tunnel, consequently compromising any communication.

Simulation

We first designed a topology, which would best allow us to test the functioning and feasibility of this Wormhole attack, which can be seen in figure 6.7, where the malicious nodes n0 and n5 are on the opposite sides of the network but linked by a wired Point-to-Point connection.

Then, to create this attack we had to add further malicious behaviours to the one already integrated for the Blackhole attack so that, after "sponsoring" itself as best node and obtaining the packet from the source, the hostile node can tunnels the packet to its companion instead of a relaying it normally.

Implementation

Adding up to the changes already made to the AODV routing protocol, we defined new variables to take into account the fact that the malicious nodes have two interfaces, a legit one and for the Point-to-Point tunnel.

We then modified once again the `RoutingProtocol::RecvRequest(..)` function so that if the attack is enabled when the source of the request is one of the two ends of the wormhole tunnel the reverse routing table entry is created with the other end as receiver.

```
if(EnableWrmAttack && (src==FirstEndOfWormTunnel)){
    std::cout << "Wormhole attack occuring, redirecting to second end of
        tunnel... \n";
    dev = m_ipv4->GetNetDevice(
              m_ipv4->GetInterfaceForAddress(SecondEndOfWormTunnel));
```

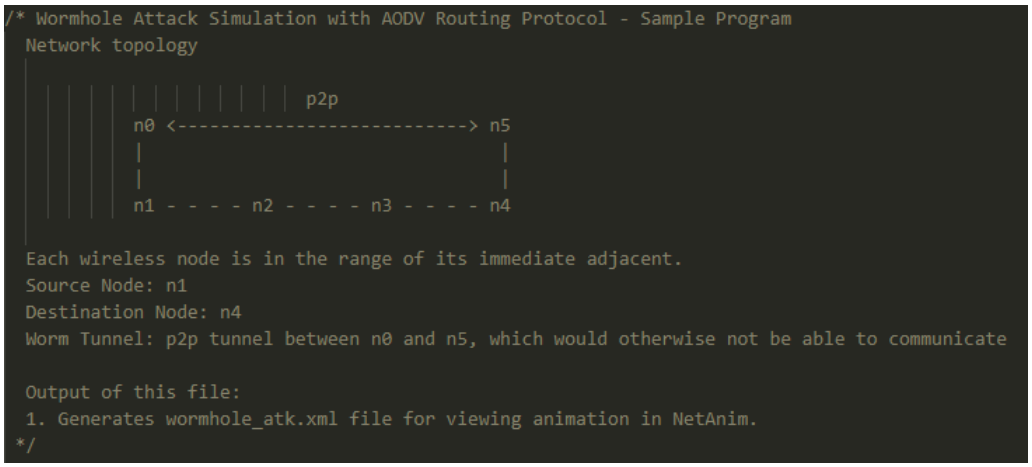

Figure 6.7. Network topology for the Wormhole attack

```
    receiver=SecondEndOfWormTunnel;
}else if(EnableWrmAttack && (src==SecondEndOfWormTunnel)){
    std::cout << "Wormhole attack occuring, redirecting to first end of
        tunnel... \n";
    dev = m_ipv4->GetNetDevice(
                m_ipv4->GetInterfaceForAddress(FirstEndOfWormTunnel));
    receiver=FirstEndOfWormTunnel;
}
```

And a similar change was also applied to the `RoutingProtocol::ProcessHello(..)` function so that when a node receives an "Hello message" from a neighbor, and said node is one of the end of the tunnel the route is set up to the other end.

We also modified the `RoutingProtocol::RecvAodv(..)` function so that when received at the end of the tunnel a packet is 'untunneled' which means pretending it didn't arrive from the tunnel by changing the address of the receiver from the tunnel one to the real network one.

```
if(EnableWrmAttack){
    if(sender==FirstEndOfWormTunnel && receiver==SecondEndOfWormTunnel){
        std::cout << "Received by Second Wifi Wrm Tunnel \n";
        receiver=SecondEndWifiWormTunnel;
    }
    if(sender==SecondEndOfWormTunnel && receiver==FirstEndOfWormTunnel){
        std::cout << "Received by First Wifi Wrm Tunnel \n";
        receiver=FirstEndWifiWormTunnel;
}
```

Once again the simulation is controlled by the `enableAtk` parameter and if we launch it with said parameter set to *true* the AODV protocol installed on the two malicious node will have it's `enableWrmAtk` flag set to `true` so that consequently the nodes will have the behaviour just described. We can see the logs of the packets being tunneled from n0 to n5 (and vice-versa for the answers).

Analysis

Although this kind of attack is context dependent, which means that its feasibility depends on some factors such as the network topology or the possibility of creating the single long-range wireless or wired link, it is definitely something that we should be aware of.

If such attack was to take place without the system, and consequently us, noticing a big part (if not all) of the transmissions on the network could be compromised and the attacker could very easily sniff packets or alter them while in transit.

6.3 GNS3 simulation

This section is going to report the work done with the GNS3 simulation tool. Conversely to the two previously seen tools, the number one reason for using GNS3 (and as mentioned EVE-NG) was trying to recreate an example network of a Smart Home environment as similar and as reliable as possible to be used as a benchmark for future studies.

By default, GNS3 provides only access to a generic VPCs (virtual PCs) appliance and to a few network appliances such as Ethernet hub and switch, ATM switch, and Frame Relay switch. It also offer a "Cloud" node thanks to which it's possible to link the simulated network, and consequently every device added to it, the internet connection of the host computer.

So, one of the first steps taken was research and testing the possible appliances related to the Smart Home network that could be implemented from the GNS3 marketplace.

6.3.1 Network Simulation

Unfortunately, many of the most common appliances are proprietary devices such as all the Cisco's family routers, so the images for the installation cannot be obtained unless you buy or have some kind of enterprise license.

Therefore, we had to opt for a router with a free software download, and we choose the appliance "MikroTik CHR" which is a GNS3 appliance set up with Cloud Hosted Router (CHR) produced by Mikrotik, a RouterOS version meant to run as a virtual machine. [41] RouterOS is a known operating systems based on a Linux kernel but with some simplifications and standardization in the network settings setup processes.

As for the nodes, one of the embedded Linux distribution frequently used on IoT devices is Tiny Core Linux, a free and open-source Linux kernel based operating system created by the Core Project group, which provides: "TinyCore ..., an 16MB FLTK/FLWM desktop". [42] Considering the often limited resources of IoT devices, such as sensors, gateways, and other edge devices, TinyCore minimalism is a perfect fit because it can run on such hardware constrained devices with ease.

The Tiny Core Linux is available on the GNS3 Appliances marketplace but while importing the device template we experienced various problems with the installation, as first the image had to be unpacked and its configuration modified but even after that once installed the device would not start up correctly.

Hence, we pivoted to the even smaller version "*Core*", also known as *"Micro Core Linux"* which is basically the same Linux variant as TinyCore but without a graphical desktop. This one was installed correctly, and it provide just enough functions to act as a basic IoT node in a network simulation.

Additionally, a Kali Linux appliance was installed to have an host to better test the system settings or to potentially act as an attacker.

Every new appliance template in GNS3 is installed by importing the downloaded *.gns3a* appliance file and the corresponding image file, for example for the just mentioned Kali node we had *"kali-linux.gns3a"* and *"kali-linux-2021.1-live-amd64.iso"* plus an additional *".qcow2"* virtual disk file.

Network Setup

After importing the appliance templates we started setting up a basic example network with the devices mentioned and we can see a system summary in figure 6.8 and the topology built in figure 6.9.

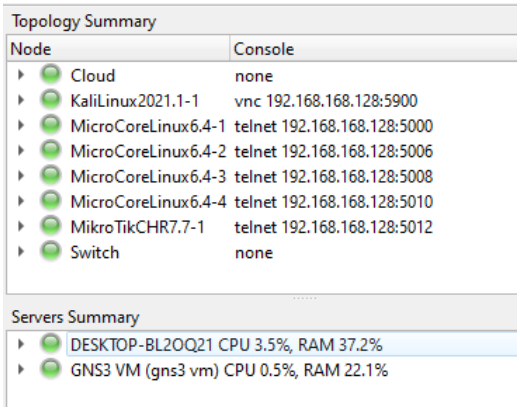

Figure 6.8. Screenshot of GNS3 system summary.

