UCLouvain

epl

École polytechnique de Louvain

Creating a CoAP DataSet of Normal and Attacked Traffic

Author: **Clide Stell FEUTIO WOUTSOP, 9945-21-00**
Supervisors: **Ramin SADRE, Cristel PELSSER**
Reader:**Jean-Michel DRICOT**
Academic year 2022–2023
Master [120] in Cybersecurity

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Chapter Summary

This chapter explores the rapidly expanding realm of the Internet of Things (IoT), which is reshaping the way we interface with the world by integrating physical devices into a comprehensive Internet communication infrastructure. Despite its promising potential, the limited computational capabilities of IoT devices necessitate the use of lightweight protocols, like the Constrained Application Protocol (CoAP). Although the CoAP protocol is gaining traction for its efficiency, it is not exempt from vulnerabilities. Recognizing the potential security pitfalls, this chapter emphasizes the need to create a dataset combining both normal and attacked traffic. This would aid in scrutinizing potential vulnerabilities, simulating attacks, and evaluating defensive measures. The overarching aim is to bolster the security framework surrounding CoAP and ensure its resilient integration into IoT devices.

The advent of the Internet of Things (IoT) has brought about a paradigm shift in our interactions with the environment, unlocking a multitude of opportunities for diverse applications. It integrates physical devices with the Internet communication infrastructure, enabling end-to-end communication with other devices anywhere on the Internet. By 2025, it is estimated that there will be 75.44 billion IoT devices connected worldwide [2], underscoring the growing prevalence and importance of this technology.

However, the majority of these IoT applications are supported by sensing and actuating devices, which inherently have limited computing power, energy, and memory. These devices are designed to be lean and efficient, often powered with just enough resources to accomplish the tasks they were specifically designed for. As such, their operational capacity and efficiency are reliant on lightweight protocols.

The integration of IoT devices with the broader Internet also presents a unique set of security challenges. Specifically, this connectivity opens up new potential attack vectors from hosts with fewer resources. Therefore, understanding and mitigating these potential vulnerabilities are critical for ensuring the security and functionality of the IoT.

The Constrained Application Protocol (CoAP) is a protocol intended to be used in low-powered devices and networks. It is rapidly growing in usage as it outperforms other similar protocols [36]. CoAP is a REST-based protocol largely inspired by HTTP. However, it brings the Web Server concept to the very constrained space where IoT devices are the ones exposing their resources.

It's important to note that like any other protocol, CoAP is not impervious to vulnerabilities. Despite its unique design and robust performance, potential security loopholes and flaws may exist. Understanding and addressing these vulnerabilities is crucial to ensure the secure and efficient operation of CoAP-enabled devices and networks.

Hence, creating a dataset consisting of both normal and attacked traffic is vital. Such a dataset serves as an invaluable resource for studying potential vulnerabilities and threats, simulating attacks, and testing defensive measures. By analyzing this traffic, we can gain insights into the patterns and characteristics of different types of attacks, which in turn aids in enhancing security measures and building more resilient IoT systems.

We believe that the Constrained Application Protocol (CoAP) deserves special attention in our study because it is expected to play a significant role in enabling efficient end-to-end communication between IoT devices.

1.1 Motivations

There is currently a lack of proposals in the literature that focuses on detecting and dealing with attacks against the security and stability of CoAP devices and communication environments. Our research objectives are threefold:

Understanding CoAP vulnerabilities: The CoAP protocol is relatively new and its vulnerabilities are not yet fully understood. Creating a CoAP dataSet of normal and attacked traffic can help identify vulnerabilities in the protocol and the types of attacks that can be launched against devices and IoT networks using CoAP.

Evaluate anomaly detection methods: Creating a CoAP DataSet consisting of normal and attack traffic can be used to evaluate the effectiveness of machine learning-based anomaly detection methods in detecting malicious CoAP attacks. This can help develop more effective anomaly detection methods to protect IoT devices and networks from attacks.

Improving the security of IoT devices and networks: IoT devices and networks are often used for critical applications such as health monitoring, industrial control, home security, etc. Creating CoAP DataSet of normal and attacked traffic can help improve the security of IoT devices and networks by identifying potential attacks and developing methods to prevent them.

1.1.1 Context

The context of this research is the security of connected objects (IoT) and sensor networks, which are increasingly used in areas such as industry, agriculture, health, home automation, etc. IoT communication protocols such as CoAP are often used to facilitate the exchange of information between sensors and IoT devices, but they can also be vulnerable to malicious attacks, which can lead to disruptions in the normal operation of devices and networks, as well as the leakage or compromise of sensitive data. Therefore, it is crucial to develop effective methods to detect and prevent attacks against IoT communication protocols such as CoAP. The creation of CoAP DataSet composed of normal and attacked traffic is an

important step to facilitate the development and evaluation of such security methods for IoT devices and sensor networks.

1.1.2 Problem statement

IoT devices running the CoAP protocol, are frequently compromised with attacks such as cache manipulations, which make devices vulnerable to man-in-the-middle attacks, Distributed Denial of Service (DDoS) magnified attacks, spoofing attacks, and cross-layer attacks, which may enable firewall bypassing. Furthermore, internal attackers might attempt to compromise the CoAP protocol's semantic guidelines and regular operations.
Creating a reliable and representative CoAP DataSet is a significant challenge as the CoAP protocol is relatively new and examples of malicious traffic are limited, which can make it difficult to evaluate the performance of machine learning-based anomaly detection and security models.
Therefore, it is necessary to create a robust data set of normal and attacked CoAP network traffic which will be used for proper detection and prevention measures by NIDs in the context of Internet-integrated WSN to supplement end-to-end security mechanisms.

1.2 Project statement & contributions

This project seeks to produce a cutting-edge dataset of normal and attacked traffic over the CoAP protocol for evaluating and improving the performance of CoAP-based intrusion detection systems and enhancing the security and reliability of CoAP-based IoT systems. The following highlights this project's primary goals:

- Support research in the area of network security and intrusion detection. This dataset can be used to evaluate the performance of CoAP-based intrusion detection systems (IDS) in detecting and mitigating various types of CoAP attacks, such as message flooding, message fragmentation, and message manipulation.
- Researchers can develop and test new CoAP IDS techniques that can accurately detect and mitigate CoAP attacks while minimizing false positives and false negatives. The dataset can also be used to compare the effectiveness of different CoAP IDS techniques and identify areas for improvement.
- The dataset can be used by CoAP protocol developers to improve the security features of the CoAP protocol and make it more resilient to attacks. The dataset can provide insights into the types of CoAP attacks that are prevalent in real-world scenarios and the weaknesses in the current CoAP security mechanisms.

Overall, the goal of creating a dataset of normal and attacked traffic over the CoAP protocol is to advance the state-of-the-art in CoAP security and improve the security and reliability of CoAP-based IoT systems.

Our contribution is twofold. Firstly, we meticulously established a controlled environment tailored to facilitate the simulation of targeted attacks on the Constrained Application Protocol (CoAP) protocol. Secondly, we undertook the significant task of generating a comprehensive dataset that effectively captures and elucidates the outcomes of network scan attacks, IP spoofing attacks, and DoS attacks. Through the orchestration of these deliberate attacks within our controlled framework, we aim to conduct an in-depth analysis of their implications, encompassing network traffic patterns, system behavior, and response

dynamics. Notably, this endeavor embodies a significant advancement, enhancing our understanding of the intrinsic vulnerabilities within the CoAP protocol, and concurrently providing a crucial resource for future researchers and practitioners.

1.3 Organization of this document

— chapter 2, we dive into understanding the CoAP protocol in detail.

— Chapter 3, we explore a literature review of existing data sets of network traffic-related IoT in general and CoAP protocol in particular. We equally propose a framework that we adopt throughout this research. We end this section by looking at the tools and requirements need for to perform this research project.

— Chapter 4 We design and model our chosen attacks. we experiment with our design by practically performing the attacks in a simulated environment, we then capture the traffic and present our results.

— Chapter 5, We analyze the results obtained by discussing the possible implications and dataset creation procedure.

— Chapter 6: We summarise, conclude, state our research project's limitations, and suggest future work.

Chapter 2
CoAP Protocol and Security

Chapter Summary

This chapter delves into the specifics of the Constrained Application Protocol (CoAP) header format, detailing its structure and components like Version, Type, Token Length, Code, and Message ID. The chapter further elaborates on various CoAP message types, such as Confirmable, Non-Confirmable, Acknowledgment, and Reset. It introduces how CoAP manages request/response communication, comparing it to HTTP methods and responses. A pivotal point discussed is the discovery mechanism, emphasizing its significance in machine-to-machine (M2M) communication and how the CoRE Link Format protocol facilitates this. Additionally, the chapter touches upon CoAP's security aspects, especially its optional integration with DTLS to ensure secure communications in IoT environments. The insights provided are grounded in hands-on experience from IoT product testing and CoAP security research.

The CoAP protocol is gaining traction across numerous industries, and it also intrigues IoT newbies, ardent IoT developers, and security researchers who wish to investigate this technology for potential future applications.

CoAP was created with the intention of incorporating sensors and actuators into the web's REpresentational State Transfer (REST) architecture. The Create, Read, Update, and Delete (CRUD) procedures, as well as the error codes and some URL similarities, are all inherited by the CoAP protocol from ReST in a compressed binary format. While CoAP commonly delegates its support to the Datagram Transport Layer Security (DTLS) protocol to secure communications at the transport layer, it's important to note that the use of DTLS is not obligatory.

CoAP also introduces security measures at the application layer itself, suggesting a versatile approach to security. The necessity for using DTLS or any other security protocol with CoAP would ultimately depend on the specific requirements of the given application or environment. Devices can use public keys, pre-shared keys, or X.509 digital certificates to authenticate using DTLS. Although DTLS provides transparent security for end-to-end communications between devices, it is vital to remember that it does not shield such devices from a variety of external and internal assaults.

Because of straightforward CoAP to HTTP translation and proxying, it can be simply integrated with the Web. Consider it a less-expensive HTTP and it is a suitable protocol choice when working with restricted devices with short battery life and limited processing

power. Table 2.1, shows CoAP's ports and communication types.

UDP Port	Communication Type
5683	Plain text
5684	DTLS

Table 2.1 – CoAP ports and Communication Type

CoAP offers several capabilities, including:

— A straightforward discovery mechanism

— Seamless integration with the Web

— Support for asynchronous message exchange

— URIs to define resources/services

— REST-like request/response model

2.1 CoAP Communication

CoAP communication uses a simple request-response model, similar to HTTP, but with less overhead, making it better suited for constrained networks. The ultimate goal of CoAP communication is to enable seamless and efficient data exchange between IoT devices in a highly scalable and secure manner.

2.1.1 Protocol Overview

CoAP employs the Client-Server communication model, wherein nodes initiate requests and receive responses from other nodes. This process is facilitated through a dual-layer approach, illustrated in figure 2.1, comprising a layer dedicated to the UDP protocol and another focused on application data:

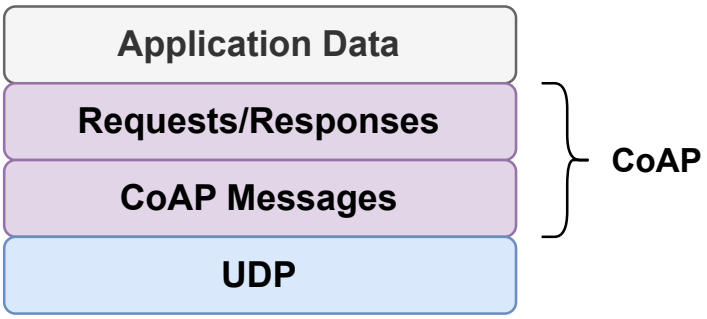

Figure 2.1 – CoAP Protocol Communication Layers [6]

— **CoAP Messages** - These govern the nature of CoAP packets and interface with UDP.

— **Requests/Response (Similar to HTTP)** - These encapsulate the underlying application payload/data using Request Methods and Response Codes.

Within the CoAP framework, the exchange of request/response and application data occurs within messages, with UDP serving as the transport protocol. While the use of ACK (Acknowledgement) messages can confer communication reliability, their implementation remains discretionary, contingent upon the specific requirements of the communication protocol or application.

2.1.2 CoAP Header Format

The CoAP message encoding employs a straightforward binary format. The header of the message contains a fixed-size 4-byte format, and then a variable-length Token value spanning 0 to 8 bytes. Following the Token value, the CoAP Options are presented in a Type-Length-Value (TLV) format, or a series of zeros is indicated for non-options. Subsequently, an optional payload occupies the remaining space within the datagram. The CoAP message format is visualized in 2.2.

2 bit	2 bit	4 bit	4 bit	16 bit
V	T	TKL	Code	Message ID
Token (if any, TKL bytes) ...				
Options (if any) ...				
Payload (if any) ...				

Table 2.2 – CoAP Header Format [30]

The header attributes can be elaborated as follows:

— **Version (V)**: This 2-bit unsigned integer signifies the CoAP version number. The field holds the binary value 01, with reserved values designated for future versions. Messages with unfamiliar version numbers are to be disregarded.

— **Type (T)**: Represented by a 2-bit unsigned integer, where 0 corresponds to Confirmable, 1 to Non-Confirmable, 2 to Acknowledgement, and 3 to Reset, as further detailed in the subsequent section.

— **Token Length (TKL)**: This 4-bit unsigned integer specifies the length of the Token, ranging from 0 to 8 bytes. Lengths from 9 to 15 are allocated for addressing message format errors.

— **Code**: An 8-bit unsigned integer divided into the most significant bits (3 bits) and the least significant bits (5 bits). It adopts the format "c.dd" with "c" representing a digit from 0 to 7 for the 3-bit segment, and "dd" encompassing two digits from 00 to 31 for the 5-bit segment. The most significant bits signify 0 for requests, 2 for successful responses, 4 for client error responses, and 5 for server error responses. Other most significant bit values remain reserved. The code 0.00 signifies an Empty message as a distinct case.

— **MessageID**: This 16-bit unsigned integer serves for duplicate verification and correlates Acknowledgement/Reset messages with Confirmable/Non-Confirmable message types, respectively.

— **Token**: Spanning 0 to 8 bytes based on the length specified in the TKL field, the Token serves to establish associations between requests and responses.

— **Options**: Can be either absent, supplanted by another option, or succeeded by the payload.

— **Payload**: If present, it is preceded by a (0xFF) marker indicating the start of the payload. The payload's length is computed from the marker's end to the termination of the UDP datagram.

2.1.3 CoAP Messages

The CoAP standard specifies 4 different message types:

— **Confirmable Messages (CON)**: These messages demand that the recipient send an acknowledgment message to the sender to verify that the message has been received.

— **Non-Confirmable Message (NON)**: Requires no acknowledgement. When there is little need for acknowledgment, or when dependability is not crucial, this is employed.

— **Acknowledgment Message (ACK)**: This message is dispatched by a recipient to acknowledge the reception of a Confirmable message sent by the sender

— **Reset Message (RST)**: This message is often delivered when a Confirmable or Non-Confirmable message cannot be processed by the recipient because of an error.

