Technical University of Crete

School of Electrical and Computer Engineering

A Moving Target Defense System for IoT environments

Master Thesis by

Thomas Kyriakakis

Thesis Committee

Professor Sotiris Ioannidis (supervisor)
Professor Eftychios Koutroulis
Dr. Evangelia Papadogiannaki

Chania, May 2025

Abstract

As cyber attacks become more sophisticated and frequent, traditional security solutions are proving to be insufficient to protect against them. Moving Target Defense (MTD) is a promising approach that involves constantly changing the system's configuration, making it more difficult for attackers to exploit known vulnerabilities. By making the system dynamic and unpredictable, MTD can significantly improve its resilience to a range of cyber threats.

In this thesis, we propose MTDIoT, a moving target defense solution designed specifically for IoT applications. IoT devices are often resource-constrained and may not have the processing power or memory to support complex security solutions. MTDIoT has minimal compute requirements and can be easily integrated with other security components, making it an ideal choice for IoT deployments.

MTDIoT works by changing the network configuration dynamically over time or in response to specific events. This makes it harder for attackers to identify and target specific devices and services. Additionally, MTDIoT has the ability to exclude compromised nodes from updated network configurations, isolating them from the rest of the network.

We evaluate the performance of MTDIoT in a simulated network environment. Our results demonstrate the effectiveness of MTDIoT in reducing the success rate of attacks on IoT applications. By constantly changing the network configuration, MTDIoT makes it more difficult for attackers to identify and exploit vulnerabilities, reducing the overall risk of cyber attacks.

MTD has the potential to improve the security of a wide range of systems and applications, including cloud computing, critical infrastructure and networks. While MTD may not provide a complete solution to all cyber threats, it can significantly enhance the resilience of systems and reduce the attack surface available to attackers. As such, MTD is likely to become an increasingly important component of modern cybersecurity strategies.

Περίληψη

Καθώς οι επιθέσεις στον κυβερνοχώρο γίνονται πιο εξελιγμένες και συχνές, οι παραδοσιακές λύσεις ασφάλειας αποδεικνύονται ανεπαρκείς για την προστασία από αυτές. Το Moving Target Defense (MTD) είναι μια πολλά υποσχόμενη προσέγγιση που περιλαμβάνει συνεχή αλλαγή της διαμόρφωσης του συστήματος, καθιστώντας πιο δύσκολο για τους επιτιθέμενους να εκμεταλλευτούν γνωστά τρωτά σημεία. Κάνοντας το σύστημα δυναμικό και απρόβλεπτο, το MTD μπορεί να βελτιώσει σημαντικά την ανθεκτικότητά του σε μια σειρά από απειλές στον κυβερνοχώρο.

Σε αυτή την μεταπτυχιακή διατριβή, προτείνουμε το MTDIoT, μια λύση άμυνας κινούμενου στόχου σχεδιασμένη ειδικά για εφαρμογές IoT. Οι συσκευές IoT είναι συχνά περιορισμένες σε πόρους και ενδέχεται να μην έχουν την ισχύ επεξεργασίας ή τη μνήμη για να υποστηρίξουν πολύπλοκες λύσεις ασφάλειας. Το MTDIoT έχει ελάχιστες υπολογιστικές απαιτήσεις και μπορεί εύκολα να ενσωματωθεί με άλλα στοιχεία ασφαλείας, καθιστώντας το ιδανική επιλογή για αναπτύξεις IoT.

Το MTDIoT λειτουργεί αλλάζοντας τη διαμόρφωση του δικτύου δυναμικά με την πάροδο του χρόνου ή ως απόκριση σε συγκεκριμένα συμβάντα. Αυτό καθιστά πιο δύσκολο για τους εισβολείς να αναγνωρίσουν και να στοχεύσουν συγκεκριμένες συσκευές και υπηρεσίες. Επιπλέον, το MTDIoT έχει τη δυνατότητα να αποκλείει παραβιασμένους κόμβους από ενημερωμένες διαμορφώσεις δικτύου, απομονώνοντάς τους από το υπόλοιπο δίκτυο.

Αξιολογούμε την απόδοση του MTDIoT σε περιβάλλον προσομοίωσης δικτύου, χρησιμοποιώντας μια ποικιλία σεναρίων επίθεσης. Τα αποτελέσματά μας καταδεικνύουν την αποτελεσματικότητα του MTDIoT στη μείωση του ποσοστού επιτυχίας των επιθέσεων σε εφαρμογές IoT. Αλλάζοντας συνεχώς τη διαμόρφωση του δικτύου, το MTDIoT καθιστά πιο δύσκολο για τους εισβολείς να εντοπίσουν και να εκμεταλλευτούν τα τρωτά σημεία, μειώνοντας τον συνολικό κίνδυνο επιθέσεων στον κυβερνοχώρο.