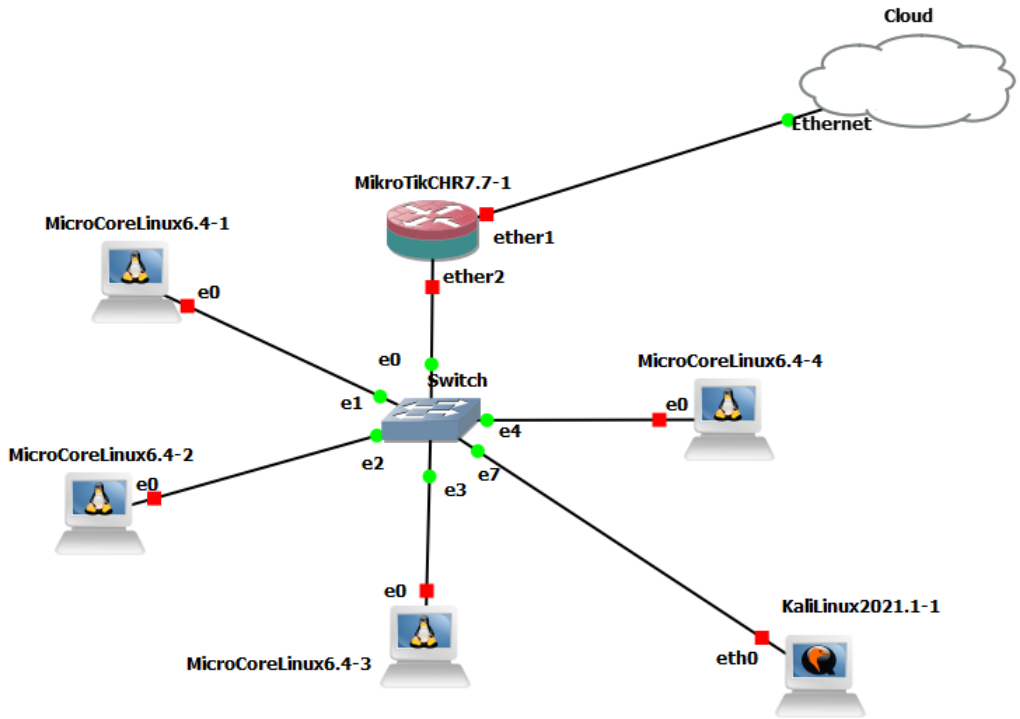

Figure 6.9. Screenshot of GNS3 topology.

The “Cloud” was installed on the host PC and not the on the VM so that the internet connection can be set up correctly.

The MikroTik router also had to be set up to properly configure IP access, set up a DHCP server, configure the Internet Connection, and set up the NAT configuration. This was done through the following series of commands:

```
$ /interface bridge add name=local
$ /interface bridge port add interface=ether2 bridge=local
$ /ip address add address=192.168.88.1/24 interface=local
```

which are used to add bridge interface and ports and add an IP address to the LAN.

Then we have to set up the DHCP through the `setup` command:

```
$ /ip dhcp-server setup [enter]
       dhcp server interface: local [enter]
       dhcp address space: 192.168.88.0/24 [enter]
       gateway for dhcp network: 192.168.88.1 [enter]
       addresses to give out: 192.168.88.2-192.168.88.254 [enter]
       dns servers: 192.168.88.1 [enter]
       lease time: 10m [enter]

$ /ip dhcp-client add disabled=no interface=ether1
```

and we also configure internet connection by adding a DHCP client on the public interface so that we have Dynamic Public IPs automatically set up.

Finally, we add a NAT rule so that network hosts can access the internet by making their local addresses routable over the internet.

```
$ /ip firewall nat add chain=srcnat out-interface=ether1 action=masquerade
```

Connectivity Testing

After having set up the router any MicroCore host connected to it via Ethernet, either directly or going through the switch, was automatically assigned an IP address and joined the network. This can be easily checked by turning on one of the host, accessing its console and running the simple `ifconfig` command which will show us the Ethernet interface on which we placed the connection (eth0) being active; additionally we can ping "Google.com" at `8.8.8.8` to check for internet connectivity and as seen in figure 6.10 we were able to reach the server, which also mean that the router is working correctly.

Furthermore, as given away by figure 6.9, we connected the Kali Linux host to the router and thus to the network. The introduction of an host such as this Kali one, with its attack tools, should be of great concern for any network. For example, in our network it allowed for a simple use case from the Kali of the renowned "nmap" command: host discovery.

Opening the command line from the Kali host we could launch this network exploration 'attack' with the single command

```
$ nmap -sn -PR 192.168.88.1/24
```

and a potentially malicious host it's able to easily identify all of the system devices.

Going a step further and using a more advanced scan, the OS detection one, we can verify the host discovery process and although nmap is not able to identify the peculiar MicroCore linux version that we installed it can still detect any port that could be open on a device based on the need of using a particular service.

The is done by adding the *-O* parameter so the command:

```
$ nmap -O -v 192.168.88.1/24
```

will yield results similar to what is shown in figure 6.11.

Analysis

This example simulation allowed us to create a solid baseline of network usable for additional research on Smart Home environments and nodes. Although the devices installed are not an exact copies of IoT devices they can be set up with the same operating system (e.g., Micro Core Linux) and further tweaked by turning on specific services to resemble the operations and behaviours of real world devices.

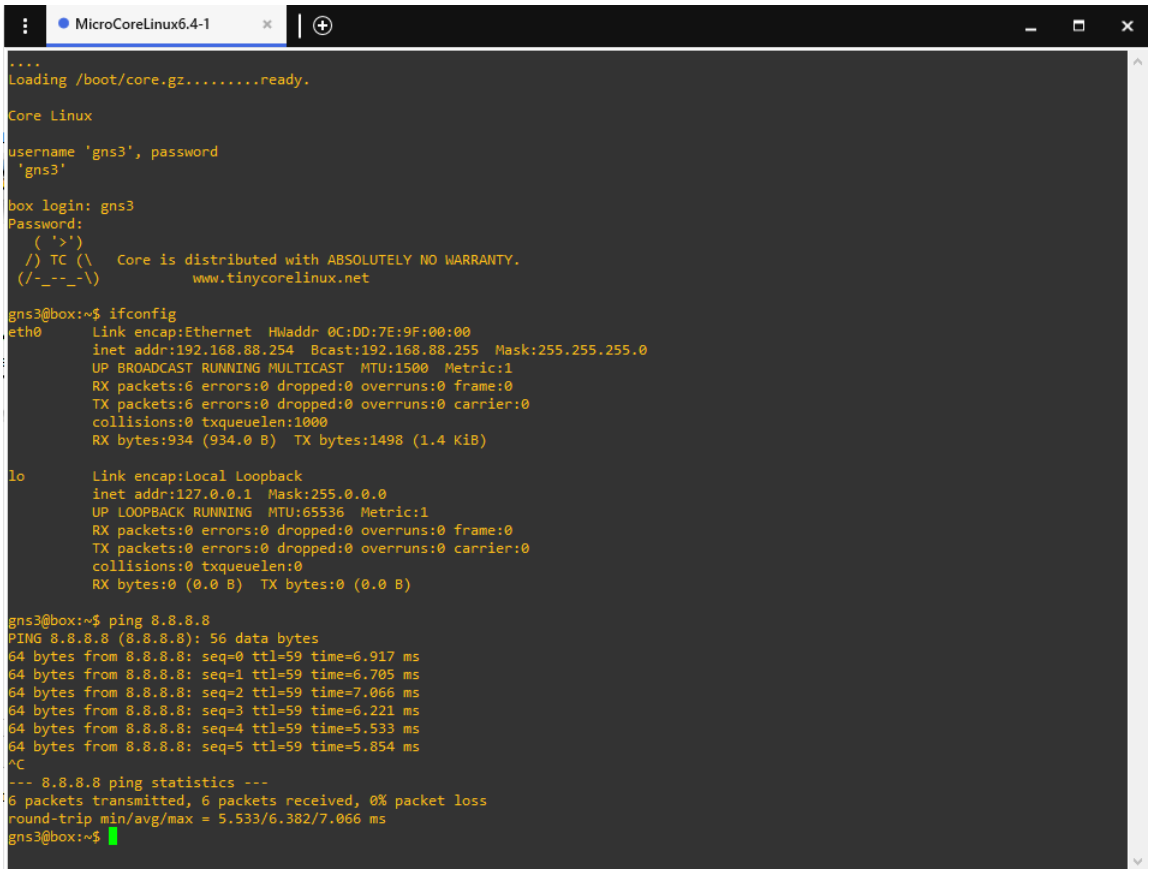

Figure 6.10. Screenshot of the MicroCore host network interfaces and connectivity test via ping.