To uniquely identify each message and map its related ACK, if any, each message carries a 16-bit message ID. Along with other protocols, this is often used for message identification and deduplication. Figures 2.2 and 2.3 show a few straightforward sequence diagrams that illustrate fundamental message exchange.

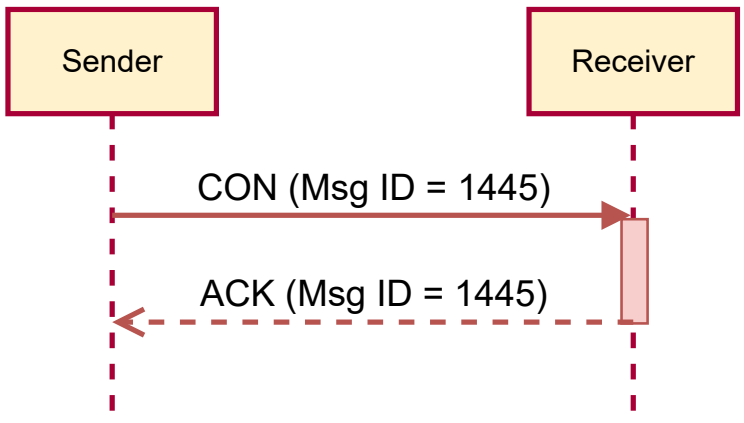

Figure 2.2 – CON Message [6]

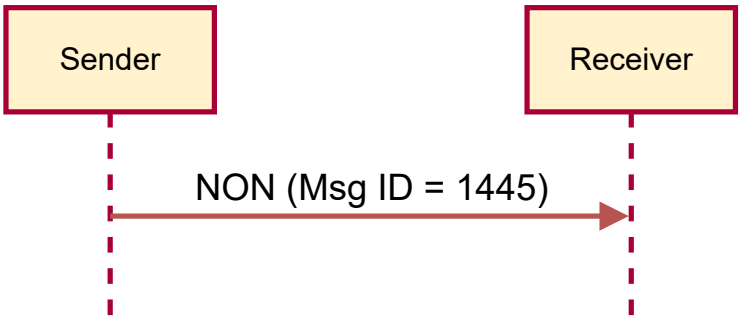

Figure 2.3 – NON Message [6]

2.1.4 CoAP Request/Response

The messages may be empty, contain a request, or a response. Other than the fact that some combinations define certain purposes, or are not permitted, the content of a request or a response is determined independently of the overall message type. Before delving further into the packet format, let's first examine the Request and Respond format.

— Method codes used for requests and responses, are identical to HTTP but defined in binary.

— Four request methods are specified by the CoAP standard: GET, PUT, POST, and DELETE.

— Similar to HTTP [4], the CoAP standard has response codes: 2.xx for success, 4.xx for client error, and 5.xx for server error.

— Requests and responses are each uniquely identified and associated with one another using unique tokens, which can be between 0 to 8 bytes in length. In a similar manner, message-IDs are used to distinguish between different messages and to correctly associate ACKs messages with their original messages.

— Responses to requests can be sent as ACKs messages or standalone CON or NON messages.

To better understand how applications communicate data, let's take a look at some examples of request/response communication.

In figure 2.4, the request is shown as a CON message, and the reply is piggybacked in the ACK Message.

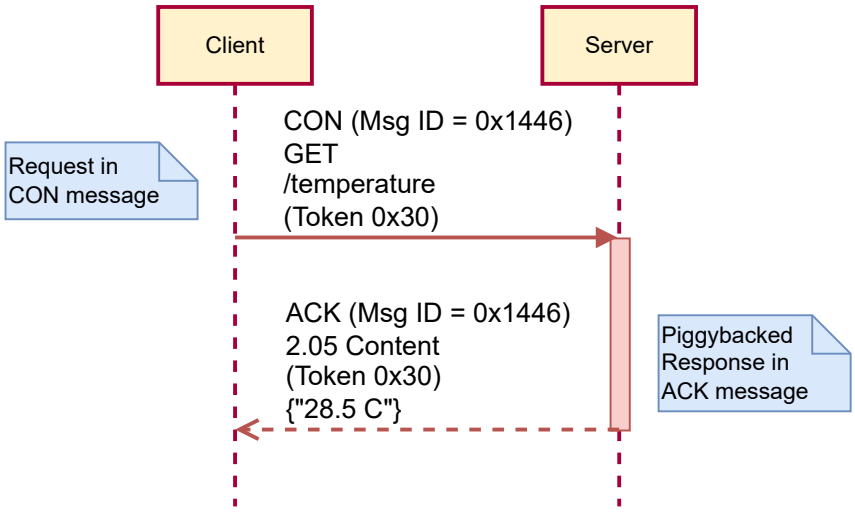

Figure 2.4 – CON Message Request with Piggybacked Response [6]

When the request and response are sent in separate CON messages and the ACK Messages are empty, the lifespan and mapping for Message-IDs and Tokens are shown in the figure 2.5.

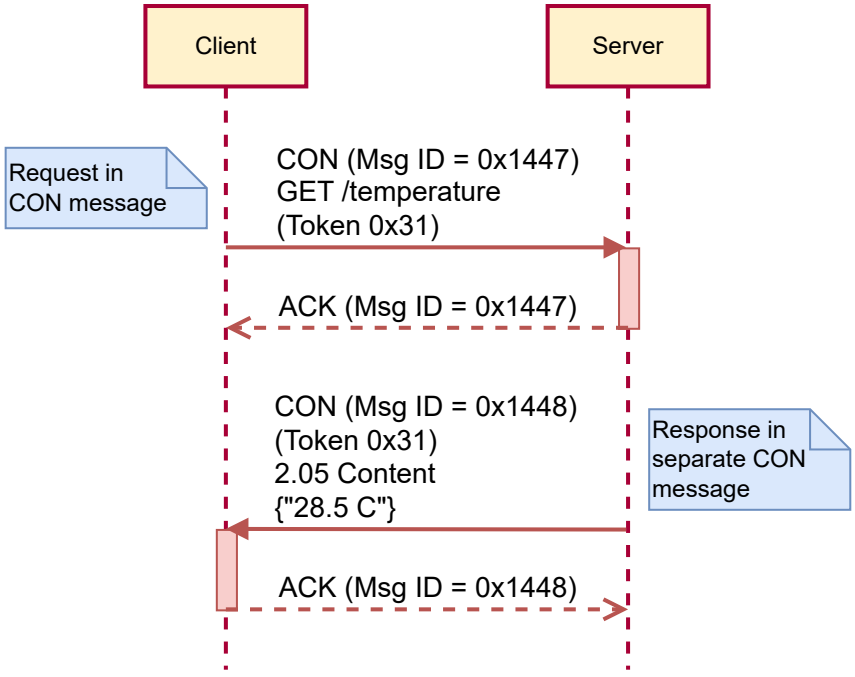

Figure 2.5 – Separate CON Messages for Request and Response [6]

The example of requests and responses being sent in distinct NON messages is shown in figure 2.6.

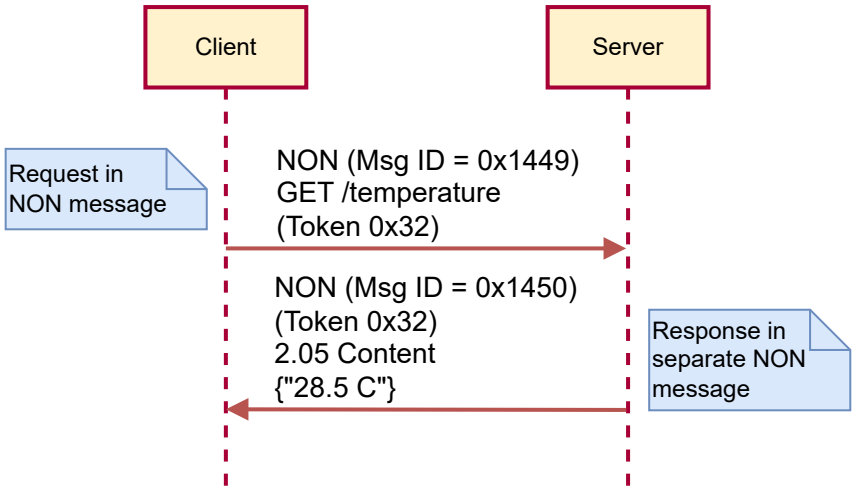

Figure 2.6 – NON Message Request and Response [6]

The CoAP protocol defines certain rules and restrictions for sending messages to ensure the proper functioning of the protocol and to facilitate effective communication between devices. These restrictions might prohibit certain combinations of request/response messages or assign them specific purposes. Here are a few reasons why some combinations might be prohibited or assigned a particular purpose:

— **Protocol Efficiency**: Certain combinations may cause unnecessary network traffic or burden the limited resources of constrained devices, thus they are prohibited to maintain the efficiency of the protocol. For instance, it could be inefficient and unnecessary for a CoAP message to require an acknowledgment if the message is part of a routine, periodic update that doesn't necessarily need confirmation each time. Such unnecessary confirmations could consume precious bandwidth and processing power.

— **Error Prevention**: Some combinations could lead to ambiguity or misinterpretation, causing errors in communication. These combinations are usually prohibited to ensure clear and error-free message exchanges. For example, sending a response message without a preceding request message could cause confusion or misinterpretation, as the recipient may not have the necessary context to process the response. Hence, the protocol might prohibit such combinations to prevent potential errors.

— **Security**: Certain combinations may have security implications, like opening potential attack vectors. By prohibiting these combinations, the protocol enhances its security. A scenario could be when a message type can be used to initiate a new session. If the protocol allowed this type of message to be sent without requiring an acknowledgment, an attacker could potentially flood a device with session initiation requests, leading to a denial-of-service attack. As such, some combinations may be restricted for security reasons.

— **Specific Purpose Assignment**: Certain combinations of request/response messages are designed to fulfill specific purposes in the communication process. For example, a specific combination might be used to establish a new session, request data, or signal an error. An example could be the pairing of a GET request with an Acknowledgement (ACK) message. In CoAP, a GET request is used to retrieve information from a resource on a device. When this request is paired with an ACK, it

signals that the device has received the request and will process it, providing clarity and confirmation in the communication process.

These rules and restrictions are part of the design of the CoAP protocol, which is intended to provide reliable and efficient communication for the Internet of Things (IoT) and other machine-to-machine (M2M) environments where resources may be constrained. Table 2.3 summarizes what is and isn't possible.

CoAP Message	CoAP Message Type			
	CON	**NON**	**ACK**	**RST**
Request	Yes	Yes	No	No
Response	Yes	Yes	Yes	No
Empty	CoAP Ping	No	Yes	Yes

Table 2.3 – CoAP allowed message combinations

The CoAP ping functionality, as depicted in figure 2.7, can be utilized to confirm the server's operational status and overall performance, as recommended by the protocol. CoAP ping communication operates in the following way:

— The sender sends an empty Confirmable message.

— The receiver replies with an empty Reset message.

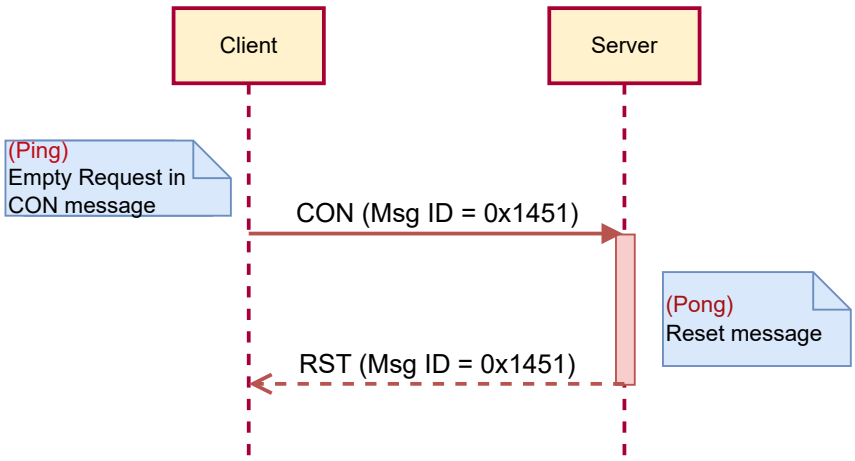

Figure 2.7 – CoAP Ping Communication [6]

2.2 CoAP Request and Response Codes

Codes are used to distinguish between requests and responses. The response codes are derived from HTTP. There are two sections in the code field:

— 3-bit class

— 5-bit detail

Tables 2.4 and 2.5 list requests and response codes respectively in decimal format as such ***c.de***, where ***c*** stands for class and ***de*** for detail.

Code	Description
0.00	Empty message (Special Case)
0.01	GET
0.02	POST
0.03	PUT
0.04	DELETE

Table 2.4 – CoAP Request Codes

Code	Description
2.01	Created
2.02	Deleted
2.03	Valid
2.04	Changed
2.05	Content
4.00	Bad Request
4.01	Unauthorized
4.02	Bad Option
4.03	Forbidden
4.04	Not Found
4.05	Method Not Allowed
4.06	Not Acceptable
4.12	Precondition Failed
4.13	Request Entity Too Large
4.15	Unsupported Content-Format
5.00	Internal Server Error
5.01	Not Implemented
5.02	Bad Gateway
5.03	Service Unavailable
5.04	Gateway Timeout
5.05	Proxying Not Supported

Table 2.5 – CoAP Response Codes

2.3 Discovery Mechanism

CoAP incorporates a discovery mechanism, a feature that it shares with several other IoT protocols. Examples of such protocols include the Message Queuing Telemetry Transport (MQTT) [8], which provides mechanisms for clients to discover and connect to brokers; the Simple Service Discovery Protocol (SSDP) [35], which facilitates the advertisement and discovery of networked devices. The discovery mechanism is a crucial component of machine-to-machine (M2M) communication that promotes automation without requiring human interaction is the discovery mechanism.

The goal of CoAP discovery is to locate the resources that are accessible on a CoAP server.

Constrained **RE**STful Environments (CoRE) Link Format protocol RFC 6690 [29] is used to support it. In order to identify the resources that are available on the server, a client queries a CoAP server that supports a CoAP Link Format of resources. Sending a GET request to the resource path `/.well-known/core`, as required by the standard, accomplishes this. A list of all the resource URIs the server has available is returned in response to the request.

An example from the actual specification is provided below. The list of resource URI items that are present on the server, separated by commas, is contained in the response. Each entry includes:

— The resource URI enclosed in angle brackets, for example, `</sensors>`

— URI attributes, such as `title`, `rt`, and `if`, are separated by `;`.

Request

```
GET coap://[server:port]/.well-known/core
```

Response

```
2.05 Content
</dos/benign>;title="Benign-Resource ?len=0..";rt="Text",
</dos/malicious>;rt="Malicious-Resource ?len=0..";rt="Text"
```

2.4 CoAP Security

Now that we are familiar with how CoAP operates, let's look at the security component of CoAP from both an offensive and a defensive standpoint. The data in the following paragraphs is all based on our experience with IoT product and infrastructure penetration testing initiatives and CoAP security research. In the section below, we'll try to answer some of the common questions that people have when they run IoT penetration testing against new protocols.