Το MTD έχει τη δυνατότητα να βελτιώσει την ασφάλεια ενός ευρέος φάσματος συστημάτων και εφαρμογών, συμπεριλαμβανομένου του υπολογιστικού νέφους, των κρίσιμων υποδομών και των δικτύων. Αν και το MTD μπορεί να μην παρέχει ολοκλη-

ρωμένη λύση σε όλες τις απειλές στον κυβερνοχώρο, μπορεί να βελτιώσει σημαντικά την ανθεκτικότητα των συστημάτων και να μειώσει την επιφάνεια επίθεσης που είναι διαθέσιμη στους επιτιθέμενους. Ως εκ τούτου, το MTD είναι πιθανό να γίνει ένα ολοένα και πιο σημαντικό συστατικό των σύγχρονων στρατηγικών ασφάλειας στον κυβερνοχώρο.

Acknowledgements

I would like to express my deepest gratitude to the members of my thesis committee for their invaluable guidance and support throughout this journey. I am especially thankful to Professor Sotiris Ioannidis for his expert advice and encouragement, which greatly contributed to the quality and completion of this work.

I also sincerely thank Dr. Evangelia Papadogiannaki and Prof Eftyxios Koutroulis for their thorough and insightful review of my thesis, as well as for kindly agreeing to be members of the committee.

A special and heartfelt thank you goes to Andreas Brokalakis, whose unwavering assistance and support were instrumental across all sections of this thesis. His presence throughout the entire process was a great source of motivation.

I am also grateful to Babis Savvakos and Vasilios Amourgianos for their continuous technical support, which was essential for the successful realization of this project.

I would like to thank all my colleagues, including those who are no longer with the team, for the great team spirit we shared over these past four years. Their collaboration and camaraderie made this journey truly memorable.

Finally, I extend my deepest appreciation to my family and friends for their unwavering support and encouragement, which sustained me throughout this entire endeavor.

Contents

List of Figures

List of Tables

Listings

Table 1: List of Abbreviations

Abbreviation	**Full Name**
6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
AI/ML	Artificial Intelligence/Machine Learning
AMQP	Advanced Message Queuing Protocol
AMI	Advanced Metering Infrastructure
CA	Certificate Authority
DoS	Denial of Service
DTLS	Datagram Transport Layer Security
ECC	Elliptic Curve Cryptography
EDR	Endpoint Detection and Response
HSM	Hardware Security Module
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
IR	Incident Response
MANET	Mobile Ad hoc Network
MITM	Man-In-The-Middle
MT6D	Moving Target IPv6 Defense
MTD	Moving Target Defense
OTA	Over-The-Air
PKI	Public Key Infrastructure
RPi	Raspberry Pi
RPL	Routing Protocol for Low-Power and Lossy Networks
RSSI	Received Signal Strength Indication
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TSN	Time-Sensitive Networking
UDP	User Datagram Protocol
VPN	Virtual Private Network

Chapter 1

Introduction

1.1 General Overview

In the rapidly evolving landscape of cybersecurity, traditional static defense mechanisms are proving to be inadequate in safeguarding sensitive digital assets from sophisticated and persistent adversaries. Cyber-attacks are becoming increasingly advanced and frequent, necessitating a paradigm shift in defensive strategies. One innovative approach that has gained significant attention is the concept of Moving Target Defense (MTD). MTD involves dynamically altering a system's environment, configurations, or other relevant parameters to increase its resilience against cyber threats. By constantly changing attack surfaces, attack paths, or system characteristics, MTD significantly raises the cost and complexity for adversaries seeking to exploit vulnerabilities.

Traditional cybersecurity relies on the assumption that a static, well-defended system can adequately repel attacks. However, this assumption is increasingly unrealistic as attackers employ sophisticated tactics, techniques, and procedures to identify and exploit vulnerabilities within these static systems. The dynamic and evolving nature of MTD aligns with the evolving tactics of cyber adversaries. Rather than relying on a fixed set of defenses, MTD operates on the principle that the best defense is a moving target, making it challenging for attackers to premeditate their attacks effectively.