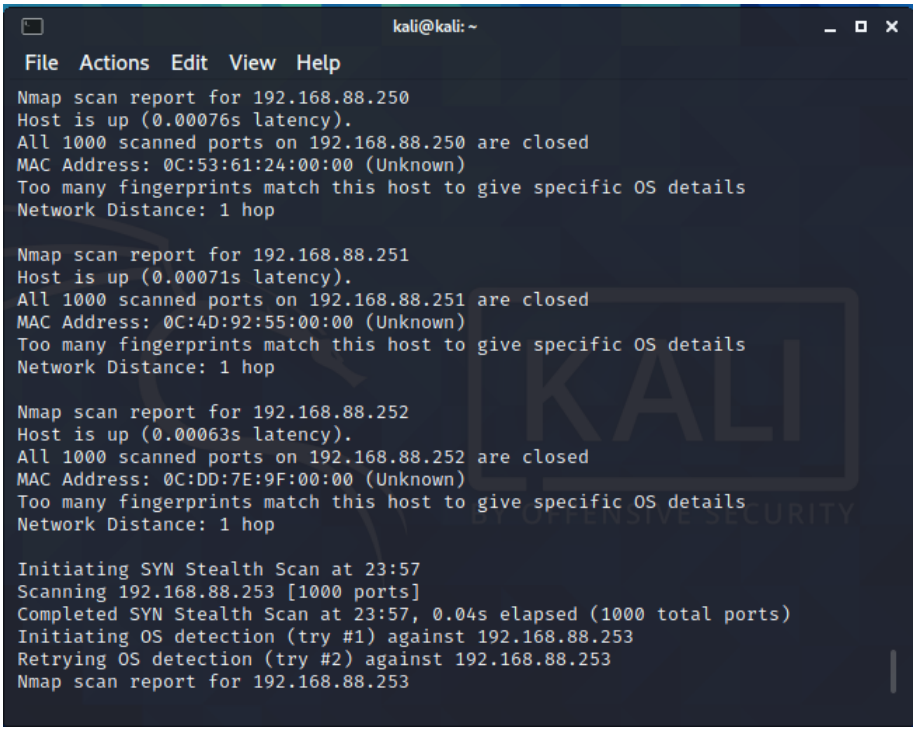

Figure 6.11. Screenshot of nmap host discovery.

6.4 Chapter recap

In this chapter we presented the simulation scenarios built during the production of this thesis. The simulations are categorised by the tool on which they were integrated, and if more than one was built on the same tool then they divided by the category of the simulation, as it happened for *ns-3*. For each scenario presented we had a generic section explaining the context of the simulation, one describing how it was actually implemented, and one drawing some conclusion on what was achieved.

Chapter 7

Conclusions

The work done in this thesis can be split in two main parts: the first part was focused on thoroughly researching the topic of Cyber-Physical Systems and their specific application in Smart Home's environments. This included analysing both the history of these systems and their historical uses as well as the state of the art more recent protocols, devices, and practices. The just mentioned research was fundamental in understanding the topic of this research work, since before trying to simulate a system you need to deeply understand how said system works.

Then, we moved on to the second part, the evaluation of simulation tools with the purpose of reproducing the Cyber-Physical Systems just studied. Being able to recreate such systems in a virtual environment would allow researchers for more flexible analysis without the need of all the physical resources usually needed to run a Smart Home system. Additionally, the topology of the system being studied could be quickly and efficiently modifies without having to physically alter the set up.

The main concern with this second task is being able to create a realistic and reliable simulation which should have a behaviour similar to the one of the real-world systems being simulated. The simulation tools used for this research were of two different natures.

As seen in section 6 discrete-event network simulators such as ns-3 and OMNeT++ are valid tools to build networks using a trustworthy reproduction of real-world protocols (e.g., 802.15.4, AODV routing, etc) but lack the possibility of integrating exact copies of real-world devices into our network.

On the other hand, more complex tools such as GNS-3 and EVE-NG provide more flexibility in the choice of devices while building a simulation, although this process of build and set up can require more time compared to the other tools mentioned. It should also be mentioned that many of the most common devices (e.g., the Cisco family of routers) used in the real-world are proprietary devices and as such the firmware images to be used for simulating purposes are not free, but a license is required.

Nevertheless, the simulation of Cyber-Physical Systems, particularly the ones integrated in smart homes, was confirmed to be a great way of studying such environments, as it enables the possibility of modelling the network to your preferences and defining the most appropriate behaviour for network's nodes. The simulations in OMNeT++ and ns-3 gave us a good opportunity of testing protocols by verifying the feasibility of known attacks, and once a simulation is tested and stable this could be further expanded into looking for new attacks. Meanwhile, the use of tools such as EVE-NG and GNS3 allowed us to generate topology containing nodes which are exact copies of the real-world ones, for example the Linux embedded distro called TinyCore Linux, often used on IoT devices.

My main impression, while working on this thesis, is the feeling of the lack of a "universal" tool where reliable copies of real-world devices from different vendors can be integrated into a simulated network. Instead, we have multiple different tools which can be more or less convenient for different vendors related devices and services. For example, we have a proprietary Cisco simulator, one for AWS IoT services, etc.

I believe that this can be partly blamed on the fact that often people who research Smart Home systems and their IoT devices are either the R&D developers from the vendor itself who clearly have access to the source firmware and any additional internal simulation tools, or people who buy the actual physical devices to reverse engineer them looking for vulnerabilities, which can test said devices in a "hardware in the loop" environment.

Furthermore, the technologies integrated and used in Cyber-Physical Systems, whether it's devices, communication protocols, or services, are many and varied, which adds up to the difficulty of having a single tool where all of them can be integrated.

Some future research work lies together with the "newer" protocols being developed, such as Matter and Thread, for which multiple IoT devices vendors and service providers are joining forces to build a unified standard promising seamless compatibility and security. Particularly the Matter standard, which is still being actively developed (version 2.0 is set for release in March-April 2023), once more evolved and widespread could be a unifying standard that leads to the creation of more complete simulation tools, perhaps through services such as Google's developer center, where hints of virtualization are already available.

Appendix A

User Manual

The following section of this first appending has the purpose of explaining how to reproduce any of the simulation proposed in this research, starting from the installation of different tools used.

Any of the source code developed during this thesis can be found online in a GitHub repository of my personal account.

A.1 OMNeT++

The OMNeT++ tool is available for Linux, Windows, and macOS hosts in the following versions:

- Linux aarch 64 bit
- Linux x86 64 bit
- macos x86 64 bit
- Windows x86 64 bit

For the purpose of this research, the latest available 64bit Windows release, *OMNeT++ 6.0.1*, was installed on a Windows 10 host.

A.1.1 Tool installation

The installer for Omnetpp can be downloaded from the official OMNeT++ website. This will download the Windows-specific archive, named *omnetpp-6.0-windows-x86_64.zip* which is a self-contained package that includes all the necessary elements to run the software.

Once the files are extracted in the desired program location the *mingwenv.cmd* file has to opened which will launch a console with a MSYS bash shell; here all we have to do is run the following commands:

```
$ ./configure
$ make
```

and the build progress will start creating both debug and release binaries.

From now on to launch the program we just need to first open again the shell and simply write `omnetpp` and the OMNeT++ IDE will start.

Upon the first launch of the program the user will be prompted for the installation of any additional packages, such as the INET framework used for this research, so they can be installed directly from the IDE from the *Help - Install Simulation Models...* window by picking the desired model and following the project import procedures. Any additional model can also be installed by downloading it from the Simulation Models and Tools official webpage and then imported by following the *File - Import.. - Import Existing Project into Workspace* menu.