2.4.1 DTLS for Secure CoAP

CoAP's integration with DTLS involves a series of adaptations to unicast CoAP, which can be considered the basic foundation for this secure binding. DTLS, in essence, enhances TLS by introducing features specifically designed to tackle the unreliable nature of UDP transport. The information provided in this section is based on the description in the CoAP's standard [30].

when considering constrained nodes or networks, characterized by limited flash or RAM, restricted bandwidth, or high scalability requirements, the applicability of all modes of DTLS may be compromised. This depends greatly on the specific cipher suites employed, as some can introduce substantial implementation complexity and an initial handshake overhead during security association setup.

After establishing the security association, DTLS introduces minimal per-datagram overhead, estimated at around 13 bytes, excluding any potential initialization vectors, nonces, or integrity check values dictated by the cipher suite.

Consequently, the selection of a specific DTLS mode for a CoAP-based application warrants careful evaluation. Key considerations encompass compatible cipher suites, session maintenance alignment with application flows, available resources on constrained nodes, and the associated network overhead.

It's also important to note that DTLS, in its current form, does not apply to group keying, which is necessary for multicast communication. Nevertheless, it is not to be ruled out that DTLS could potentially be incorporated into a future group key management protocol.

Moreover, while some modes of DTLS do specify a mandatory-to-implement cipher suite to ensure maximal interoperability, the actual set of cipher suites used may ultimately be determined by the specific security policies of the application in question.

2.4.2 CoAP Protocol Security/Authentication

Despite our thorough examination of the protocol, it is important to note that we have not yet extensively covered the topic of authentication. This may come as an unexpected gap considering the depth of our preceding discussions. The CoAP standard itself does not specify a particular authentication method but relies on DTLS to provide protocol-level security. For post-provisioning devices, the standard outlines four security settings:

— **NoSec** - DTLS is deactivated, and no protocol-level security is present.

— **PreSharedKey** - In this mode, devices hold a list of pre-shared keys, each associated with nodes it can establish connections. DTLS is enabled in this mode.

— **RawPublicKey** - In this setting, DTLS is active, and devices possess an unsigned, asymmetric key pair. The key's validation employs an Out-of-bounds (OOB) method.

— **Certificate** - Devices operating in this mode possess an X.509 certificate signed by a widely recognized Root CA, an asymmetric key pair, and DTLS enabled.

The CoAP protocol standard [30] provides valuable insights into the security considerations of CoAP. According to the protocol's guidelines, CoAP itself does not inherently provide authentication or authorization features. When such security measures are needed, they can be implemented using communication security protocols such as IPsec or DTLS, or through object security mechanisms integrated within the payload. Object Security is defined by another standard, RFC 8613 [28].

Chapter 3
State of the Art and Requirements

Chapter Summary

This chapter overviews various experiments and studies conducted on the Constrained Application Protocol (CoAP) to analyze and simulate common attacks. Notably, Azad et al., 2021 [7] and Jared, 2022 [20] undertook significant efforts in generating CoAP traffic datasets to simulate and understand DDoS attacks, each having distinct methodologies and dataset sizes. The chapter then introduces the Adopted Framework, which outlines a systematic six-step approach to understanding and simulating attacks on the CoAP protocol, beginning with thread modeling and culminating in dataset splitting. Essential tools and materials were also discussed. Contiki-OS/NG 3.0 was identified as the preferred IoT operating system for the research due to its capabilities and compatibility with the simulation tool Cooja. Other tools, including Scapy and Wireshark, provided the necessary packet manipulation and network analysis capabilities, respectively, for the project. The chapter provides a comprehensive guide to the methodologies, tools, and datasets available for analyzing DDoS attacks on CoAP-based IoT devices.

IoT offers a variety of applications based on the fusion of the Internet and smart devices. The physical components of the infrastructure are integrated by intelligent devices to improve resource management and utilization. The operations and services of many firms are greatly benefited by IoT. The primary objective of the IoT is to offer a digital means of addressing societal needs in routine daily operations. Security, Data Management, Cost, Efficiency, Scalability, Availability, and Application Development are the primary obstacles in IoT networks. Among these difficulties, security is considered one of the biggest obstacles to creating smart infrastructure.
In this section, we will first analyze the most common attacks on the CoAP protocol, which will facilitate our understanding of the study that will follow in a second phase, which is the analysis of existing datasets with a particular focus on the CoAP protocol.

3.1 IP Address Spoofing on UDP

By being built upon UDP, CoAP inherently faces the vulnerability of IP spoofing [30]. Unlike the Transmission Control Protocol (TCP), UDP operates as a connectionless protocol that lacks a handshake phase wherein both endpoints mutually establish sequence numbers to authenticate a connection [19]. This trait entails that in a scenario where an attacker (A) transmits a UDP packet to a recipient (C) with a falsified source IP address (B), C lacks a mechanism to verify the true origin of the packet, whether it's from A or B. When autonomous systems (ASs) do not deploy specific countermeasures against spoofing,

C is left with no choice but to rely on the information provided in the UDP packet header and respond accordingly.
While IP spoofing primarily exploits the absence of authentication within UDP, it's not a major concern in the current context. However, if the application layer employs a request-response protocol, attacker A can manipulate C to function as an unwitting "reflector". In such cases, C might trust the source address B and unwittingly forward response packets to B, who can then manipulate the application layer protocol to its advantage.

3.2 Risk of Amplification

Based on an extensively cited investigation carried out in 2014 [1], it has been established that there are a minimum of 15 application protocols susceptible to amplification attacks, all sharing two pivotal characteristics: they function through UDP and adhere to a request-response model. Similar to other protocols like Domain Name System (DNS) and Simple Service Discovery Protocol (SSDP), CoAP equally employs UDP and, significantly, is structured around a request-response framework. Particularly noteworthy is that the responses within the CoAP protocol can be considerably larger in size compared to the original requests. This significant disparity implies that an attacker possessing, for instance, a bandwidth of 1 Mbps, could potentially leverage this to amplify the bandwidth by a factor of X, rendering such attacks notably efficient [30].

3.3 DDoS Attacks

When an excessive amount of data floods a network, it can result in bandwidth saturation, causing the server to decline processing further requests, thus rendering it inaccessible. This surge of data represents a DDoS attack, a scenario where legitimate users are hindered from accessing the server. The advancement of the Internet of Things (IoT), while groundbreaking in technology, can paradoxically introduce harm. This is due to the potential misuse of IoT devices in contributing to DDoS attacks, leading to the formation of botnets [33]. Consequently, there exists a possibility that CoAP nodes might unintentionally participate in denial-of-service (DoS) attacks, leveraging the protocol's amplifying characteristics as discussed in Section 3.2. An attacker aiming to overwhelm a target but constrained by their own traffic generation capacity can exploit this amplification to create a substantially larger volume of traffic.

3.4 Literature Review of Existing Datasets

In order to alert a control station or take precautionary measures against attacks, an intrusion detection system (IDS) analyzes network traffic for malicious activities or rules breaches [14]. Using a dataset, the IDS may be tested and evaluated. Some of the most common datasets are analyzed in the following paragraphs.

At MIT Lincoln Lab, Lippmann et al, 2000 [10,18] created the DARPA 98/99 datasets using a simulated network environment. While the DARPA 99 dataset [10] includes five weeks of network activity, the DARPA 98 dataset [18] only includes seven days. Even though they are routinely adopted for network intrusion detection, these datasets are commonly criticized because they contain a large number of redundant records. The new dataset,

named KDD99, was created by Lee and Stolfo, 2000 [15] using a framework to build and extract characteristics from the DARPA 98/99 dataset. While the KDD99 dataset [15] includes TCP properties, it omits IP address data.

The previous datasets contain little or no information on IoT networks. However, Nickolaos et al, 2019 [13], came up with the Bot-IoT dataset of legitimate and simulated IoT network traffic captured over the MQTT protocol along with attacks such as DDoS, port scanning, and Botnet-related attacks. Services such as DNS, email, FTP, HTTP, and SSH were deployed on the server along with simulated IoT services in order to mimic real network systems. Ullah et al, 2020 [32] presented a technique used to generate a new Botnet dataset from an existing one, by selecting features and adding new computed features. Although the Bot-IoT [13] dataset contains approximately contain 9543 benign packets and 73360900 attack packets, only one IoT-specific protocol is considered, which is MQTT.

The majority of studies in the existing literature employ some of the datasets stated above to provide security solutions for the IoT environment. Nevertheless, the type of traffic produced by IoT devices in an IoT environment varies depending on the protocol used. For best effectiveness, existing security solutions must be adjusted to take into account more (or specific) IoT protocols.
While there has been much research on IoT malicious traffic detection, few researchers have taken the CoAP protocol into consideration. Hussain et al, 2021 [11] proposed a framework for traffic generation as a context-aware IoT security solution for malicious traffic detection in an IoT healthcare environment, by using IoT-Flock [9, 25] as a generator tool. The following attacks: MQTT Publish Flood, MQTT Authentication Bypass Attack, MQTT Packet Crafting Attack, and COAP Replay Attack, were modeled as well as regular traffic. The data set was further tested using Naive Bayes, K Nearest Neighbors, Random Forests, Matrix of Adaboost, Logistic Regression, and Decision Tree classifiers. Although this data set contains CoAP statistics, it lacks data on common attacks on CoAP protocol.

Al-Fuqaha et al, 2018 [5] described the creation of a CoAP traffic dataset for training machine learning models. They chose CoAP as the communication protocol for the creation of their dataset because it is widely used in IoT devices and can be considered a lightweight protocol suited to the constraints of IoT devices. They also chose a number of common IoT usage scenarios, such as health monitoring devices, environmental sensors, and smart home devices. The CoAP traffic generated by these usage scenarios was then captured using Copper, which is a Google-Chrome extension app to browse the Internet of Things [3]. The traffic data was collected using a network of interconnected IoT devices, which generated traffic data simultaneously and asynchronously using different types of requests and responses, including read requests, update requests, and request responses. The captured traffic data was then pre-processed using a range of pre-processing techniques, such as normalization, sampling, and conversion to CSV format, making it suitable for use in machine learning model training. Finally, the pre-processed CoAP traffic dataset was used to train machine learning models to detect suspicious behavior in IoT traffic. The training was done using several machine learning algorithms, such as neural networks, SVMs (Support Vector Machines), and decision trees to classify the traffic data according to their risk level. However, the paper does not provide a specific methodology for creating the CoAP traffic dataset.

In [27], the authors collected a CoAP traffic dataset to validate their method for detecting CoAP-based attacks in the Internet of Things (IoT). The CoAP traffic was captured using Wireshark, from a network of connected IoT devices using a star architecture which is an architecture in which every host is connected to a central hub [34]. The traffic was captured continuously for 24 hours at different times of the day to account for traffic variations. They captured CoAP traffic data generated by different types of requests and responses, including read requests, update requests, and request responses. Several pre-processing techniques, such as normalization and dimensionality reduction using principal component analysis (PCA), were used to make the dataset usable in the validation of the CoAP-based attack detection method. The authors also added traffic data generated by known attacks to validate the ability of their anomaly detection method to detect CoAP-based attacks. A confusion matrix was used to evaluate the performance of the anomaly detection method, based on the captured traffic. This approach produced a diverse dataset, representing a range of CoAP traffic types, sizes, and frequencies.

Azad et al., 2021 [7], used a CoAP-based IoT device to generate a CoAP traffic dataset. CoAPthon was used as a tool to generate different types of CoAP requests and responses from the IoT device. They also simulated DDoS attacks by making use of requests of different types, sizes, intensities, frequencies, and generated traffic from different IP addresses to add traffic data generated by DDoS attacks to their CoAP traffic dataset. This allowed for the creation of a diverse and representative CoAP traffic dataset. The dataset was then pre-processed using techniques such as normalization and feature extraction with discrete wavelet transform (DWT) to extract temporal, frequency, and time-frequency features from the CoAP dataset. The features were then used to train an artificial neural network (ANN) classifier for detecting DDoS attacks on CoAP-based IoT devices.

Jared 2022, [20], created a useful CoAP DDoS dataset using physical devices. A Raspberry Pi as an IoT CoAP server, and three computer clients including two that were malicious. This dataset contains 661,304 benign and malicious packets captured over 16 hours of network traffic. But only focuses on one attack (DDoS) among several attacks that can be performed on the CoAP protocol.

3.5 Adopted Framework

Figure 3.1 shows the six defined steps: thread model, attack simulation, traffic generation, traffic capturing, dataset creation, and dataset splitting.

Beginning from the first step, the initial goal was to model the most common attacks on the CoAP protocol. This required extensively studying the literature so as to properly identify and model these attacks. By taking an attacker's posture, we focused our project on three main attacks in sequential order so as to properly depict a real-life scenario. These attacks include Network Scans, IP Spoofing, and Denial of Service.

Once the attack vectors were identified, the subsequent phase involved a comprehensive analysis of their potential consequences via simulation. This methodology draws inspiration from existing research endeavors that utilize simulations to assess the potential

network impact of various attacks. As a result, we followed a similar approach and meticulously executed each attack as outlined in chapter 4. We equally performed simulations for normal scenarios with no attacks, as this permitted us to compare our results. Executing our simulation enabled us to generate traffic which was then captured, analyzed, and split into Benign and malicious traffic.

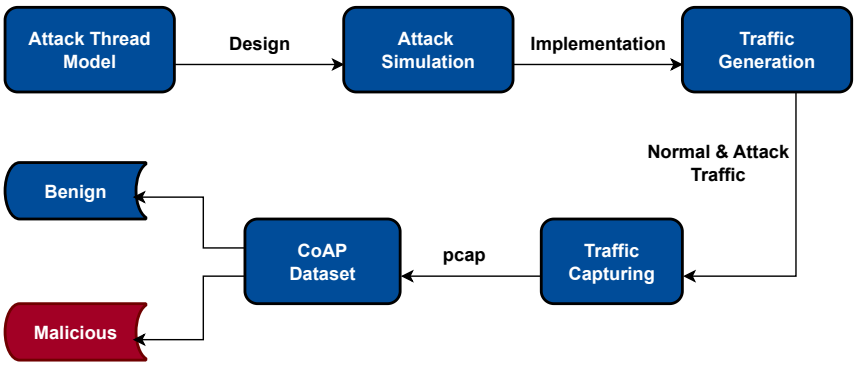

Figure 3.1 – Adopted Framework

3.6 Requirements, Environment, Tools & Materials

CoAP is an application layer protocol and thus sits at Layer 7, the top of the Open Systems Interconnection (OSI) model. CoAP is by default bound to UDP to transport information, thus CoAP relies on UDP security features to protect information. For this reason, all our requests or packets sent in the context of this research are UDP based.