This thesis focuses on the application of MTD in Internet of Things (IoT) environments, where the need for robust yet lightweight security solutions is particularly acute. IoT devices, often deployed in resource-constrained settings such as agricultural fields or industrial facilities, are highly vulnerable to cyberattacks due to their limited computational power, memory, and often exposed physical locations. To address these challenges, we propose MTDIoT, a Moving Target Defense framework specifically designed for IoT applications. MTDIoT dynamically modifies network configurations over time or in response to specific events, making it harder for attackers to identify and exploit vulnerabilities. Additionally, MTDIoT can isolate compromised nodes from the network, further enhancing its resilience. In our work, we assume that IoT networks are already operational and that attackers may gain physical access to devices, necessitating a security approach that protects the network from within. We evaluate MTDIoT in a simulated network environment, demonstrating its effectiveness in reducing the success rate of attacks on IoT applications. By leveraging the principles of MTD, MTDIoT not only enhances the security of IoT deployments but also provides a scalable and adaptable solution for other domains, such as cloud computing, critical infrastructure, and military networks.

This introduction sets the stage for a comprehensive exploration of MTD and its application in IoT environments. In the subsequent sections, we will delve into the design, implementation, and evaluation of MTDIoT, highlighting its advantages, challenges, and potential for real-world deployment. Understanding and leveraging MTD is essential in effectively combating modern cyber threats and building resilient systems capable of adapting to an ever-changing threat landscape.

1.2 Publication Contributing to this Thesis

The publication "A Moving Target Defense Security Solution for IoT Applications," [25] authored by me and S. Ioannidis, forms a foundational component of this thesis. We presented it at the 19th International Conference on the Design of Reliable Communication Networks (DRCN) in Vilanova i la Geltru, Spain. This publication provided the theoretical framework and methodologies essential for

developing and evaluating the security framework analyzed in this thesis.

1.3 Outline

The rest of this thesis is organized as follows. Chapter 2 presents the background on the basic terms of Moving Target Defenses. Furthermore, we explain more about IoT environments. In Chapter 3, we present the basic Design Requirements and the general MTDIoT architecture. In Chapter 4, we get in depth on the implementation flow by mentioning how our system was implemented, explaining the client, the server, and the integration details. Also, we provide some details about the integration with other tools and the Trust Broker details. In Chapter 5, we describe how our solution was evaluated in local testbed and some use cases inside the IntellIoT project. Next in the Chapter 6 we will present the system performance in depth. Finally, Chapter 7 presents the conclusions of our work and we analyze our future work.

Chapter 2

Background and Related Work

In this section, the fundamentals behind the concept of Moving Target Defenses are presented, along with the basic definitions, design principles and their taxonomy. We focus on their use in IoT applications and highlight the related works reported in the literature.

2.1 Detailed Examination of Existing Cybersecurity Approaches and Their Limitations

As cyber threats continue to evolve, traditional cybersecurity approaches have faced increasing scrutiny. While these methods have been foundational in protecting digital assets, their limitations have become more apparent in the face of sophisticated and persistent attacks.

The foundations of all cybersecurity defense solutions are the mechanisms to detect suspicious and/or malicious activity. Typically these detection mechanisms fall into three categories: signature-based detection, heuristic analysis and anomaly-based detection.

Signature-based detection relies on known patterns or signatures of malicious activities to identify threats. Antivirus programs and intrusion detection systems (IDS) commonly use this approach. While effective against known threats, signature-

based detection struggles with new, unknown, or polymorphic malware, which can easily evade detection by altering their signatures [14].

Heuristic analysis improves upon signature-based detection by identifying potentially malicious behavior based on certain rules or patterns. This method can detect novel threats but often results in a high number of false positives, which can overwhelm security teams and lead to alert fatigue [13].

Anomaly-based detection systems establish a baseline of normal behavior and flag deviations as potential threats. Although this approach can detect unknown attacks, it requires extensive training data and continuous tuning to maintain accuracy. Moreover, sophisticated attackers can adapt their tactics to mimic normal behavior, thereby evading detection [40].

No matter which threat detection mechanism is employed (or which combination of such solutions is put in place), endpoint security solutions, such as antivirus software and endpoint detection and response (EDR) systems, focus on protecting individual devices. These solutions often lack a holistic view of the network and can be resource-intensive, making them unsuitable for highly interconnected and resource-constrained environments like modern IoT deployments [29].

Firewalls and network segmentation are critical components of network security, controlling traffic flow and isolating sensitive areas of the network. However, these methods are static and can be bypassed by attackers who gain insider access or exploit vulnerabilities within the network [36].

Traditional cybersecurity approaches tend to be similarly static and reactive. They are configured to operate with one or more of the aforementioned detection mechanisms and respond with mitigation actions only once a threat has been identified. As such, they lack the adaptability needed to counter dynamic and evolving threats [20] and may exhibit limited prevention capabilities as they lack proactive defense modes of operation.