A.1.2 Simulation execution

After having installed OMNeT++ and the additional INET framework we are only a few steps away from running the simulation presented.

Firstly, the file describing the compromised devices needs to be added along the other modules in the source folder of the INET project, specifically the path used to include the file in the simulation was *inet4.4/src/inet/node/compromised/*. Having done this, it's now only a matter of adding the two files describing the simulation which are the *.ned* and the *.ini* file into the omnetpp workspace in use and the simulation can be easily launched by pressing the play button of the IDE.

A.2 ns-3

ns-3 is supported for Linux, macOS, and Windows systems although Windows is the least used and supported system while Linux is the preferred one. The release used in this thesis is the *ns-3.37* and it was installed on a Linux Virtual Machine through the VirtualBox tool.

The complete ns-3 installation guide can be found in HTML version on the official website.

A.2.1 Tool installation

Before building the ns-3 tool we need to install the minimum required packages needed and this can be done through the Linux *apt* command:

```
$ sudo apt-get update
$ sudo apt-get install g++ python3 cmake ninja-build git
```

Additional packages can be added for various purposes, for example the ones needed for the NetAnim animator: `qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools`.

The tool can be downloaded as a release tarball, either through the command-line *wget* tool or via browser, or cloned from the Git repository. The former option was used and these are the commands needed:

```
$ wget https://www.nsnam.org/releases/ns-allinone-3.37.tar.bz2
$ tar xfj ns-allinone-3.37.tar.bz2
$ cd ns-allinone-3.37/ns-3.37
```

Once correctly downloaded ns-3 can be built with the following commands:

```
$ ./ns3 configure --enable-examples --enable-tests
$ ./ns3 build
```

The build process an takes a few minutes to complete but at the end the output should be a *'build' finished successfully* and a list of modules built. Unit tests can be run to check the build with the `./test.py` command.

A.2.2 Simulation execution

Running a simulation in ns-3 is pretty much straightforward as the only thing that we need to do is place the C++ script used to describe the simulation in the *./ns-3.37/scratch* directory and then simply launch the said simulation from the "ns-3.37" directory.

For example to launch the DoS attack simulation we will place the *Wifi_DoS.cc* file into the *scratch* folder and then we can use the command `./ns3 Wifi_DoS`.

A.3 GNS3

GNS3 is available on Windows, macOS, and Linux with the minimum requirements being respectively Windows 7 64bit, Mavericks 10.9, and any Linux Distro but Debian and Ubuntu are the ones provided and supported.

For this research the tool was installed on a Windows 10 machine and the GNS3 VM was deployed on the VMware Workstation tool.

The complete GNS3 Windows installation document can be found on the official GNS3 documentation website.

A.3.1 Tool installation

The installation it's guided by the all-in-one installer tool downloaded so the user only needs to follow the procedure and the GNS3 software will correctly be installed.

Additionally, the VM must be installed on the chosen virtualization tool, so in this case the "GNS3VM.ova" file was imported into the VMware workstation by following the procedure for "Open an existing machine".

Even a simple simulation as the one showcased in section 6.3 can be quite resource demanding on the Virtual Machine, we tweaked the settings so that it had 4 cores and 8GB of RAM.

A.3.2 Appliances installation

The appliances can be found in the GNS3 marketplace where there usually is a direct download for the ".gns3a" file and a list of supported corresponding images of the device in question and an external link to the download of said virtual images of the device in question.

Once we have downloaded everything we can create a new object template by importing the appliance, which can be done from the GNS3 GUI following the *"File - Import appliance"* path and selecting the correct ".gns3a" file. Then the GNS3 VM has to be selected as the server on which the appliance will be installed, the preferred Qemu binary can be chosen and finally the appliance template will be created.

Appendix B

Developer Manual

This second appendix has instead the role of highlighting any changes made to the source code of the tools used, such as the variations made to the model used to implement the AODV protocol in the ns-3 tool, as well as providing some further insight on the code developed for the simulations.

B.1 OMNeT++

When OMNeT++ is installed inside both a "src" and a "samples" folder are created. Contrary to what one could expect in the *src* folder there's only the files related to the program and GUI itself, while all the models describing the network functionalities are in the *samples* folder.

Here, if installed correctly, we can find the "inet4.4" which is the network framework used for this research and that contains its own *src* sub-folder where we can then find the C++ files used to describe the various modules implemented, classified by category and type. For example the description of the battery model used in the simulation can be found following the path `omnetpp-6.0.1/samples/inet4.4/src/inet/power/storage`.

B.1.1 Tool modifications

While using the OMNeT++ tool we did not have to modify the existing source code but rather to add new code, specifically adding new modules describing the actors needed for our simulation.

In the battery deployment simulation this meant that we were able to differentiate the malicious device from the legit ones while building similar device, as both the modules extended the "WirelessHost" omnetpp model.

The following is a code example of the victim sensor module, where we set up its parameters so that it had a limited battery capacity and a energy consumer model that drains the battery based on the activity of the device.

```
module Adhoc_VictimSensor extends WirelessHost
{
    parameters:
        @display("i=misc/sensor2");
        wlan[*].mgmt.typename = default("Ieee80211MgmtAdhoc");
        wlan[*].agent.typename = default("");
        forwarding = default(true);
        energyStorage.typename = default("SimpleEpEnergyStorage");
        energyManagement.typename = default("SimpleEpEnergyManagement");
        wlan[*].radio.energyConsumer.typename =
            default("StateBasedEpEnergyConsumer");
        mobility.typename = "StationaryMobility";
}
```

B.1.2 Code developed

Battery Depletion attack

The simulation's environment is built with a *.ned* file where we defined the network by introducing the main modules, such as the `Ipv4NetworkConfigurator` and the `Ieee80211ScalarRadioMedium` as seen in the next code snippet:

```
network Ieee802154_BatteryDep
{
    @display("bgb=20,20;bgi=showcases/floorplan,s");
    types:
    submodules:
        sensor1: SensorNode_Victim {
            @display("p=9.467456,3.7141557;i=misc/sensor");
        }
        ..
        ..
        sensor9: SensorNode_Victim {
            @display("p=2.5,17.5;i=misc/sensor");
        }
        infsensor: SensorNode_infected {
            @display("p=2.4761038,8.484297;i=misc/sensor,red;i2=status/excl");
        }
        lamp1: SensorNode {
            @display("p=8,5;i=status/bulb");
        }
       ..
       ..
        lamp8: SensorNode {
            @display("p=4,5;i=status/bulb");
        }
        controller: SensorNode {
            @display("p=6.5,10.5;i=misc/sensorgateway");
        }
        configurator: Ipv4NetworkConfigurator {
            @display("p=17.405554,0.7282658");
        }
        radioMedium: Ieee802154NarrowbandScalarRadioMedium {
            @display("p=16.895767,2.9130633");
        }
        visualizer: IntegratedMultiVisualizer {
            @display("p=18.497952,4.770141");
        }
}
```

The simulation's behaviour is then defined in the *omnetpp.ini* file, were we first set up the battery visualizer with the command

```
*.visualizer.energyStorageVisualizer.displayEnergyStorages = true
```

Then, the energy storage, energy management, and energy consumption was set up:

```
*.sensor*.hasStatus = true
*.sensor*.energyStorage.nominalCapacity = 5J
*.sensor*.energyStorage.initialCapacity = uniform(0.1 * nominalCapacity,
    nominalCapacity)
*.sensor*.energyManagement.nodeShutdownCapacity = 0.1 *
    parent.energyStorage.nominalCapacity
```

```
*.sensor*.energyManagement.nodeStartCapacity = 0.5 *
    parent.energyStorage.nominalCapacity

*.sensor*.wlan[*].radio.energyConsumer.sleepPowerConsumption = 0.1mW
*.sensor*.wlan[*].radio.energyConsumer.receiverIdlePowerConsumption = 2mW
*.sensor*.wlan[*].radio.energyConsumer.receiverBusyPowerConsumption = 5mW
*.sensor*.wlan[*].radio.energyConsumer.receiverReceivingPowerConsumption =
    10mW
*.sensor*.wlan[*].radio.energyConsumer.transmitterIdlePowerConsumption = 2mW
*.sensor*.wlan[*].radio.energyConsumer.transmitterTransmittingPowerConsumption
    = 100mW
```