Contiki-OS/NG 3.0: The Contiki operating system, an open-source platform tailored for connected devices and the Internet of Things (IoT), played a pivotal role in this project. It facilitated the design, development, and practical demonstration of our proposed solution within a genuine environment. An evaluation of diverse IoT operating systems is presented in Table **??**, revealing ContikiOS/NG's distinct advantages for IoT research. Notably, its support for both multithreading and event-driven paradigms, along with its modular architecture, underscores its suitability. Additionally, its compatibility with simulation tools such as Cooja [24] contributed to our decision to adopt ContikiOS/NG as the foundation for implementing our solution in this thesis.
Philokypros, 2021 [12] performed a comparative analysis of different Internet of Things (IoT) operating systems. The table presents a breakdown of essential attributes for TinyOS, ContikiOS/NG, RIOT, and FreeRTOS. These attributes encompass the system's architectural design, programming model, programming language, and memory allocation method. Each operating system showcases unique characteristics in terms of its structure, programming approach, and memory management strategy.
Cooja : Within the realm of practical experimentation, Cooja [24] and TOSSIM [16] emerge as particularly well-suited options. This preference stems from their capacity to seamlessly deploy developed applications onto actual hardware. TOSSIM, a simulator associated with TinyOS [17], offers the capability to craft applications using the nesC language. Meanwhile, Cooja excels in replicating the behavioral dynamics of ContikiOS [23]. Notably flexible, Cooja enables nodes within the same network to execute disparate software or operate on distinct hardware platforms. A comparative analysis of prevalent simulators

provided by Philokypros, 2021 [12], gives evidence that the majority of simulators do not authentically replicate real traffic via actual application execution. In this context, Cooja stands out by generating genuine network traffic, a trait attributed to its foundation on the ContikiOS system.

Scapy: Scapy is an open-source network packet manipulation tool, which allows creating, sending, capturing, and analyzing network packets. It is written in Python and offers a simple and flexible interface for creating and manipulating packets, making it ideal for creating custom tools for testing and debugging network protocols. Scapy provides functionality for CoAP packet manipulation. This makes it possible to create custom CoAP packets for compliance testing, network simulation, or performance evaluation.

Wireshark: Wireshark is a very popular and powerful open-source network protocol analysis tool. It allows you to capture, analyze and visualize network traffic in real-time, as well as examine previously saved capture files. Wireshark is capable of decoding a wide variety of network communication protocols including CoAP protocol, and provides an intuitive graphical user interface (GUI) to display network traffic data in tabular and graphical form. It also allows you to apply filters to isolate relevant packets and track connections between different protocols.

Chapter 4
Attacks Design and Implementation

Chapter Summary

This chapter provides an in-depth exploration of three salient cyber threats targeting CoAP (Constrained Application Protocol) networks: Network Scan Attacks, IP Spoofing, and Denial of Service (DoS) attacks. Beginning with Network Scan Attacks, the chapter elucidates how attackers methodically scan networks to uncover vulnerabilities, laying the groundwork for subsequent exploits. IP Spoofing delves into the concealment tactics used by cyber adversaries to obscure their identities, complicating traceability efforts. Lastly, the Denial of Service section focuses on the menacing CoAP amplification technique, aiming to cripple legitimate services. Through meticulous design and implementation details, bolstered by illustrative simulations, this chapter underscores the significant vulnerabilities inherent in CoAP networks, emphasizing the urgent need for robust security countermeasures in contemporary IoT landscapes.

With the foundational concepts and recent advancements having been elucidated, the subsequent phase involves a comprehensive analysis of the repercussions posed by IoT-centric attacks within a network. Delving into the ramifications of these attacks constitutes a preliminary stride, essential prior to the formulation and execution of defensive strategies. This section delineates the strategic blueprint and practical enactment of IoT-based attacks targeting the CoAP protocol. Furthermore, it encompasses a detailed exposition of the simulations conducted for the identified attacks, coupled with an elucidation of the outcomes yielded by these simulations.
The implementation goals include: Setting up a client-server topology on Cooja (see section 3.6), generating and capturing normal and attacked CoAP traffic, analyze the captured traffic to extract relevant features.

We made the following general assumptions for the attacks described in this research project:

— The target device or network is vulnerable to the type of attack being performed.

— The attacker is able to maintain persistence throughout the attack, meaning they are able to continue the attack for a prolonged period of time.

— No CoAP packets are lost or unnecessarily delayed in the network. Moreover, the running CoAP applications do not support DTLS.

4.1 Network Scan Attack

In this section, the details about designing and implementing network scan attacks are presented. This attack could also be considered as **reconnaissance** since the attacker's aim is to gather as much information as possible from the network and the connected devices by performing an **active** network scan to identify all devices and services on a network in order to determine potential vulnerabilities and attack vectors.

4.1.1 Design

The attacker is located out of the system network. Although this attack is more about the transport layer than the application layer, we implemented this attack because an attacker will first scan the network to discover running hosts and opened ports. In order to remain within the context of an application layer protocol, we assume the attacker has already performed a basic network scan and knows at least one IP address in the network running the CoAP protocol. The attacker can then deduce other IP addresses by incrementing or decrementing the identified IP address. This assumption is based on the fact that in most cases network engineers allocate IP addresses to devices in a network in an incremental manner.
Once a list of IP addresses running the CoAP protocol has been retrieved, the attacker then tries to identify the different services or resources by sending a GET request to each IP address in the list, using `/.well-known/core` url (see section 2.3). If we get a list of resources with the response code *2.05*, then it is a server node otherwise it is a client and we get bad request response code *4.00*. If there is no device active or accepting connections at that IP address, then we get a Connection Timeout.

4.1.2 Implementation

We set up the system as shown in Figure 4.1. The RPL Border Router [22] provides a public interface to the CoAP servers, allowing the attacker to send a GET request through the internet to the CoAP Servers.

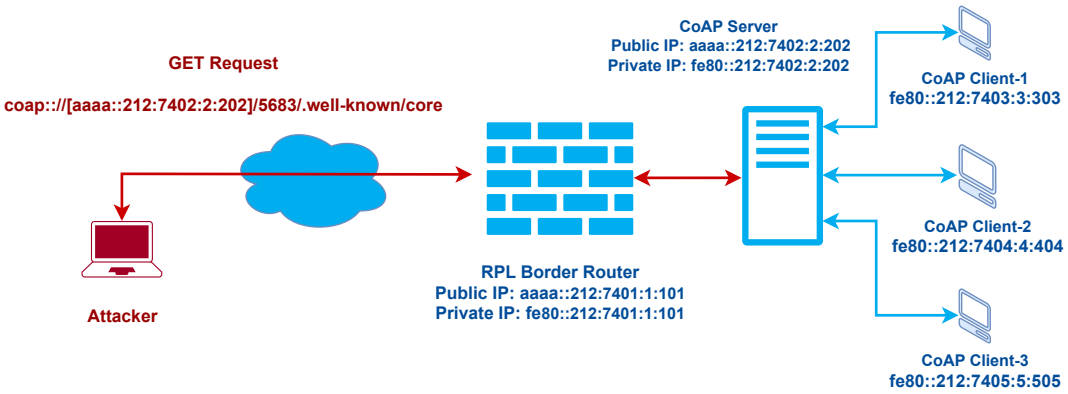

Figure 4.1 – Network scan attack

4.1.3 Scenarios and Topology

Figure 4.2 shows the adopted topology for the scan attack simulated in Cooja. The **red node** represents the attacker, the **green node** represents the Client, the **blue node**

represents the server and the **Orange node** represents the RPL border router. Cooja provides an implementation of the RPL protocol, called ContikiRPL [31].
The orange circle shows the internal network range of the border router. Only the server node can directly communicate with the border router. By so doing, internal clients are protected from external attacks through the router and a firewall can easily be setup between the server and the router. The attack is illustrated by the bidirectional red arrow as the attacker receives a response after forwarding a GET request.
Within the normal context, no nodes suffer from compromise. Each individual node is programmed to dispatch messages to the designated server (referred to as node 2) at predetermined intervals. In the alternate scenario, node 6 assumes a malicious role as the attacker, orchestrating a scan attack.

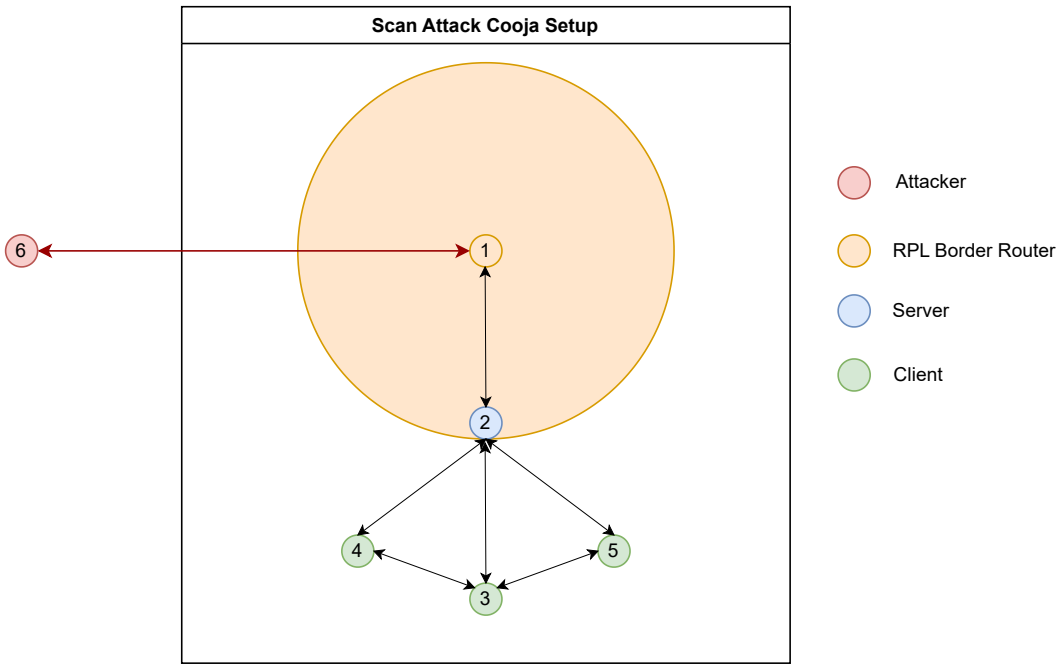

Figure 4.2 – Network scan attack topology

4.1.4 Simulation Parameters

The Cooja simulator, elaborated upon in section 3.6, was harnessed for comprehensive testing and experimentation, increasingly favored among IoT researchers. Its applicability extends effectively to real-world simulations, as the developed applications possess the versatility to be directly uploaded onto tangible hardware. Cooja's capacity to emulate ContikiOS behavior [23] is noteworthy. Table 4.1 outlines the employed node types and corresponding configurations. Within the network, the server functions as the message recipient, thus remaining perpetually operational. Configurable parameters within the Cooja simulator encompass diverse aspects. Our experiments, for instance, encompass a simulation time of 5 minutes per scenario. Another advantageous feature is the incorporation of mote startup delays, allowing nodes to initiate booting at distinct intervals to simulate real-world conditions. By default, a 1000ms startup delay is established in Cooja. A significant determinant, the random seed value, is automatically generated during the simulator's initialization phase. This seed value critically influences the behavior of nodes, including packet transmission timing.

Parameter	Description	Values
Border Router Public IP	The public IP address of the router	aaaa::212:7401:1:101
Border Router Private IP	The private IP address of the router	fe80::212:7401:1:101
Target IP Range	The range of IP addresses targeted	aaaa::212:7401:1:101 - aaaa::212:7409:9:909
CoAP Port	The UDP port used for CoAP	5683 (default CoAP port)
Request Type	The type of CoAP request sent	GET, POST
Timeout	Time to wait for a response before giving up	5s

Table 4.1 – Simulation Parameters for Scan Attack

4.1.5 Results

Upon executing the network scan attack simulation, several key observations and results were obtained. This section delves into the detailed outcomes and the interpretations of those results.

When the attacker sent GET requests to deduced IP addresses running the CoAP protocol, the response distribution varied significantly. One of the addresses (aaaa::212:7402:2:202) responded with the 2.05 response code, indicating it was a server node with resources available as illustrated in Figure 4.3. However, most of the addresses didn't provide any response, suggesting either they were not assigned or they were unreachable.

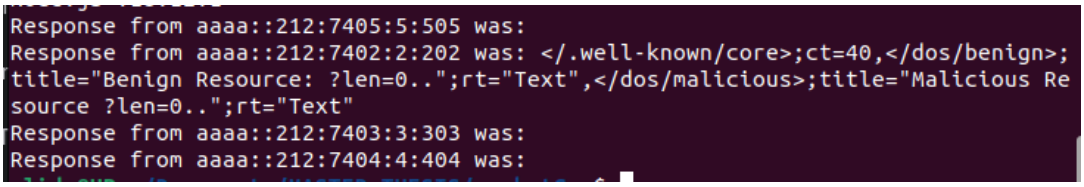

Figure 4.3 – Scan Output

In the implemented scenario, as depicted in Figure 4.2, the orange internal network range circle showed clear communication boundaries. The server node (blue) could directly communicate with the RPL border router (orange), effectively serving as the gateway for other nodes. This setup confirms that internal client nodes, represented in green, remain shielded from direct external attacks hence, explaining the unreachability of certain nodes and further emphasizing the importance of a well-configured border router and firewall between the server and the router.

4.2 IP Spoofing Attack

In this section, we provide the details underlying the conception and execution of IP address spoofing attacks. Such attacks frequently find utilization as integral components of broader Denial-of-Service (DoS) and Man-in-the-Middle attack strategies. The fundamental objective underlying IP spoofing maneuvers revolves around obfuscating the identity of the attacker, thereby introducing complexities in the endeavor to ascertain the origins of the assault.

4.2.1 Design

IP spoofing attack may be initiated either by an on-path or an off-path adversary. An on-path attacker positions themselves along the route linking the client and the server node, whereas an off-path attacker operates outside the established CoAP message delivery trajectory. Our current discourse is centered on the off-path attack scenario, exploring its attributes, behaviors, and ramifications within the network environment.

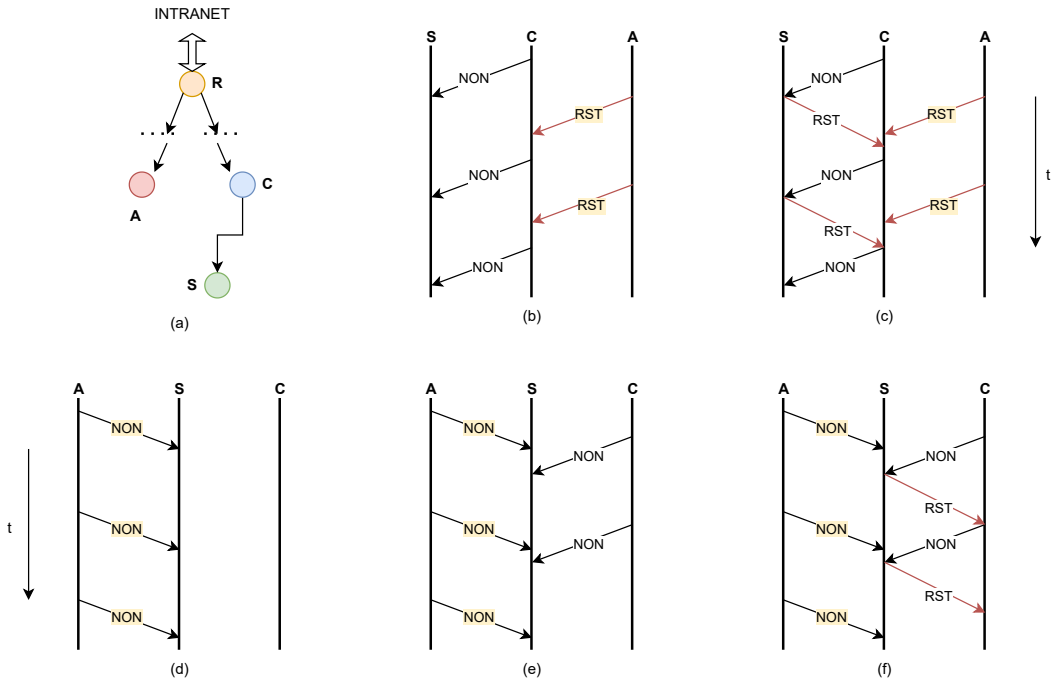

Figure 4.4 – IP spoofing attack design
[26]

Figure 4.4(a) illustrates a network composed of resource-constrained devices featuring an off-path internal attacker labeled as A, a client node designated as C, a server node denoted as S, and the root node (Border Router) indicated as R.