It is generally admitted, that in the case of static systems, an attacker, given enough time, will eventually discover and exploit their vulnerabilities [30]. For example, vulnerable and unpatched software components may be discovered (in-

cluding those related to cybersecurity solutions in place), encryption keys may be exposed or the design and execution of coordinated network attacks (e.g., denial of service type of attacks) may be facilitated. This is the reason why there is a need for more dynamic and proactive solutions.

2.2 Current IoT Environments and Their Security Solutions

The proliferation of IoT devices has introduced unique security challenges due to their limited resources, diverse environments, and wide deployment, as shown in Figure 2.1. Traditional security measures often fall short in addressing these challenges, necessitating specialized solutions tailored for IoT ecosystems.

2.2.1 IoT Environments

The Internet of Things (IoT) encompasses a diverse ecosystem of interconnected devices, ranging from consumer gadgets to industrial sensors, designed to collect, process, and exchange data autonomously. IoT environments pose unique cybersecurity challenges due to the heterogeneity of devices, resource constraints, and distributed nature of deployments [7].

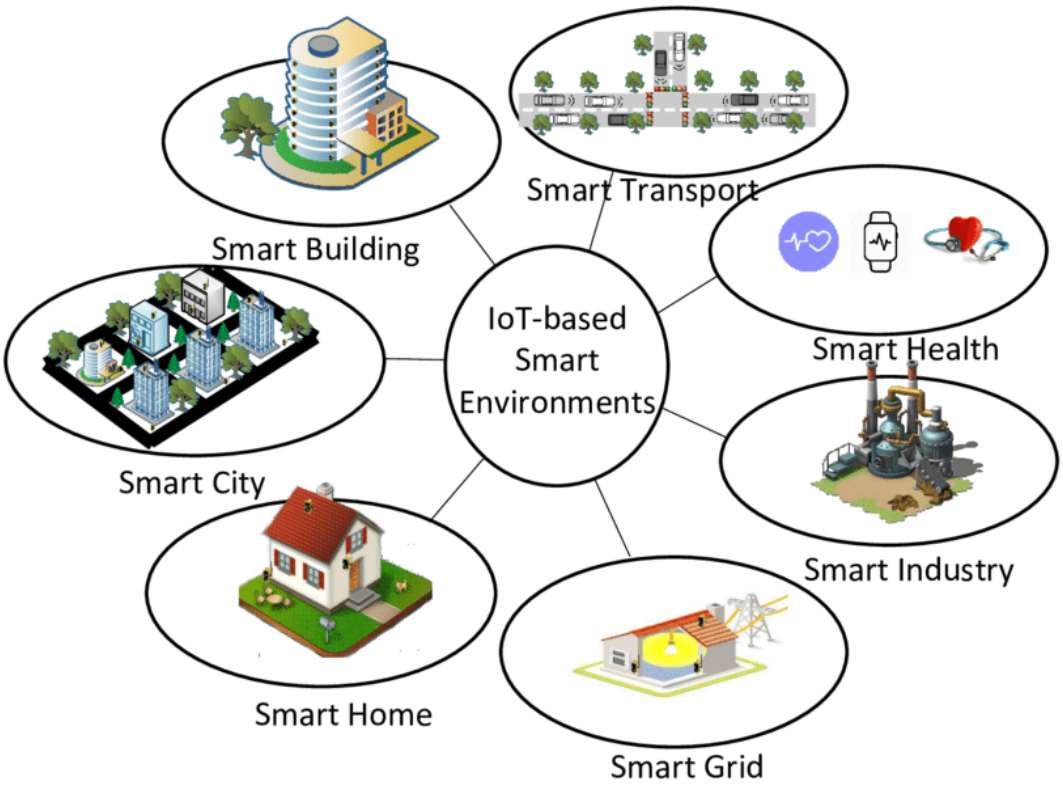

Figure 2.1: IoT environments [2]

Securing IoT ecosystems requires a multidimensional approach that addresses device security, network integrity, data privacy, and ecosystem resilience. Research efforts have focused on developing lightweight security protocols, intrusion detection systems, and anomaly detection algorithms tailored to the constraints of IoT devices [34].

Several notable cybersecurity incidents, such as the Mirai botnet attack in 2016, have underscored the vulnerabilities inherent in IoT deployments and highlighted the importance of robust security measures [24]. Subsequent research and industry initiatives have sought to improve IoT security through standards development, device certification programs, and collaboration among stakeholders [26].