Finally, the nodes behaviour was described, with the sensors periodically sending information to the controller which then sends controls to the lamps:

```
*.sensor*.numApps = 1
*.sensor*.app[*].typename = "UdpBasicApp"
*.sensor*.app[*].destAddresses = "controller"
*.sensor*.app[*].destPort = 1000
*.sensor*.app[*].sendInterval = exponential(1s)
*.sensor*.app[*].startTime = uniform(0s,1s)
*.sensor*.app[*].messageLength = 10Byte
*.sensor*.app[*].packetName = "SensorData"

*.controller.numApps = 2
*.controller.app[0].typename = "UdpBasicApp"
*.controller.app[0].destAddresses = "lamp1 lamp2 lamp3 lamp4 lamp5 lamp6
    lamp7 lamp8"
*.controller.app[0].destPort = 1000
*.controller.app[0].sendInterval = 0.125s
*.controller.app[0].startTime = exponential(1s)
*.controller.app[0].messageLength = 10Byte
*.controller.app[0].packetName = "ControlData"

*.controller.app[1].typename = "UdpSink"
*.controller.app[1].localPort = 1000

*.lamp*.numApps = 1
*.lamp*.app[0].typename = "UdpSink"
*.lamp*.app[0].localPort = 1000
```

The infected device will instead start spamming the network to deploy the battery of the legit sensors:

```
*.infsensor.numApps = 1
*.infsensor.app[*].typename = "UdpBasicApp"
*.infsensor.app[*].destAddresses = "controller"
*.infsensor.app[*].destPort = 1000
*.infsensor.app[*].packetName = "MaliciousSensorData"
*.infsensor.app[*].sendInterval = 0.0005ms
*.infsensor.app[*].messageLength = 1000Byte
*.infsensor.app[*].startTime = uniform(0s,1s)
```

Additional code has to be added to visualize the routing tables, the message in transit and the battery levels in the graphical view of the simulation:

```
# routing table visualization
*.visualizer.*.routingTableVisualizer[0].displayRoutingTables = false
*.visualizer.*.routingTableVisualizer[0].displayRoutesIndividually = false
```

```
*.visualizer.*.routingTableVisualizer[0].lineShift = 0
*.visualizer.*.routingTableVisualizer[0].displayLabels = false

# interface table visualization
*.visualizer.*.interfaceTableVisualizer[0].displayInterfaceTables = false

# data link visualization
*.visualizer.*.numDataLinkVisualizers = 2
*.visualizer.*.dataLinkVisualizer[*].displayLinks = true
*.visualizer.*.dataLinkVisualizer[0].nodeFilter = "sensor* or controller or
    inf*"
*.visualizer.*.dataLinkVisualizer[1].*Color = "blue"
*.visualizer.*.dataLinkVisualizer[1].nodeFilter = "lamp* or controller or
    inf*"

# energy visualization
*.visualizer.energyStorageVisualizer.displayEnergyStorages = true
```

B.2 ns-3

In contrast to what was done in OMNeT++, to simulate the attacks on the AODV routing protocol in ns-3 there was the necessity of modifying the AODV implementation provided by ns-3 itself as we had to add some code to describe the malicious behaviour of potential attackers.

ns-3 is a set of C++ libraries that can be used by C++ or Python programs to construct simulation scenarios and execute simulations, which means that in the program's installation folder we can find the "src" folder with an extensive set of subfolders containing the C++ used to implement the various networking modules.

B.2.1 Tool modifications

The files that needs to be changed are *"aodv-routing-protocol.h"* and *"aodv-routing-protocol.cc"* located in the `./ns-3.37/src/aodv` folder.

In the header file the following simple lines were added to include status variables that indicate whether a node is malicious or not and the additional variables to be used in the case of a WormHole attack to set the addresses of the wormhole tunnel ends:

```
..
bool EnableBHatk;
bool EnableWrmAttack;
..
void SetEnableBHatk(bool f){
       EnableBHatk=f;
}
bool GetEnableBHatk() const{
       return EnableBHatk;
}
void SetWrmAttackEnable(bool f){
    EnableWrmAttack=f;
}
bool GetWrmAttackEnable() const{
    return EnableWrmAttack;
}

Ipv4Address FirstEndOfWormTunnel;
```

```
Ipv4Address SecondEndOfWormTunnel;
Ipv4Address FirstEndWifiWormTunnel;
Ipv4Address SecondEndWifiWormTunnel;
..
```

On the other hand the "aodv-routing-protocol.cc" requires more variations to implement the behaviour discussed, first of all the attributes mentioned in the header file must be added in the protocol definition, so that the correct functions are referenced:

```
..
.AddAttribute ("EnableBHatk", "Indicates whether a BlackHole attack is
    enabled or not",
               BooleanValue (false),
               MakeBooleanAccessor (&RoutingProtocol::SetEnableBHatk,
                                    &RoutingProtocol::GetEnableBHatk),
               MakeBooleanChecker ())
.AddAttribute("EnableWrmAttack",
               "Indicates whether a Wormhole Attack is enabled or not.",
               BooleanValue(false),
               MakeBooleanAccessor (&RoutingProtocol::SetWrmAttackEnable,
                                    &RoutingProtocol::GetWrmAttackEnable),
                                    MakeBooleanChecker())
.AddAttribute("FirstEndOfWormTunnel",
               "Indicates the first end of the Wormhole tunnel.",
               Ipv4AddressValue("10.1.2.1"),
               MakeIpv4AddressAccessor
                   (&RoutingProtocol::FirstEndOfWormTunnel),
               MakeIpv4AddressChecker ())
.AddAttribute("SecondEndOfWormTunnel",
               "Indicates the second end of the Wormhole tunnel",
               Ipv4AddressValue("10.1.2.2"),
               MakeIpv4AddressAccessor(&RoutingProtocol::SecondEndOfWormTunnel),
               MakeIpv4AddressChecker())
.AddAttribute("FirstEndWifiWormTunnel",
               "Indicates the wifi interface of the first end of the Wormhole
                   tunnel",
               Ipv4AddressValue("10.0.1.37"),
               MakeIpv4AddressAccessor(&RoutingProtocol::FirstEndWifiWormTunnel),
               MakeIpv4AddressChecker())
.AddAttribute("SecondEndWifiWormTunnel",
           "Indicates the wifi interface of the second end of the Wormhole
               tunnel",
           Ipv4AddressValue("10.0.1.38"),
           MakeIpv4AddressAccessor(&RoutingProtocol::SecondEndWifiWormTunnel),
           MakeIpv4AddressChecker())
..
```

Then, to make the attacker node drop the packet the following check has to be added in the `bool RoutingProtocol::Forwarding(..)` so that the normal forwarding procedure is skipped if the blackhole attack is being executed:

```
..
if(EnableBHatk){
    std::cout <<"Warning: a Blackhole attack is happening! Packet dropped . .
        . \n";
    return false;
}
..
```

skiphere

As mentioned in 6.2.3 the attacker needs to aggressively advertise as the best node for routing and this is done by modifying the `void RoutingProtocol::RecvRequest(..)` function:

```
..
if (EnableBHatk || m_routingTable.LookupRoute(dst, toDst)){
    if (toDst.GetNextHop() == src){
        NS_LOG_DEBUG("Drop RREQ from " << src << ", dest next hop " <<
            toDst.GetNextHop());
        return;
    }
    //mod
    if (EnableBHatk || ((rreqHeader.GetUnknownSeqno() ||
        (int32_t(toDst.GetSeqNo()) - int32_t(rreqHeader.GetDstSeqno()) >= 0))
        && toDst.GetValidSeqNo())){
        if(EnableBHatk || (!rreqHeader.GetDestinationOnly() &&
            toDst.GetFlag() == VALID)){
            m_routingTable.LookupRoute(origin, toOrigin);
            //mod
            if(EnableBHatk){
                Ptr<NetDevice> dev =
                    m_ipv4->GetNetDevice(m_ipv4->GetInterfaceForAddress
                    (receiver));
                RoutingTableEntry falseToDst(dev, dst, true,
                    rreqHeader.GetDstSeqno()+100,
                    m_ipv4->GetAddress(m_ipv4->GetInterfaceForAddress(receiver),0),
                                    1,dst,m_activeRouteTimeout);
                SendReplyByIntermediateNode(falseToDst, toOrigin,
                    rreqHeader.GetGratuitousRrep());
                return;
            }
             SendReplyByIntermediateNode(toDst, toOrigin,
                 rreqHeader.GetGratuitousRrep());
             return;
        }
        rreqHeader.SetDstSeqno(toDst.GetSeqNo());
        rreqHeader.SetUnknownSeqno(false);
    }
}
..
```

Additionally, we still need to go at the beginning of the `void RoutingProtocol::SendReplyByIntermediateNode(..)` function just called by the previously modified code snippet and set the hopcount parameter as low as possible:

```
..
if(EnableBHatk)
        rrepHeader.SetHopCount(1);
..
```

This makes sure that the false routing table entry is created advertising the malicious node as the best possible route.