1. **Response spoofing**: The chronological sequence of events related to response spoofing is depicted in Figure 4.4(b) and Figure 4.4(c). In the initial scenario, C legitimately sends NON messages to S. Exploiting precise timing, attacker A sends a spoofed RST message to C, mimicking the server. C, deceived by the falsified message, repeatedly retransmits the packet, resulting in a Denial of Service (DoS) situation. Consequently, packets sent by C to server S, under the influence of A's deception, are regarded as unauthorized. In the subsequent scenario, S issues an

authentic RST message in response to the falsely received packet. However, as A has previously initiated an RST message to C, the CoAP's duplicate detection mechanisms can detect this fraudulent activity.

2. **Request spoofing**: The sequence of events characterizing the request spoofing attack is presented in Figures 4.4(d)-(f). In the first scenario, A persistently sends NON messages using C's identity, leading to an illegitimate flood of messages to the server. The second case showcases A preemptively sending a fake NON message before C's legitimate message reaches S. While the overall network overhead remains similar to the prior scenario, C never gets to know about the ongoing attack. In the third instance, an RST message is received by C due to A's deceptive message sent in the guise of C's actions.

4.2.2 Implementation

Using the results obtained from the scan attack 4.1, the attacker now knows the IP address of the devices connected to the network.

Figure 4.5 demonstrate the response spoofing implementation. The attacker starts by sniffing the client's requests toward the server in order to perform his spoofing attack. The attacker then identifies and times a CON message request from the client and responds before it reaches the server by sending an RST message request back to the client.
The client upon receiving the RST message thinks it is coming from the server, and will therefore retransmit the packet to the server. The server receives this illegitimate packet and responds with an RST message to the client.

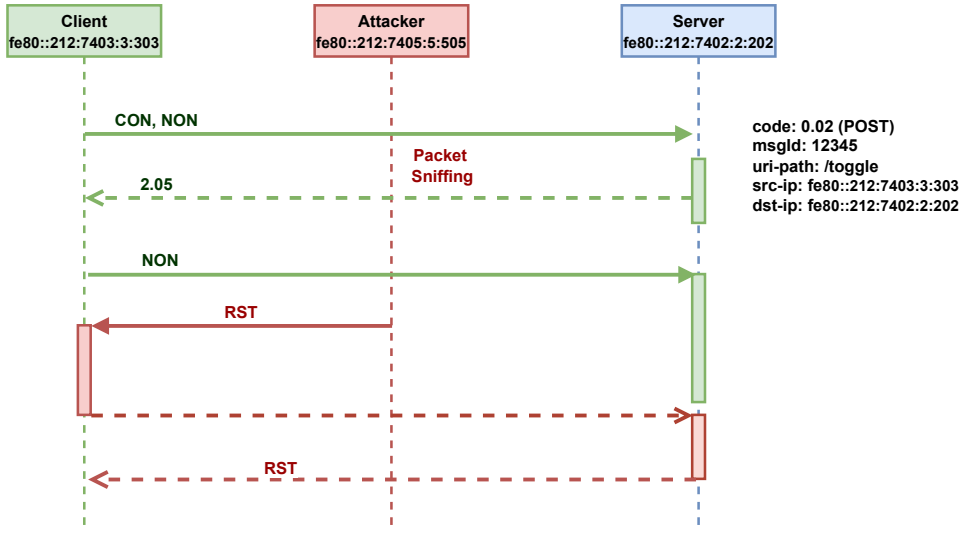

Figure 4.5 – Response spoofing attack

Figure 4.6 describes the request spoofing attack. After the attacker successfully sniffs a legitimate request from the client to the server, the attacker then carefully crafts a packet by spoofing the client's IP address. This packet crafted by the attacker, is then sent to the server. The server is deceived and responds to the client.

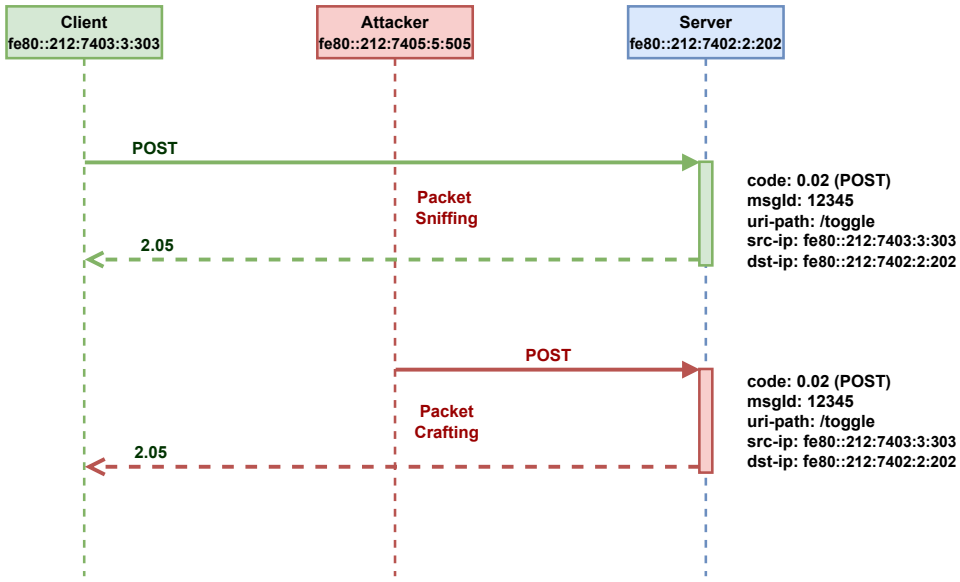

Figure 4.6 – Request spoofing attack

The simulation described in this section is focused on request spoofing attacks as it is the most common form of IP spoofing.

4.2.3 Scenarios and Topology

In figure 4.7(a), the attacker is external to the network represented by node 5 while in figure 4.7(b), we have an internal attacker represented by node 5. This simulation encompassed two primary scenarios: normal and malicious conditions. In the normal scenario, nodes are in regular operation without the presence of any malicious attackers. In contrast, the malicious scenario involves a solitary attacker. This scenario is studied to understand the impact that IP spoofing has on devices and networks as a whole.

In the case of an external attacker, the attack is launched manually and packets are sent every 20s whereas in the case of an internal attacker, the malicious node is configured to join the network automatically and start sending packets every 20s. In both cases, node 4 is spoofed.

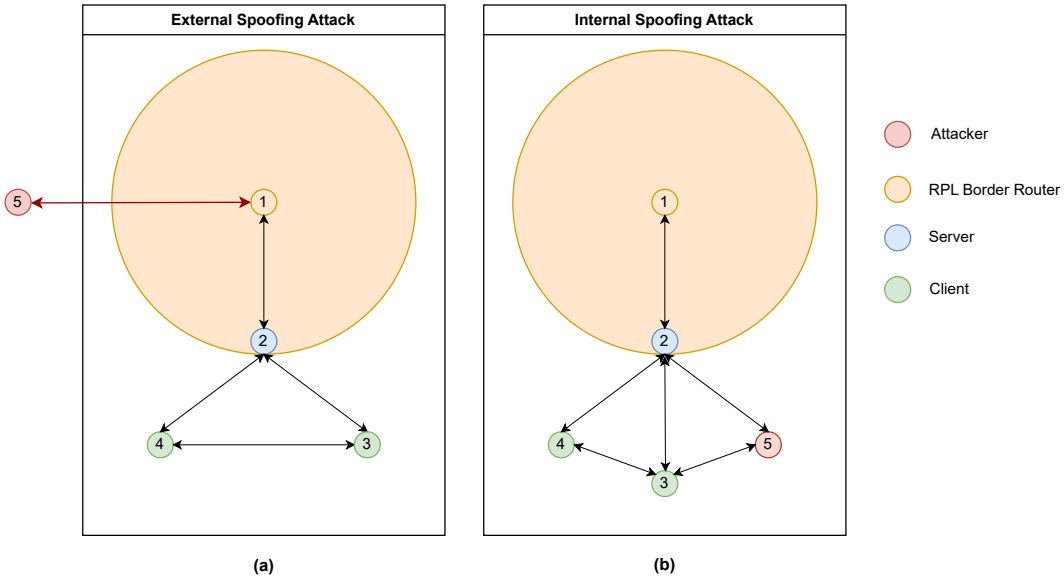

Figure 4.7 – IP spoofing attack topology

4.2.4 Simulation Parameters

Table 4.2 shows the simulation parameters for an IP spoofing attack in the Cooja network simulator. These parameters were meticulously set to analyze the attack's effects in two distinct scenarios. First, we simulated an internal attack scenario where a malicious node within the network undertook the spoofing activity. This scenario is representative of situations where an authorized network participant with malevolent intent leverages their network access to perform the attack. Secondly, we created a simulation that modeled an external attacker. This scenario represents instances where an entity outside the network perimeter attempts to spoof an IP address to trick the network's nodes or bypass security measures. Both scenarios enabled us to comprehensively understand the severity and potential impacts of IP spoofing attacks in network environments.

Parameter	Description	Values
Spoofed IP Address	The IP address that is being spoofed	fe80::212:7404:4:404
Attacker IP Address	The actual IP address of the attacker	Internal - fe80::212:7405:5:505, external - 2a02:a03f:a0cd:e400:60e:d5dc:e733:9421
Target IP Address	The IP address that is targeted by the attack	fe80::212:7404:4:404
CoAP Port	The UDP port used for CoAP	5683 (default CoAP port)
Request Type	The type of CoAP request sent	POST
Attack Frequency	How often the spoofing attack is performed	Every 10s
Timeout	Time to wait for a response before giving up	5s

Table 4.2 – Simulation Parameters for IP Spoofing Attack

4.2.5 Results

Upon completing the simulation of the IP spoofing attack, we observed that the malicious node performing an IP spoofing attack is quite difficult to detect as it perfectly mimics normal devices' operations. Figure 4.8 shows the results of the implementation of the IP spoofing attack performed by the internal malicious node 5 as illustrated by the network topology in Figure 4.7(b). We can observe more packets from the IP address (fe80::212:7404:4:404) of the spoofed node 4 as compared to packets from the normal node 3 with IP address fe80::212:7403:3:303.

No.	Time	Source	Destination	Protocol	Length	Info
122	1.157000	fe80::212:7404:4:…	fe80::212:7402:2:…	CoAP	86	CON, MID:38800, POST, /actuators/toggle
123	1.213000	fe80::212:7404:4:…	fe80::212:7402:2:…	CoAP	86	CON, MID:38801, POST, /actuators/toggle
124	1.228000	fe80::212:7404:4:…	fe80::212:7402:2:…	CoAP	86	CON, MID:38801, POST, /actuators/toggle
125	1.345000	fe80::212:7403:3:…	fe80::212:7402:2:…	CoAP	71	CON, MID:39278, POST, /actuators/toggle
126	1.447000	fe80::212:7404:4:…	fe80::212:7402:2:…	CoAP	71	CON, MID:6266, POST, /actuators/toggle
127	1.447000	fe80::212:7404:4:…	fe80::212:7402:2:…	CoAP	71	CON, MID:6267, POST, /actuators/toggle
128	1.507000	fe80::212:7404:4:…	fe80::212:7402:2:…	CoAP	86	CON, MID:38801, POST, /actuators/toggle
129	1.507000	fe80::212:7404:4:…	fe80::212:7402:2:…	CoAP	71	CON, MID:6267, POST, /actuators/toggle
130	1.607000	fe80::212:7404:4:…	fe80::212:7402:2:…	CoAP	71	CON, MID:6267, POST, /actuators/toggle
131	1.792000	fe80::212:7404:4:…	fe80::212:7402:2:…	CoAP	86	CON, MID:38801, POST, /actuators/toggle
132	1.922000	fe80::212:7403:3:…	fe80::212:7402:2:…	CoAP	71	CON, MID:39278, POST, /actuators/toggle
133	1.974000	fe80::212:7403:3:…	fe80::212:7402:2:…	CoAP	71	CON, MID:39279, POST, /actuators/toggle
134	2.027000	fe80::212:7404:4:…	fe80::212:7402:2:…	CoAP	71	CON, MID:6267, POST, /actuators/toggle

Figure 4.8 – Internal malicious node IP spoofing traces

4.3 Denial of Service Attack

A Denial of Service Attack (DoS) is geared towards inducing service downtime, rendering the service inaccessible to legitimate users. One of the Denial-of-Service attacks that target the CoAP protocol is the so-called **CoAP amplification**. The amplification attack sends unwanted messages to a target victim using a spoofed IP address. Essentially, an attacker uses an intermediary tool such as Scapy to spoof a victim's IP address, craft a packet and send a relatively small amount of data to a server, causing it to send a significantly larger amount of unwanted data to the victim. This discrepancy between the initial data sent by the attacker and the large volume of response data sent to the victim is where the term "amplification" arises.

4.3.1 Design

An attacker can either send a large number of requests or responses to a CoAP server. The denial-of-service might be caused by the server receiving a large amount of data, sending a large amount of data, doing heavy processing, or using too much memory.

Given that the CoAP protocol is built for lightweight devices with limited resources, it's optimized for minimal overhead. This makes it a prime target for amplification in the following ways:

— **Large Response Payloads**: In standard configurations, the CoAP protocol is designed to process requests that may yield responses with substantial payloads. An attacker can craft a request that elicits a maximum-size response from a CoAP server, even if the initial request was quite small.

— **Stateless Nature**: The CoAP protocol, in its stateless mode, doesn't maintain a continuous connection. Attackers can exploit this by spoofing a client's IP address and sending a barrage of requests to a CoAP server, which then directs the large responses toward the spoofed IP.

The amplification attack can manifest as either a DoS (Denial-of-Service) or DDoS (Distributed Denial-of-Service) attack, contingent on the attack's source. In the context of a DoS attack, a single endpoint is used to initiate extensive requests or responses toward the target server, engendering substantial processing demands that inundate the server's resources and render it inaccessible to legitimate clients. In contrast, a DDoS attack employs multiple sources instead of a single endpoint. The amplification factor, a pivotal metric for amplification attacks, can be computed by comparing the volume of data generated by the attacker against the actual data dispatched to the victim [21].
The attack scenario discussed in this section represents a **simple amplification attack**. In this scenario, the attacker sends either a single or numerous CON message requests, and in return, the victim receives either a single or multiple responses. In the context of a DoS attack scenario, the CoAP server is coerced into transmitting a large data load. However, amplification attacks are paralleled by the IP address spoofing of the victim. As a result, the attacker gains the capability to generate a higher number of requests from various servers toward the victim, multiplying the traffic and resulting in a DDoS amplification attack.

4.3.2 Implementation

By spoofing a client's IP address, the attacker forces the CoAP server to respond x times to a single request by updating the server's resources using POST or PUT requests. These requests are aimed at increasing the server's responses. If the responses are x times larger than the request, then:

$$\text{Amplification factor} = x$$

In its standard configuration, CoAP is designed to process up to 1024 bytes for each request [30]. In this scenario, an attacker transmits a message that surpasses this limit. An amplification attack using a singular response is depicted in Figure 4.9. This assault will only triumph if the IP of the victim's CoAP client is effectively spoofed.

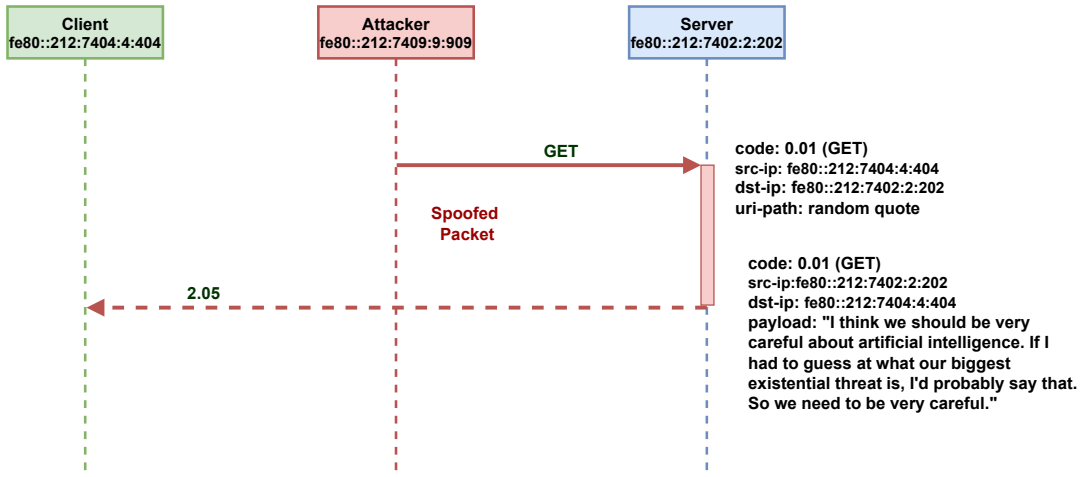

Figure 4.9 – Simple amplification against the CoAP

A potential attacker possesses the capacity to manipulate the amplification factor to a predefined level by increasing the quantity of GET requests aimed at the targeted victim. This can be achieved through adjustments to the resources engaged in the offensive action. This specific situation is depicted in Figure 4.10.

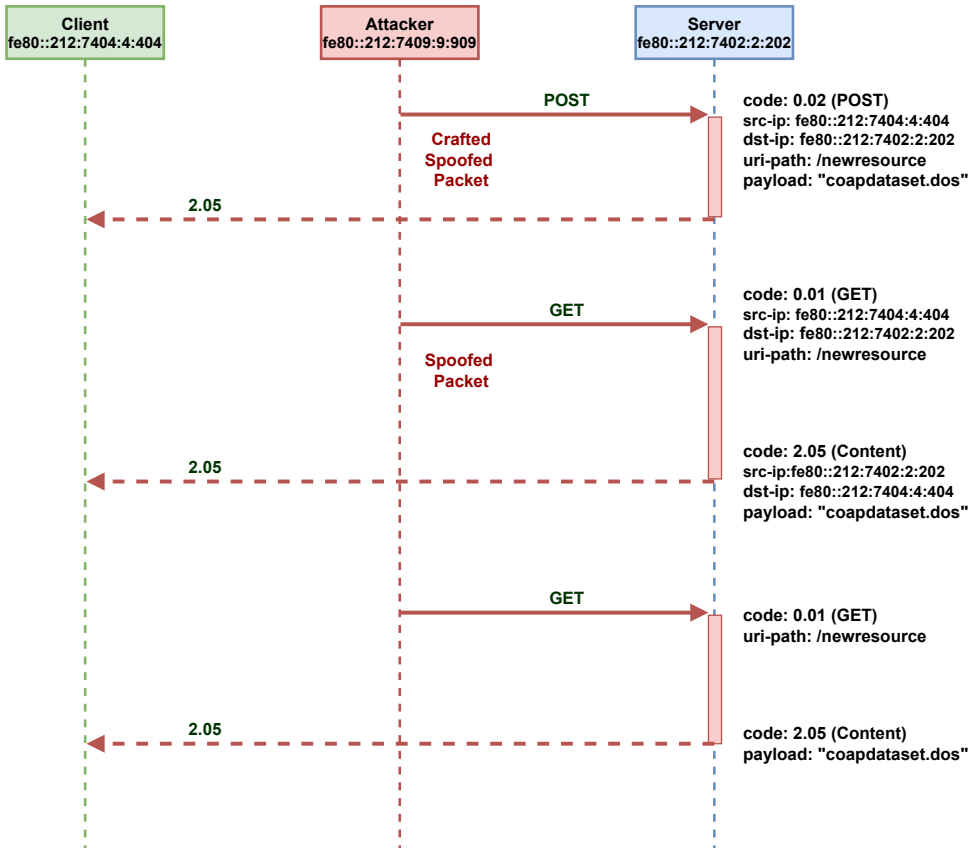

Figure 4.10 – Amplification Attack using multiple requests and customized amplification factor

4.3.3 Scenarios and Topology

In the first scenario (see figure 4.11(a)), the malicious node 5 or attacker is external to the network. Using Scapy, we configure and craft CoAP packets destined to overload the server (node 2). In the second scenario (see figure4.11(b)), the malicious node 5 is internal to the network. In both cases, the malicious node is configured to send packets that increase the amplification factor by spoofing the IP address of the benign node 4.

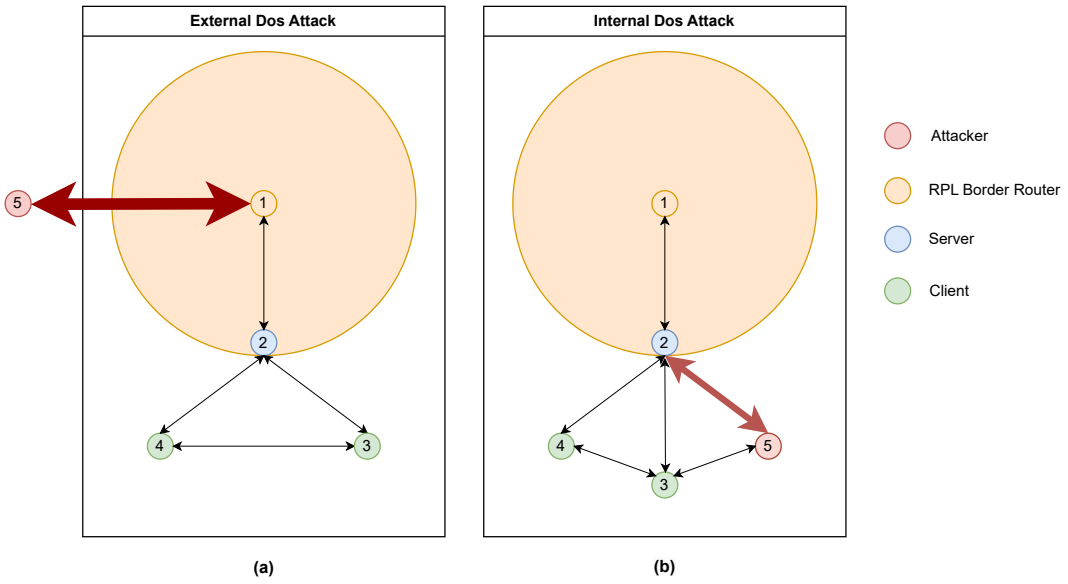

Figure 4.11 – Denial of service attack Topology

4.3.4 Simulation Parameters

Implementing the Denial of Service (DoS) attack simulation, especially in the context of CoAP amplification, several key parameters were used. These parameters dictate how the attack is performed, whether it's a singular DoS or a Distributed Denial of Service (DDoS) attack, and how the amplification factor is calculated and adjusted. The design focuses on the initiation of massive requests from single or multiple sources to overwhelm the server's resources.

Here's an example table that shows the simulation parameters:

Parameter	Value
Attack Type	DoS
Source	Single endpoint
Request/Response Size	1-1024 bytes
Amplification Factor	Varies (x)
Spoofing	Yes

Table 4.3 – Simulation Parameters for DoS Attack

For each simulation, the attack type is selected, either DoS or DDoS, determining if the attack is initiated from a single endpoint or multiple sources. The size of the requests or responses sent during the attack is chosen, and it can range from 1 to 1024 bytes, as the CoAP protocol is designed to process up to 1024 bytes per request. The amplification factor, which is the ratio of the data generated by the attacker to the actual data sent to the victim, is also varied based on the simulation requirements. For example, to increase the amplification factor, the attacker spoofs a client's IP address and sends a 40 bytes CoAP request, knowing that this request will generate a response of 160 bytes from the server towards the spoofed client. In this example, the amplification factor is 4, meaning for every byte the attacker sends, the victim receives 4 bytes. Additionally, the simulation takes into account whether IP spoofing is used in the attack, enabling the attacker to generate more requests from multiple sources, leading to a DDoS amplification attack.

4.3.5 Results

Upon initial observation, as indicated in the network trace showcased in Figure 4.12, acquired through the implementation of the topology depicted in Figure 4.11(b), it becomes evident that a request packet containing 26 bytes of application payload (CoAP) — totaling 68 bytes — results in an expanded payload of 85 bytes — 127 in total. This corresponds to a modest 3.27x amplification factor, with a focus solely on the application payload for simplification purposes. Nevertheless, if an attacker identifies a server that permits POST or PUT actions on its content, they can prepare an extensive response in advance and subsequently request it using a spoofed request. This would lead the server to transmit the sizable packet to the victim.

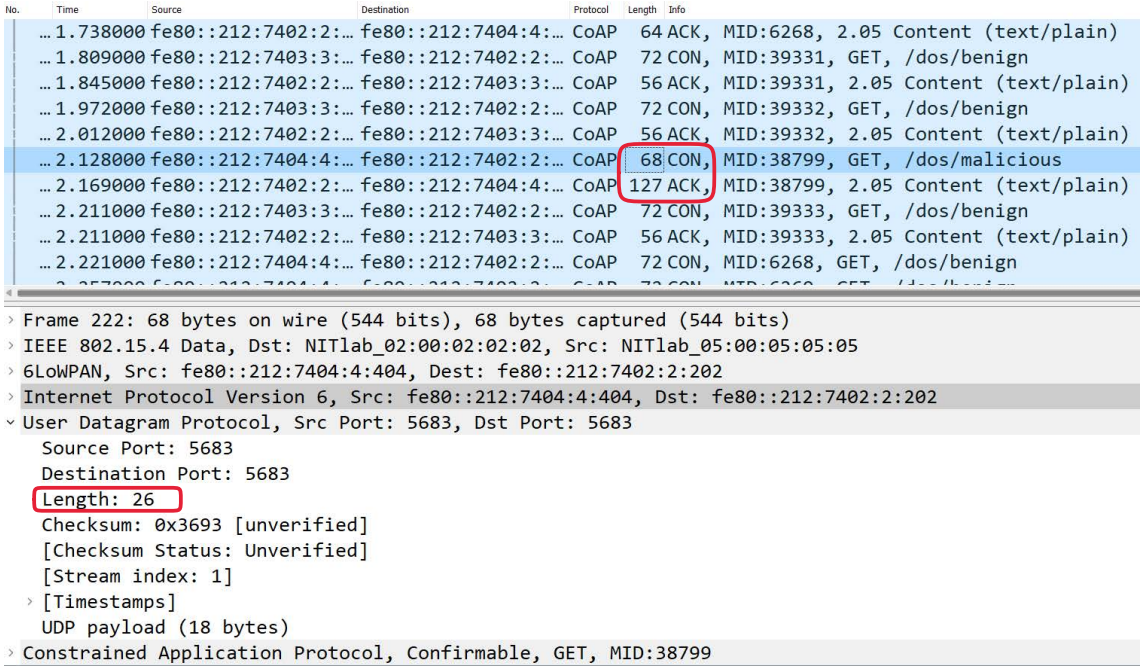

Figure 4.12 – Internal malicious node DoS attack network trace

An in-depth analysis of the protocol specification reveals that CoAP introduces a feature known as "block-wise transfer," which allows a large response to be fragmented into smaller segments. Notably, both the client and server possess the ability to determine the block size to be employed. In this context, an attacker could construct a request packet that prompts the utilization of the maximum block size. This maneuver elevates the amplification factor to 32x, a significant figure, especially given the typically constrained resources of CoAP nodes (such as those operating on limited battery power). This phenomenon is demonstrated in Figure 4.13, which displays a network trace obtained from the implementation of the topology depicted in Figure 4.11(a). Within this illustration, a malicious packet trace request incorporating 26 bytes of application payload — totaling 70 bytes — results in a payload of 272 bytes — 367 in total. This configuration leads to an amplification factor of 10.46x.