For the WormHole attack we also added more changes to the `RoutingProtocol::RecvRequest(..)` function so that if the attack is enabled and the source of the request is one of the two ends of the wormhole tunnel then the reverse routing table entry is created with the other end as the receiver:

```
if (!m_routingTable.LookupRoute(origin, toOrigin))
```

```
{
        Ptr<NetDevice> dev;
        if(EnableWrmAttack && (src==FirstEndOfWormTunnel)){
            std::cout << "Wormhole attack occuring, redirecting to second end
                of tunnel... \n";
            dev = m_ipv4->GetNetDevice(
                        m_ipv4->GetInterfaceForAddress(SecondEndOfWormTunnel));
            receiver=SecondEndOfWormTunnel;
        }else if(EnableWrmAttack && (src==SecondEndOfWormTunnel)){
            std::cout << "Wormhole attack occuring, redirecting to first end
                of tunnel... \n";
            dev = m_ipv4->GetNetDevice(
                        m_ipv4->GetInterfaceForAddress(FirstEndOfWormTunnel));
            receiver=FirstEndOfWormTunnel;
        }
}
```

And a similar change was also applied to the `RoutingProtocol::ProcessHello(..)` function so that when a node receives an "Hello message" from a neighbor, and said node is one of the end of the tunnel the route is set up to the other end:

```
..
toNeighbor.SetLifeTime(std::max(Time(m_allowedHelloLoss * m_helloInterval),
    toNeighbor.GetLifeTime()));
toNeighbor.SetSeqNo(rrepHeader.GetDstSeqno());
toNeighbor.SetValidSeqNo(true);
toNeighbor.SetFlag(VALID);

//mod
Ipv4Address wrmDst=rrepHeader.GetDst();

if(EnableWrmAttack && wrmDst==FirstEndOfWormTunnel){
    toNeighbor.SetOutputDevice(m_ipv4->GetNetDevice(
        m_ipv4->GetInterfaceForAddress(SecondEndOfWormTunnel)));
    toNeighbor.SetInterface(m_ipv4->GetAddress(
        m_ipv4->GetInterfaceForAddress(SecondEndOfWormTunnel), 0));
}else if(EnableWrmAttack && wrmDst==SecondEndOfWormTunnel){
    toNeighbor.SetOutputDevice(m_ipv4->GetNetDevice(
        m_ipv4->GetInterfaceForAddress(FirstEndOfWormTunnel)));
    toNeighbor.SetInterface(m_ipv4->GetAddress(
        m_ipv4->GetInterfaceForAddress(FirstEndOfWormTunnel), 0));
}else{
    toNeighbor.SetOutputDevice(m_ipv4->GetNetDevice(
        m_ipv4->GetInterfaceForAddress(receiver)));
    toNeighbor.SetInterface(m_ipv4->GetAddress(
        m_ipv4->GetInterfaceForAddress(receiver), 0));
}
toNeighbor.SetHop(1);
toNeighbor.SetNextHop(rrepHeader.GetDst());
m_routingTable.Update(toNeighbor);
..
```

The `RoutingProtocol::RecvAodv(..)` function also needs to be changed so that once the packet is received at the other end of the tunnel it gets properly processed:

```
..
if(EnableWrmAttack){
    if(sender==FirstEndOfWormTunnel && receiver==SecondEndOfWormTunnel){
        std::cout << "Received by Second Wifi Wrm Tunnel \n";
```

```
        receiver=SecondEndWifiWormTunnel;
    }
    if(sender==SecondEndOfWormTunnel && receiver==FirstEndOfWormTunnel){
        std::cout << "Received by First Wifi Wrm Tunnel \n";
        receiver=FirstEndWifiWormTunnel;
}
..
```

B.2.2 Code developed

DoS Attack

Firstly, the nodes are created and added to the corresponding containers in the following way, and the point-to-point connection present in the topology is set up:

```
NodeContainer p2pNodes_c;
p2pNodes_c.Create(2);

NodeContainer wifi_Nodes_c;
wifi_Nodes_c.Create(nBase);

NodeContainer wifi_atk_Node_c;
wifi_atk_Node_c.Create(nAtk);

NetDeviceContainer p2pDevices;
p2pDevices = p2p_h.Install(p2pNodes_c);
NodeContainer wifi_Ap_Node_c = p2pNodes_c.Get(0);
```

Then, the Wi-Fi channel has to be installed, by setting the physical channel and the mac to then install it on the nodes which need to be registered on the Wi-Fi connection; after this the internet stack and IPv4 modules still needs to be installed on the nodes, to correctly configure them:

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
YansWifiPhyHelper phy_ch;
phy_ch.SetChannel(channel.Create());

WifiMacHelper mac_h;
Ssid ssid = Ssid("ns-3-ssid");

WifiHelper wifi;

NetDeviceContainer baseDevices_c;
mac_h.SetType("ns3::StaWifiMac", "Ssid", SsidValue(ssid), "ActiveProbing",
    BooleanValue(false));
baseDevices_c = wifi.Install(phy_ch, mac_h, wifi_Nodes_c);

NetDeviceContainer atkDevices_c;
mac_h.SetType("ns3::StaWifiMac", "Ssid", SsidValue(ssid), "ActiveProbing",
    BooleanValue(false));
atkDevices_c = wifi.Install(phy_ch, mac_h, wifi_atk_Node_c);

NetDeviceContainer apDevices_c;
mac_h.SetType("ns3::ApWifiMac", "Ssid", SsidValue(ssid));
apDevices_c = wifi.Install(phy_ch, mac_h, wifi_Ap_Node_c);
```

```
InternetStackHelper stack_h;
stack_h.Install(wifi_Ap_Node_c);
stack_h.Install(wifi_Nodes_c);
stack_h.Install(wifi_atk_Node_c);
stack_h.Install(p2pNodes_c.Get(1));

Ipv4AddressHelper address_h;

address_h.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces_c;
p2pInterfaces_c = address_h.Assign(p2pDevices);

Ipv4InterfaceContainer baseNodes_Interfaces_c;
address_h.SetBase("10.1.2.0", "255.255.255.0");
address_h.Assign(apDevices_c);
address_h.Assign(atkDevices_c);
baseNodes_Interfaces_c = address_h.Assign(baseDevices_c);
```

To mobility and animation module then had to be setup in the following way:

```
MobilityHelper mobility_h;

UniformDiscPositionAllocator disc_all;
disc_all.SetRho(30.0);
disc_all.SetX(25.0);
disc_all.SetY(25.0);
disc_all.SetZ(25.0);
mobility_h.SetPositionAllocator(&disc_all);

mobility_h.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility_h.Install(p2pNodes_c.Get(1));
mobility_h.Install(wifi_Ap_Node_c);
mobility_h.Install(wifi_atk_Node_c);
mobility_h.Install(wifi_Nodes_c);

AnimationInterface anim("Wifi.xml");
anim.UpdateNodeColor(wifi_atk_Node_c.Get(0),255,255,255);
anim.UpdateNodeDescription(wifi_atk_Node_c.Get(0),"Attacker");
anim.EnablePacketMetadata(true);
anim.SetConstantPosition(p2pNodes_c.Get(1),25,25);
```

And lastly, the behaviour of the nodes is described by configuring and installing the applications needed:

```
//Base behaviour
UdpEchoServerHelper echoServer_h(9);
ApplicationContainer echoserver_App_c =
    echoServer_h.Install(p2pNodes_c.Get(1));
echoserver_App_c.Start(Seconds(1.0));
echoserver_App_c.Stop(Seconds(20.0));

UdpEchoClientHelper echoClient_h(p2pInterfaces_c.GetAddress(1), 9);
ApplicationContainer clientApps_c = echoClient_h.Install(wifi_Nodes_c);
clientApps_c.Start(Seconds(2.0));
clientApps_c.Stop(Seconds(15.0));

//Attackers app + sink for udp packets
OnOffHelper atk_client_h("ns3::UdpSocketFactory",
    Address(InetSocketAddress(p2pInterfaces_c.GetAddress(1),9001)));
```

```
atk_client_h.SetConstantRate(DataRate(ATK_RATE));
atk_client_h.SetAttribute("OnTime",
    StringValue("ns3::ConstantRandomVariable[Constant=15]"));
atk_client_h.SetAttribute("OffTime",
    StringValue("ns3::ConstantRandomVariable[Constant=0]"));
ApplicationContainer atk_apps_c = atk_client_h.Install(wifi_atk_Node_c);
atk_apps_c.Start(Seconds(5.0));
atk_apps_c.Stop(Seconds(20.0));

PacketSinkHelper udpsink_h("ns3::UdpSocketFactory",
    Address(InetSocketAddress(Ipv4Address::GetAny(),9001)));
ApplicationContainer udpsink_App_c = udpsink_h.Install(p2pNodes_c.Get(1));
udpsink_App_c.Start(Seconds(1.0));
udpsink_App_c.Stop(Seconds(20.0));
```

DDoS attack

In this simulation the topology was created similarly to the previous one, minus the fact that we had more attacker nodes and the connections were all Point-To-Point, which can be seen in the following code snippet:

```
NodeContainer legitNodes_c,atkNodes_c;
legitNodes_c.Create(3);
atkNodes_c.Create(nAtk);

//Separate p2p links so atk can be done with different settings
PointToPointHelper p2pLegitNodes_h,p2pAtkNodes_h;
p2pLegitNodes_h.SetDeviceAttribute("DataRate", StringValue("100Mbps"));
p2pLegitNodes_h.SetChannelAttribute("Delay", StringValue("1ms"));
p2pAtkNodes_h.SetDeviceAttribute("DataRate", StringValue("100Mbps"));
p2pAtkNodes_h.SetChannelAttribute("Delay", StringValue("1ms"));

NetDeviceContainer p2pLegitDevices_c[2],p2pAtkDevices_c[nAtk];
p2pLegitDevices_c[0] =
    p2pLegitNodes_h.Install(legitNodes_c.Get(0),legitNodes_c.Get(1));
p2pLegitDevices_c[1] =
    p2pLegitNodes_h.Install(legitNodes_c.Get(1),legitNodes_c.Get(2));

for(int i=0; i<nAtk; i++){
    p2pAtkDevices_c[i] =
        p2pAtkNodes_h.Install(atkNodes_c.Get(i),legitNodes_c.Get(1));
}
```

The internet stack, IPv4 protocol, and address configuration was done similarly to what previously shown.

The different mobility model was build with the following code:

```
MobilityHelper mobility_h;
mobility_h.SetPositionAllocator("ns3::GridPositionAllocator",
                                "MinX",
                                DoubleValue(0.0),
                                "MinY",
                                DoubleValue(0.0),
                                "DeltaX",
                                DoubleValue(20.0),
                                "DeltaY",
                                DoubleValue(5.0),
```

```
                                "GridWidth",
                                UintegerValue(3),
                                "LayoutType",
                                StringValue("RowFirst"));

mobility_h.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility_h.Install(legitNodes_c);
mobility_h.Install(atkNodes_c);

AnimationInterface anim("DDoS.xml");
anim.EnablePacketMetadata(true);

uint32_t x_pos = 0;
for(int i=0; i<nAtk; i++){
    anim.UpdateNodeColor(atkNodes_c.Get(i),255,255,255);
    anim.UpdateNodeDescription(atkNodes_c.Get(i),"Infected node");
    anim.SetConstantPosition(atkNodes_c.Get(i),x_pos,30);
    x_pos+=5;
}
```

Blackhole attack

To setup the Blackhole attack we followed once again the procedure of creating the nodes and their containers, and then we proceed to install the WiFi module, but this time we properly tweaked the Wireless physical settings so that we could get a specific range which was needed for this specific topology in which only adjacent nodes should be in range of each others:

```
WifiHelper wifi_h;
wifi_h.SetStandard(WIFI_STANDARD_80211b);

YansWifiPhyHelper wifiPhy_h;
wifiPhy_h.SetErrorRateModel("ns3::NistErrorRateModel");
wifiPhy_h.SetPcapDataLinkType(YansWifiPhyHelper::DLT_IEEE802_11);

YansWifiChannelHelper wifiChannel_h;
wifiChannel_h.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel_h.AddPropagationLoss("ns3::TwoRayGroundPropagationLossModel",
    "SystemLoss", DoubleValue(1), "HeightAboveZ", DoubleValue(1.5));

// Settings for a range of ~250m
wifiPhy_h.Set("TxPowerStart", DoubleValue(33));
wifiPhy_h.Set("TxPowerEnd", DoubleValue(33));
wifiPhy_h.Set("TxPowerLevels", UintegerValue(1));
wifiPhy_h.Set("TxGain", DoubleValue(0));
wifiPhy_h.Set("RxGain", DoubleValue(0));
wifiPhy_h.Set("RxSensitivity", DoubleValue(-61.8));
wifiPhy_h.Set("CcaEdThreshold", DoubleValue(-64.8));

wifiPhy_h.SetChannel(wifiChannel_h.Create());

WifiMacHelper wifiMac_h;
wifiMac_h.SetType("ns3::AdhocWifiMac");


NetDeviceContainer devices_c;
devices_c = wifi_h.Install(wifiPhy_h, wifiMac_h, nodes_c);
```

After having installed the WiFi module on the network devices, we had to install the internet stack and the routing protocol, in this case AODV, with special attention in installing the malicious version on the attacker node:

```
AodvHelper aodv;
AodvHelper malicious_aodv;

// Setting up internet and routing protocols
InternetStackHelper internet;

internet.SetRoutingHelper(aodv);
internet.Install(not_malicious);

malicious_aodv.Set("EnableBHatk",BooleanValue(enableAtk));
internet.SetRoutingHelper(malicious_aodv);
internet.Install(malicious);
```

Then the IPv4 address and the application to send basic packets from source to destination were installed similarly to what seen in previous examples.

Wormhole attack

For this scenario the network setup was similar to the Blackhole one, aside from the number of nodes and their topology, which simply needed the node to be set in constant positions to match the desired topology structure with the command: `anim.SetConstantPosition(nodes_c.Get(n), X, Y);`

The WiFi model was installed in the same way as the previous attack, and then we added the Point-to-Point connection used as a tunnel between the malicious nodes:

```
..
PointToPointHelper p2p_wormtunnel_h;
p2p_wormtunnel_h.SetDeviceAttribute("DataRate", StringValue("100Mbps"));
p2p_wormtunnel_h.SetChannelAttribute("Delay", StringValue("1ms"));

NetDeviceContainer devices, mal_devices_c;
mal_devices_c = p2p_wormtunnel_h.Install(nodes_c.Get(0),nodes_c.Get(5));
..
```

Additionally, when installing the AODV module the additional parameters needed for the wormhole attack had to be configured:

```
..
AodvHelper aodv;
AodvHelper malicious_aodv;