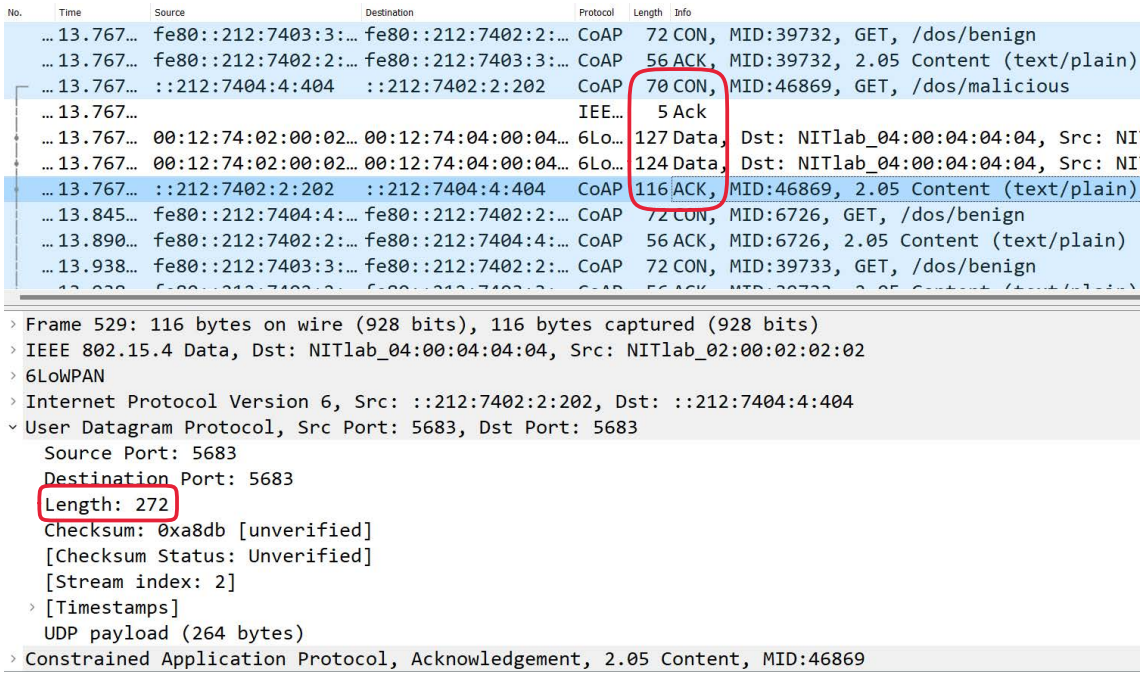

Figure 4.13 – Increasing amplification factor using block-wise transfer

By intensifying the number of GET requests, the amplification factor can be adjusted to a predetermined value as illustrated in the network trace in Figure 4.14.

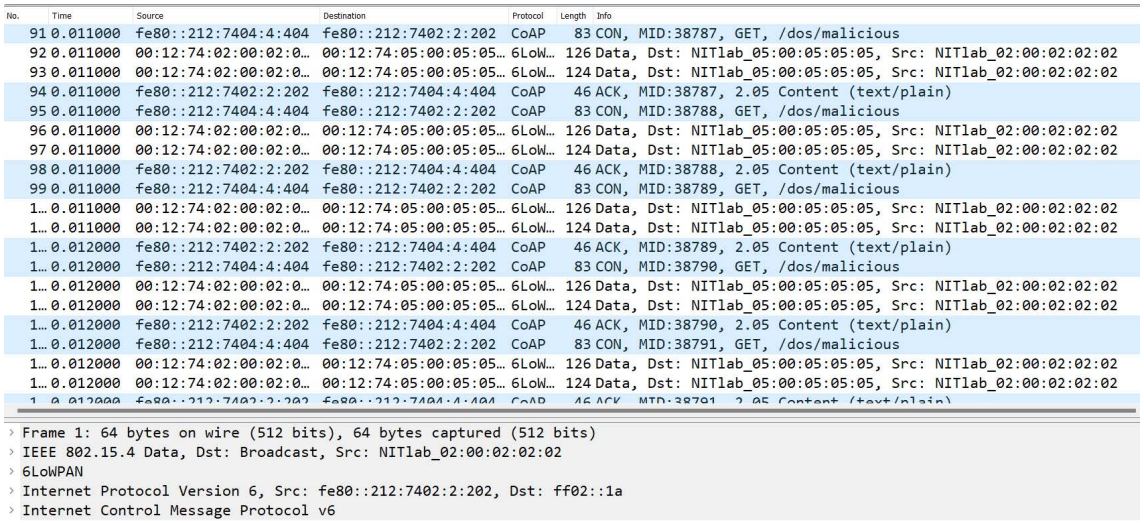

Figure 4.14 – Adjusting amplification factor to a predetermined value

Chapter 5
Analysis and Discussion

Chapter Summary

This chapter focuses on analyzing the Constrained Application Protocol (CoAP) network traffic, focusing on distinguishing between benign and malicious patterns. Through a methodological approach, the research observes typical CoAP traffic and contrasts it against controlled malicious attacks, including IP Spoofing, DoS, and Network Scanning. The findings, presented with parameters like Timestamp and Payload Size, reveal predictable patterns in benign traffic and anomalies during attacks. These insights highlight the vulnerabilities in CoAP-based IoT systems, emphasizing the need for enhanced security measures to guard against potential threats.

5.1 Goal of the Work

The principal objective of this research is to provide a detailed and structured analysis of the Constrained Application Protocol (CoAP) network traffic, encompassing both benign and malicious patterns. By understanding the distinction between these traffic types, we aim to strengthen the security and reliability of IoT systems operating on the CoAP protocol. This analysis will enable stakeholders, ranging from developers to security professionals, to devise more informed and effective security measures against potential threats.

5.2 Methodology

Our methodological approach pivots on a two-pronged examination: observing benign, regular traffic patterns and then contrasting these with malicious traffic patterns induced through controlled attacks.

5.2.1 Benign traffic creation & data gathering

Setup: A set of IoT devices operating on the CoAP protocol were interconnected within a controlled environment. These devices were programmed to emulate regular, everyday activities—such as temperature readings, light adjustments, and other sensor-based tasks.

Data Collection: Traffic was captured over a period of three weeks using packet sniffing tools like Wireshark. Efforts were made to ensure a diverse range of actions and requests were recorded to get a comprehensive representation of benign CoAP traffic.

5.2.2 Malicious traffic creation & data gathering

Setup: Employing the attack vectors detailed in Chapter 4, we simulated malicious activities against our CoAP network.

CoAP Network Scan Attack: Requests were sent to a range of IP addresses and responses analysed in order to identify the devices.

IP Spoofing Attack: Unauthorized requests were sent with falsified IP addresses to mislead recipient devices.

DoS Attack: The CoAP servers were overwhelmed by exploiting the risk of amplification vulnerability in the CoAP Protocol.

Data Collection: Similar to the benign traffic, malicious traffic was also captured over a concentrated time frame. The focus was to document the initial signs of the attack, its peak impact, and the post-attack behavior of the network.

5.3 Presentation of results

Our results aim to demystify the complex patterns of network traffic, categorizing them into discernible patterns, and enabling a clearer understanding of how regular and malicious activities manifest on a CoAP network.

5.3.1 Description of dataset parameters

Our dataset is structured into the following parameters:

- **Timestamp**: The precise date and time when each packet was transmitted or received.
- **Source IP & Destination IP**: Addresses indicating the sender and receiver of packets.
- **Payload Size**: The size of the actual content being transmitted, excluding header data.
- **Message Type**: Categorization of messages into requests or responses.
- **CoAP Methods**: Specific CoAP methods being utilized, e.g., GET, POST, PUT, DELETE.
- **Status Codes**: Responses from servers, e.g., 2.05 (Content) or 4.04 (Not Found).
- **URI-Path**: The path or resource being accessed or modified.
- **Payload Data**: Actual data being sent or received, especially useful in observing malicious content.
- **Malicious Indicator**: A binary flag indicating if the traffic is benign (0) or malicious (1), derived from our controlled attacks.

This dataset, thus, not only provides a snapshot of the network's operational behavior but also highlights the distinguishing characteristics between benign and malicious activities, offering invaluable insights for future security endeavors.

5.3.2 Normal Scenario traffic analysis

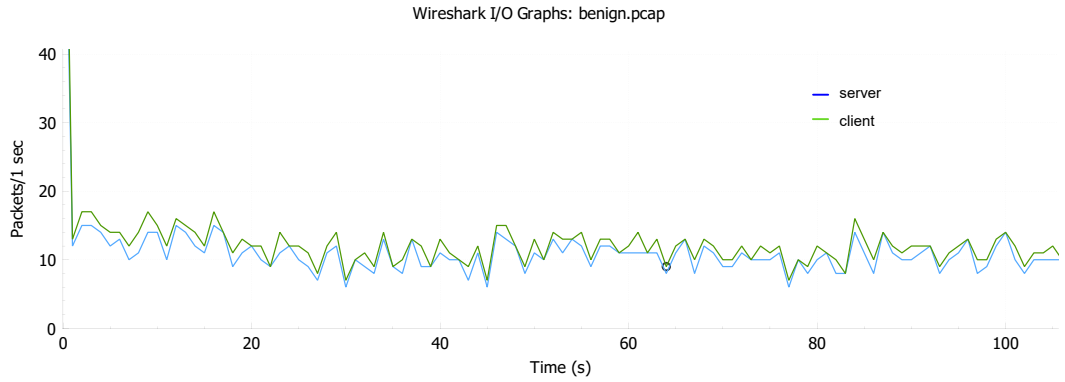

Figure 5.1 – Analysis of benign packet traces

Figure 5.1 depicts the activity diagram of a normal scenario. The green chart represents the activity of benign clients whereas the blue chart represents the server's activity. We observe a similar activity level between the client and the server.

5.3.3 Network Scan Attack traffic analysis

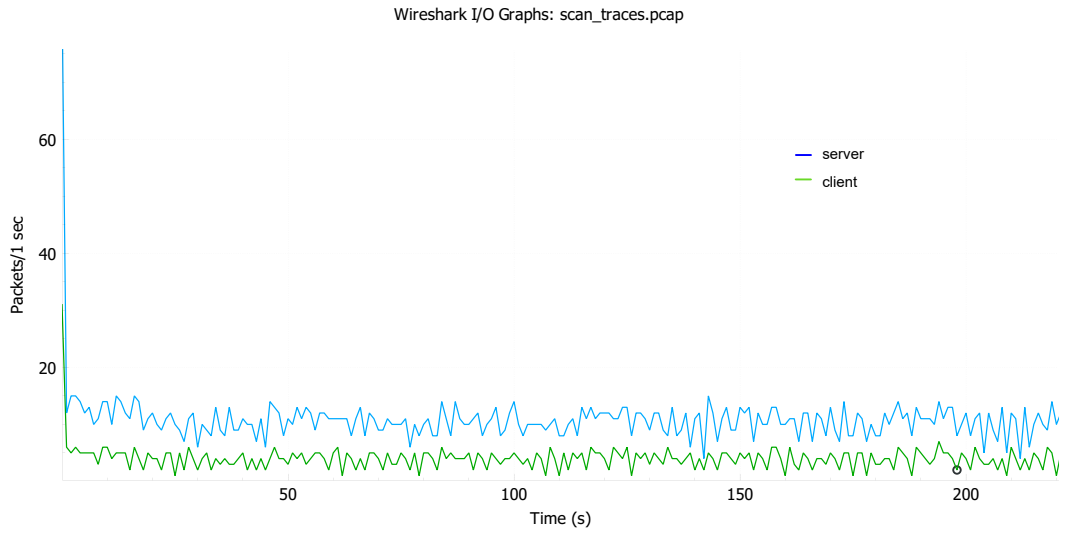

Figure 5.2 – Analysis of scan attack results

In figure 5.2, there is a deviation between the activity level of the client and that of the server as compared to the normal scenario in section 5.3.2. This difference is due to the scan attack being executed, leading to a conspicuous increase in the server's network activity as more packets were being handled per second by the server. This additional traffic not only introduced an overhead on the network but also showcased how external threats can have a more direct impact despite their unprivileged position outside the network.

The Cooja simulator provided valuable performance metrics, crucial for analyzing the network's behavior under a scan attack. These metrics included request-response latency, the number of successful resource discoveries, and the ratio of successful scans to failed or non-responsive scans. Such metrics offer deep insights into the network's resilience against

scanning attacks and can help in devising countermeasures.

The auto-generated random seed by Cooja during the initialization phase played a pivotal role in shaping the nodes' behavior. Packet transmission times showed variability in different simulation runs, emphasizing the unpredictable nature of real-world networks and the need for adaptive defensive mechanisms.

In summary, the results obtained from the network scan attack simulation provide a comprehensive understanding of potential vulnerabilities and the network's response dynamics. The key takeaways from this section will significantly aid in designing more robust and secure CoAP-based systems.

5.3.4 IP Spoofing Attack traffic analysis

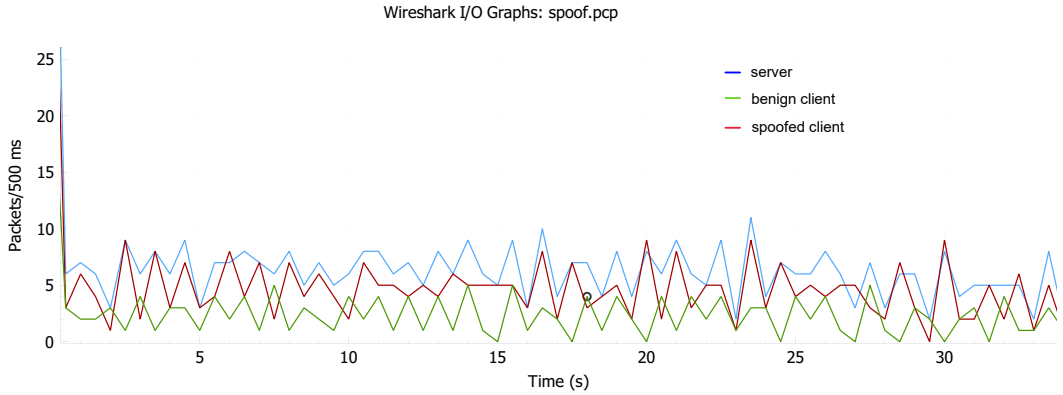

Figure 5.3 – Analysis of IP spoofing attack results

The server node experimented an unusual influx of CON messages. This was due to the malicious node sending these messages by spoofing the benign client's identity. Not only did this flood the server with unnecessary traffic, but it also highlighted how easily an attacker could inundate network resources. In Figure 5.3, the activity of the spoofed client is above that of the benign client, reaching the server's activity level at certain points. This indicates that packets coming from the spoofed IP are above the expected level.

In the scenario where the spoofed client's fake CON message reached the server before the benign client's legitimate request, the benign client remained oblivious to the attack. This kind of silent disruption can be particularly damaging as it operates covertly.

Both external and internal attackers effectively deceived nodes within the network. This implies that, without proper security mechanisms, even internal nodes can be just as vulnerable to spoofing attacks as external nodes.

In conclusion, the results derived from the IP spoofing attack simulations underscore the vulnerabilities present in network environments. They highlight the need for robust security solutions, especially when dealing with protocols like CoAP. Furthermore, these findings can serve as a foundation for developing more advanced mitigation strategies against IP spoofing attacks.

5.3.5 DoS Attack

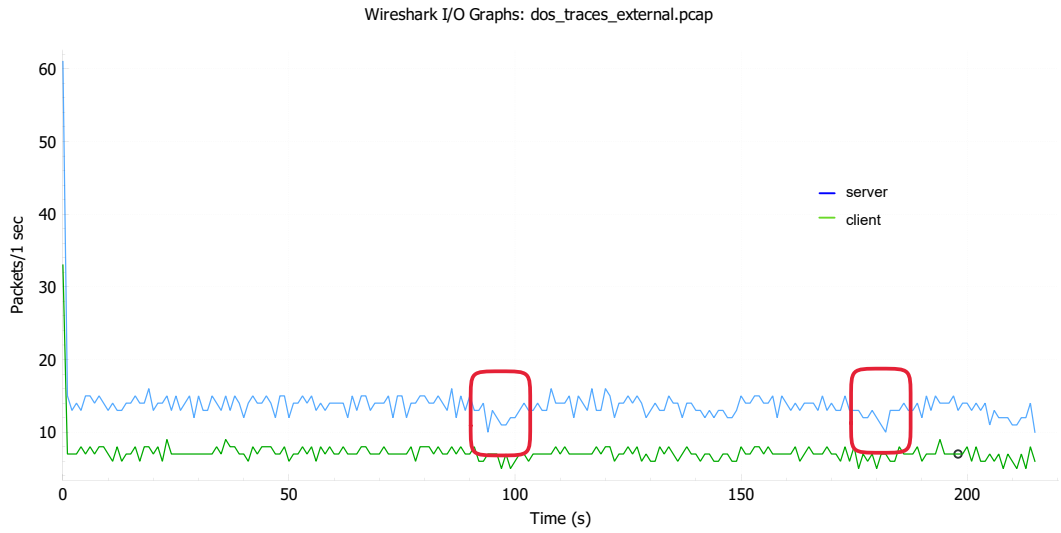

Figure 5.4 – Analysis of DoS attack results

Following the simulation of the Denial of Service (DoS) attack, several notable results emerged, shedding light on the real-world implications of such attacks. This section elucidates the outcomes of the amplification attacks targeting CoAP and how they affect server performance and service availability.

CoAP Amplification Outcomes

Service Downtime: As anticipated, the CoAP server was overwhelmed by the amplified requests, leading to a significant service downtime. This is illustrated in figure 5.4 by a sharp decrease in the server's activity at two points. Legitimate clients attempting to access the service during this period experienced substantial delays or total denial of access.

Data Overload: The CoAP amplification attacks led to the server processing an extensive amount of data, far more than its regular operational limits. This increase in data, especially when the message size surpasses the standard 1024 bytes, adds significant strain on server resources.

Amplification Factor: As per the simulation, an amplification factor was achieved where the server responded with data that was 10.46x times larger (considering only the payload) than the request. This phenomenon elucidated the potential for attackers to magnify the effects of their initial malicious payloads.

Successful IP Spoofing: The simulation revealed that the attack was especially potent when the victim's CoAP client IP was effectively spoofed. This underscores the combined threat of CoAP amplification attacks in tandem with IP spoofing.

Manipulation of GET Requests: An interesting outcome was that the attacker could escalate the amplification factor by increasing the number of GET requests aimed at the victim. This allows for a more tailored attack depending on the attacker's desired impact

and the available resources.

DDoS Amplification Attack Insights

Multi-source Traffic: When the attack was initiated in conjunction with IP address spoofing across multiple sources, the influx of traffic to the CoAP server was exponentially increased. This demonstrated how easily a DoS attack could escalate to a DDoS amplification attack.

The results derived from the simulation provide an alarming glimpse into the potential damage and disruption that CoAP amplification attacks can inflict. These findings emphasize the urgency to develop and implement robust security measures to counteract such threats, especially given the increasing reliance on CoAP in modern IoT environments.

5.4 Discussion of Results

Upon close examination of our dataset, several critical observations emerged:

Distinct Patterns in Benign Traffic: Benign traffic exhibited a regular, predictable pattern, often characterized by consistent intervals between requests, standard payload sizes, and common CoAP methods like GET and POST. The consistency and predictability were a testament to the operational stability of the CoAP IoT devices under normal circumstances.

Sudden Surge during Malicious Activities: As expected, during the onslaught of attacks, especially during the DoS attack, there was a noticeable surge in traffic volume. This anomaly was further magnified by uncommon CoAP methods and an array of failed status codes.

IP Inconsistencies: During the IP spoofing attack, several source IPs that weren't part of the original network made requests. Such anomalies are clear indicators of potential malicious activities, especially if the spoofed IPs repeatedly make unusual or harmful requests.

Resource Overload: The DoS attack notably stressed resources, leading to a high number of dropped requests and increased response times. This emphasized the CoAP network's vulnerability to flooding attacks.

Network Reconnaissance Signs: The CoAP network scan attack revealed repeated attempts to access various resources, indicating an intruder's intent to discover potential vulnerabilities.

These observations reinforce the need for proactive security measures in IoT networks, especially those operating on lightweight protocols like CoAP. Such protocols, while efficient, can become easy targets for attackers, underscoring the urgency to fortify them against malicious threats.

5.5 Summary of Benign and Malicious Packets distribution

Table 5.1 offers a concise overview of the packet distribution between benign and malicious traffic for each attack type. We quantified the proportion of packets originating from benign sources as opposed to those generated by specific malicious activities. The percentages showcased in the table reveal the impact of different attack vectors on the CoAP network traffic. Notably, during the Network Scan attack, benign packets accounted for 80% of the total, while malicious packets constituted 20%. Similarly, in the case of IP Spoofing, benign packets comprised 70% of the total, with 30% being malicious. Finally, the DoS Attack saw a distribution of 60% benign packets and 40% malicious packets. These percentages offer valuable insights into the varying degrees of impact of different attacks on the network.

Attack Type	**Benign Packets** (%)	**Malicious Packets** (%)
Network Scan	1200 (80%)	300 (20%)
IP Spoofing	1260 (70%)	540 (30%)
DoS Attack	1500 (60%)	1000 (40%)
Total	3960 (68.28%)	1840 (31.72%)

Table 5.1 – Summary of Benign and Malicious Packets for Each Attack Type

Chapter 6
Conclusion and Future Work

Chapter Summary

Chapter 6 concludes our study on the CoAP protocol by detailing its operational strengths and vulnerabilities. Through simulations of both benign and malicious traffic, our research identifies distinct patterns and emphasizes the vital balance between efficiency and security in IoT devices. While this study offers significant findings, it acknowledges its limitations, such as the controlled environment and the focus on only three attack vectors. Suggestions for future work include broadening the attack spectrum, assembling a larger dataset, integrating machine learning for threat detection, and testing in real-world environments. This chapter envisions a future where IoT devices function securely, ensuring that their benefits aren't offset by security risks.

6.1 Conclusions

Our study ventured deep into the operational intricacies of the CoAP protocol, revealing its strengths and potential vulnerabilities. We successfully simulated benign and malicious traffic, highlighting the discerning patterns between them. Our research not only sheds light on the immediate threats IoT devices face but also underlines the importance of proactive security measures.

The importance of lightweight protocols in IoT cannot be undermined. Yet, as our research reveals, it is of paramount importance that the balance between efficiency and security is maintained. By understanding the vulnerabilities of such protocols, especially CoAP, stakeholders can make informed decisions and ensure that IoT's promise of a more connected and efficient world does not become its own downfall.

6.2 Limitations and Future Work

While our study offers valuable insights into CoAP network vulnerabilities, it is not without limitations. The controlled environment we employed, albeit efficient for our objectives, might not completely replicate real-world scenarios where numerous external factors come into play. Our research mainly emphasized three specific attack vectors, leaving the possibility of other unidentified vulnerabilities.

As a trajectory for future work:

Expand the Attack Spectrum: While we focused on three primary attacks, there are myriad other potential vulnerabilities that can be explored.

Larger Dataset: Incorporating more diverse devices and broader network configurations can help create a more comprehensive dataset.

Machine Learning Integration: With the dataset at hand, machine learning algorithms can be developed to automatically detect and possibly counteract malicious activities.

Real-world Testing: Simulating these attacks in a more dynamic environment, akin to real-world conditions, can offer a more authentic understanding of potential vulnerabilities.

We envisage a future where IoT devices, powered by protocols like CoAP, can operate seamlessly without the looming threat of security breaches, ensuring that the convenience they bring does not come at the cost of compromised safety.

Bibliography

[1] Amplification hell: Revisiting network protocols for ddos abuse - ndss symposium, https://www.ndss-symposium.org/ndss2014/programme/amplification-hell-revisiting-network-protocols-ddos-abuse/ [Cited on page 17.]

[2] Internet of things (iot) connected devices installed base worldwide from 2015 to 2025, https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ [Cited on page 1.]

[3] mkovatsc/copper: Copper (cu) coap user-agent (javascript implementation) (2016), https://github.com/mkovatsc/Copper [Cited on page 18.]

[4] Http response status codes - http | mdn (NaN), https://developer.mozilla.org/en-US/docs/Web/HTTP/Status [Cited on page 9.]

[5] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Iot security: Review, blockchain solutions, and open challenges. IEEE Internet of Things Journal (2018) [Cited on page 18.]

[6] Aseem: Introduction To CoAP Protocol And Security. https://payatu.com/masterclass/iot-security-part-11-introduction-to-coap-protocol-and-security/ (2020), [Online; accessed 26-February-2023] [Cited on pages II, 6, 8, 9, 10, 11, and 12.]

[7] Azad, A., Khan, M.A., Hossain, A., Song, H.: A lightweight approach to detect ddos attacks on coap-based internet of things devices. Sensors (2021) [Cited on pages 16 and 19.]

[8] Egli, P.: Mqtt - message queueing telemetry transport introduction to mqtt, a protocol for m2m and iot applications (01 2017) [Cited on page 13.]

[9] Ghazanfar, S., Hussain, F., Rehman, A.U., Fayyaz, U.U., Shahzad, F., Shah, G.A.: Iot-flock: An open-source framework for iot traffic generation. In: Proceedings of the 2020 International Conference on Emerging Trends in Smart Technologies (ICETST). pp. 26–27. Karachi, Pakistan, pp. 1–6 (March 2020) [Cited on page 18.]

[10] Haines, J., Rossey, L., Lippmann, R., Cunningham, R.: Extending the darpa off-line intrusion detection evaluations. In: Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01. vol. 1, pp. 35–45 vol.1 (2001) [Cited on page 17.]

[11] Hussain, F., Abbas, S.G., Shah, G.A., Pires, I.M., Fayyaz, U.U., Shahzad, F., Garcia, N.M., Zdravevski, E.: A framework for malicious traffic detection in iot healthcare environment. Sensors 21(9) (2021) [Cited on page 18.]

[12] Ioulianou, P.: Protecting iot networks against routing attacks (December 2021) [Cited on pages 20 and 21.]

[13] Koroniotis, N., Moustafa, N., Sitnikova, E., Turnbull, B.: Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. Future Generation Computer Systems 100, 779–796 (2019) [Cited on page 18.]

[14] Krimmling, J., Peter, S.: Integration and evaluation of intrusion detection for coap in smart city applications (10 2014) [Cited on page 17.]

[15] Lee, W., Stolfo, S.J.: A framework for constructing features and models for intrusion detection systems. ACM Trans. Inf. Syst. Secur. 3(4), 227–261 (nov 2000) [Cited on page 18.]

[16] Levis, P., Lee, N.: Tossim: A simulator for tinyos networks. UC Berkeley, September 24 (2003) [Cited on page 20.]

[17] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., AlecWoo, D.G., Hill, J., MattWelsh, E.B., et al.: Tinyos: An operating system for sensor networks. In Ambient intelligence, pages 115–148. Springer (2005) [Cited on page 20.]

[18] Lippmann, R., Haines, J.W., Fried, D.J., Korba, J., Das, K.: Analysis and results of the 1999 darpa off-line intrusion detection evaluation. In: Debar, H., Mé, L., Wu, S.F. (eds.) Recent Advances in Intrusion Detection. pp. 162–182. Springer Berlin Heidelberg, Berlin, Heidelberg (2000) [Cited on page 17.]

[19] Maggi, F., Vosseler, R., Quarta, D.: The fragility of industrial iot's data backbone: Security and privacy issues in mqtt and coap protocols. Tech. rep., Trend Micro, Inc. [Cited on page 16.]

[20] Mathews, J., Chatterjee, P., Banik, S.: CoAP-DoS: An IoT network intrusion data set. In: 2022 6th International Conference on Cryptography, Security and Privacy (CSP). IEEE (jan 2022) [Cited on pages 16 and 19.]

[21] Mattsson, J.P., Selander, G., Amsüss, C.: Amplification Attacks Using the Constrained Application Protocol (CoAP). Internet-Draft draft-irtf-t2trg-amplification-attacks-02, Internet Engineering Task Force (Apr 2023), work in Progress [Cited on page 31.]

[22] Musaddiq, A., Zikria, Y.B., Kim, S.W., et al.: Routing protocol for low-power and lossy networks for heterogeneous traffic network. EURASIP Journal on Wireless Communications and Networking 2020(1), 1–23 (2020) [Cited on page 23.]

[23] Oikonomou, G., Duquennoy, S., Elsts, A., Eriksson, J., Tanaka, Y., Tsiftes, N.: The Contiki-NG open source operating system for next generation IoT devices. SoftwareX 18, 101089 (2022) [Cited on pages 20 and 24.]

[24] Osterlind, F., et al.: Cross-level sensor network simulation with cooja 641 (November 2006) [Cited on page 20.]

[25] Rathore, S., Park, J.H.: Semi-supervised learning based distributed attack detection framework for iot. Applied Soft Computing 72, 79–89 (2018) [Cited on page 18.]

[26] Ray, D., Bhale, P., Biswas, S., Nandi, S., Mitra, P.: Daiss: Design of an attacker identification scheme in coap request/response spoofing. pp. 941–946 (12 2021) [Cited on page 26.]

[27] Romdhani, M., Al-Fuqaha, A.M., Guizani, M.: An anomaly detection approach for the constrained application protocol in the internet of things. IEEE Communications Letters (2019) [Cited on page 19.]

[28] Selander, G., Mattsson, J.P., Palombini, F., Seitz, L.: Object Security for Constrained RESTful Environments (OSCORE). RFC 8613 (Jul 2019), `https://www.rfc-editor.org/info/rfc8613` [Cited on page 15.]

[29] Shelby, Z.: Constrained RESTful Environments (CoRE) Link Format. RFC 6690 (Aug 2012), `https://www.rfc-editor.org/info/rfc6690` [Cited on page 14.]

[30] Shelby, Z., Hartke, K., Bormann, C.: The Constrained Application Protocol (CoAP). RFC 7252 (Jun 2014), `https://www.rfc-editor.org/info/rfc7252` [Cited on pages III, 7, 14, 15, 16, 17, and 32.]

[31] Tsiftes, N., Eriksson, J., Dunkels, A.: Low-power wireless ipv6 routing with contikirpl. In Proc. 9, 406–407 (2010) [Cited on page 24.]

[32] Ullah, I.: A technique for generating a botnet dataset for anomalous activity detection in iot networks (12 2020) [Cited on page 18.]

[33] Vishwakarma, R., Jain, A.: A honeypot with machine learning based detection framework for defending iot based botnet ddos attacks. pp. 1019–1024 (04 2019) [Cited on page 17.]

[34] Wikipedia contributors: Star network — Wikipedia, the free encyclopedia (2023), `https://en.wikipedia.org/w/index.php?title=Star_network&oldid=1160476260`, [Online; accessed 22-July-2023] [Cited on page 19.]

[35] Wu, J., Dong, J.: A simple service discovery and configuration protocol for embedded devices 37 (11 2006) [Cited on page 13.]

[36] Ünal Çavuşoğlu, Mehmet Ali Ebleme, C.B.K.K.: Coap and its performance evaluation. Sakarya University Journal of Science, 24(1), 78–85 (2020) [Cited on page 1.]

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**