InternetStackHelper internet;

internet.SetRoutingHelper(aodv);
internet.Install(not_malicious_c);

malicious_aodv.Set("EnableWrmAttack",BooleanValue(enableAtk));
malicious_aodv.Set("FirstEndWifiWormTunnel",Ipv4AddressValue("10.0.1.1"));
malicious_aodv.Set("SecondEndWifiWormTunnel",Ipv4AddressValue("10.0.1.6"));
malicious_aodv.Set("FirstEndOfWormTunnel",Ipv4AddressValue("10.1.2.1"));
malicious_aodv.Set("SecondEndOfWormTunnel",Ipv4AddressValue("10.1.2.2"));

internet.SetRoutingHelper(malicious_aodv);
internet.Install(malicious_c);
..
```

Bibliography

[1] ELPROCUS (ELectronics, PROjects, foCUS), "What is Network Simulation: Types and its advantages", https://www.elprocus.com/what-is-network-simulation-types-and-its-advantages/

[2] R. Patel, M. Pathak, A. Nayak, "Survey on Network Simulators", International Journal of Computer Applications, Vol. 182, No. 21, October 2018 https://www.researchgate.net/profile/Ronit-Patel-2/publication/333262623_Survey_on_Network_Simulators/links/5ce4fe54458515712eba77e0/Survey-on-Network-Simulators.pdf

[3] M. Peruzzini, A. Capitanelli, A. Papetti, M. Germani, "Designing and simulating Smart Home environments and related services", International Design Conference - Design 2014, Dubrovnik (Croatia), May 19, 2014, pp. 1145-1156, https://www.designsociety.org/publication/35259/DESIGNING+AND+SIMULATING+SMART+HOME+ENVIRONMENTS+AND+RELATED+SERVICES

[4] H. Kavak, J. Padilla, D. Vernon-Bido, S. Diallo, R. Gore, S. Shetty, "Simulation for cybersecurity: state of the art and future directions", Journal of Cybersecurity, Vol. 7, No. 1, October 2021, DOI 10.1093/cybsec/tyab005

[5] National Science Foundation, "Cyber-Physical Systems", Document number: nsf21551, https://www.nsf.gov/pubs/2021/nsf21551/nsf21551.htm

[6] NIST Computer Security Resource Center - Glossary, "Cyber-Physical System(s)", https://www.nsf.gov/pubs/2021/nsf21551/nsf21551.htm

[7] J.A. Yaacoub, O. Salman, H.N. Noura, N. Kaaniche, A.Chehab, M. Malli, "Cyber-physical systems security: Limitations, issues and future trends", Microprocessors and Microsystems, Vol. 77, September 2020, ISSN 0141-9331, DOI 10.1016/j.micpro.2020.103201

[8] A. Pentland, "Smart rooms, desks and clothes", 1997 IEEE International Conference on Acoustic, Speech, and Signal Processing, Munich (Germany), April 21-24, 1997, pp. 171-174 vol.1, DOI 10.1109/ICASSP.1997.599589

[9] Oxford Languages Dictionary, https://languages.oup.com/

[10] B. Dvorsak, J. Havelka, E. Mainardi, H. Pandzic, T. Selic, M. Tretinjak, "Smart Home Systems", SHVET project, https://ec.europa.eu/programmes/erasmus-plus/project-result-content/4df4e928-8958-4552-80da-146977e666b9/Smart_Home_systems_FINAL.pdf

[11] Vijay K. Varma, "Wireless Fidelity - WiFi", IEEE Emerging Technology portal, 2006-2012, https://web.archive.org/web/20170829232140/http://www.ieee.org/about/technologies/emerging/wifi.pdf

[12] IEEE Standards Association, "IEEE Standard for Low-Rate Wireless Networks", https://standards.ieee.org/ieee/802.15.4/7029/

[13] M. McGuire, S. Dowling, "Chapter 2: Cyber-enabled crimes - fraud and theft" in the book "Cyber crime: A review of the evidence", Report 75, 2013

[14] M. Plachkinova, A. Vo, A. Alluhaidan, "Emerging Trends in Smart Home Security, Privacy, and Digital Forensics", Twenty-second Americas Conference on Information Systems, San Diego (USA), 2016, https://web.archive.org/web/20200323123821id_/https://aisel.aisnet.org/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1434&context=amcis2016

[15] Open Web Application Security Project (OWASP) Foundation, "OWASP Internet of Things Project - Top 10", https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Top_10

[16] NIST Computer Security Resource Center - Glossary, "Side-Channel Attack", https://csrc.nist.gov/glossary/term/side_channel_attack
[17] E. Ronen, A. Shamir, A. Weingarten,C. O'Flynn, "IoT Goes Nuclear: Creating a Zigbee Chain Reaction", 2018, pp. 171-174 vol.1, DOI 10.1109/ICASSP.1997.599589
[18] "Trendnet Cameras - I always feel like someone's watching me", http://console-cowboys.blogspot.com/2012/01/trendnet-cameras-i-always-feel-like.html
[19] "TRENDnet IP Camera Vulnerability", TRENDnet, https://web.archive.org/web/20120208001237/http://www.trendnet.com/langen/press/view.asp?id=1958
[20] radware, "BrickerBot Results In Permanent Denial-Of-Service", ERT Threat Alert, April 5, 2017, https://www.radware.com/security/ddos-threats-attacks/brickerbot-pdos-permanent-denial-of-service/
[21] C. Cimpanu, "BrickerBot Author Retires Claiming to Have bricked over 10 Million IoT Devices", BleepingComputer, December 11, 2017, https://www.bleepingcomputer.com/news/security/brickerbot-author-retires-claiming-to-have-bricked-over-10-million-iot-devices/
[22] S. De Bique, "The botnet threat against smart refrigerator security", May 2019, Utica College
[23] N. Sidiropoulos, P. Stefopoulos, "Smart TV Hacking", 2013, University of Amsterdam, https://rp.os3.nl/2012-2013/p39/report.pdf
[24] "Ring Video Doorbell Pro Under the Scope", Bitdefender, November 2019, https://www.bitdefender.com/files/News/CaseStudies/study/294/Bitdefender-WhitePaper-RDoor-CREA3949-en-EN-GenericUse.pdf
[25] "Cracking the August SmartLock: WiFi Password Eavesdropping Made Easy", Bitdefender, August 2020, https://www.bitdefender.com/files/News/CaseStudies/study/363/Bitdefender-PR-Whitepaper-AugustConnect-creat4699-en-EN-GenericUse.pdf
[26] "Vulnerabilities in Foscam IP Cameras", F-Secure, https://blog.f-secure.com/wp-content/uploads/2017/06/vulnerabilities-in-foscam-ip-cameras_report.pdf
[27] R. Heartfield, G. Loukas, S. Budimir, A. Bezemskij, J.R.J. Fontaine, A. Filippoupolitis, E. Roesch "A taxonomy of cyber-physical threats and impact in the smart home", Computers and Security, Vol. 78, September 2018, pp. 398-428, DOI 10.1016/j.cose.2018.07.011
[28] N. Komninos, E. Philippou, A. Pitsillides, "Survey in Smart Grid and Smart Home Security: Issues, Challenges and Countermeasures", IEEE Communications Surveys & Tutorials, Vol. 16, No. 4, Fourthquarter 2014, pp. 1933-1954, DOI 10.1109/COMST.2014.2320093
[29] Cisco Packet Tracer, https://www.netacad.com/courses/packet-tracer
[30] COSSIM, https://cossim.org/
[31] EVE-NG, https://www.eve-ng.net/
[32] GNS3, https://www.gns3.com/
[33] IoTIFY, https://iotify.io/
[34] IoTSecSim, https://github.com/kokonnchee/IoTSecSim
[35] MiniCPS, https://github.com/scy-phy/minicps
[36] ns-3, https://www.nsnam.org/
[37] OMNeT++, https://omnetpp.org/intro/
[38] UCEF, https://pages.nist.gov/ucef/
[39] "What is a denial of service attack (DoS)?", Cyberpedia, PaloAlto Networks, https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos
[40] Yih-Chun Hu, A. Perrig, D.B. Johnson, "Wormhole attacks in wireless networks", IEEE Journal on Selected Areas in Communications, Vol. 24, February 2006, pp. 370-380, DOI 10.1109/JSAC.2005.861394
[41] "Cloud Hosted Router: Manual", MikroTik, https://wiki.mikrotik.com/wiki/Manual:CHR
[42] "The Core Project", Tiny Core Linux, http://www.tinycorelinux.net/