Furthermore, advances in edge computing and blockchain technology offer promising avenues for improving IoT security by enabling distributed trust models, secure

data sharing, and decentralized device management. Research by Ju et al. (2018) explores the integration of blockchain-based solutions with IoT systems to address security and privacy challenges effectively [43].

2.2.2 Device-Level Security Solutions

Device-level security in IoT relies on several mechanisms, including lightweight encryption, firmware updates, and hardware security modules (HSMs). Given the limited computational power of IoT devices, lightweight encryption algorithms such as Elliptic Curve Cryptography (ECC) are commonly used. While these algorithms provide a certain level of security, they remain vulnerable to sophisticated attacks due to constrained key sizes and reduced computational capabilities [16], [28].

Firmware updates are essential for addressing vulnerabilities, yet the highly distributed and diverse nature of IoT devices makes timely updates a challenge. Many IoT devices also lack the capability to perform secure over-the-air (OTA) updates, leaving them exposed to long-term security threats [1]. Additionally, some IoT devices incorporate HSMs to protect cryptographic keys and perform secure operations. Although HSMs enhance security, their high cost makes them impractical for many IoT applications [17].

2.2.3 Network-Level Security Solutions

At the network level, security strategies include network segmentation, secure communication protocols, and intrusion detection systems (IDS). Network segmentation helps limit the impact of compromised devices by isolating them into distinct segments. However, attackers who gain access to the network may still be able to bypass static segmentation rules [36].

Secure communication between IoT devices and other network entities is achieved through protocols such as Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). These protocols enhance security but introduce additional latency and computational overhead, which may be challenging for resource-constrained IoT devices [33]. Another critical network security measure is the use

of IDS tailored for IoT networks. These systems monitor network traffic for suspicious activity that may indicate an attack. However, due to the high volume and variability of IoT traffic, IDS often generate a significant number of false positives, making incident response more complex [31].

2.2.4 Cloud-Based Security Solutions

Cloud-based security solutions play a vital role in securing IoT ecosystems, with cloud IoT platforms and edge computing being two key approaches. Many IoT deployments depend on cloud-based platforms for device management and data processing. These platforms often include built-in security features such as encryption, access control, and anomaly detection. However, relying on third-party cloud services raises concerns regarding data privacy and control [9].

An alternative to cloud computing is edge computing, which involves processing and analyzing data closer to its source rather than sending it to centralized cloud servers. This approach reduces latency and enhances security by minimizing data exposure. However, securing edge devices themselves presents an additional challenge, increasing the complexity of the overall security infrastructure [38].

2.2.5 Limitations of Current IoT Security Solutions

Despite various security measures, IoT security solutions face several limitations. One significant challenge is **resource constraints**, as many IoT devices lack sufficient processing power, memory, and energy resources to support strong security mechanisms, making them vulnerable to attacks.

Another challenge is **heterogeneity**, as the diversity of IoT devices, operating systems, and communication protocols complicates the implementation of universal security solutions. Additionally, **scalability** remains a pressing issue, as maintaining consistent security across extensive IoT deployments with thousands or even millions of devices is difficult.

Lastly, **interoperability** issues arise when integrating security solutions across different IoT platforms and ecosystems. This often requires custom configurations,

leading to compatibility challenges and making seamless security implementation more complex.

2.2.6 Wrap up for Current IoT security Solutions

Current IoT security solutions offer various degrees of protection but are often limited by the unique constraints and requirements of IoT environments. These limitations highlight the need for innovative approaches.

2.3 The MTD Concept

Moving Target Defenses (MTD) are in principal cyber-defense techniques that try to make a series of aspects of the system under protection dynamic in order to thwart attackers that rely on their static nature. Information gathered on those aspects of the system becomes highly temporal and vulnerabilities or other weaknesses have a limited time to be discovered or exploited [30]. A more formal definition, attributed to [47], describes MTD techniques as "constantly changing a system to reduce or move the attack surface available for exploitation by attackers". Recent work by [41] further categorizes MTD strategies into "intelligently affordable, optimized, and self-adaptive" frameworks, emphasizing their applicability to resource-constrained environments like IoT.

The above definition and descriptions imply that system designers acknowledge the fact that vulnerabilities are present in any system and that an attacker of a system that remains static will be successful in exploiting it if he/she has enough resources (time in particular). Therefore MTD acts as a way to limit information leakage from the system and minimize the available time window that an attacker has in order to gather them. On top of that, should an attacker manage to execute any initial steps of the attack, their effectiveness can be minimised since the change of system attributes will soon render them useless and it may not be possible to be propagated again. Recent evaluations by [27] demonstrate that reinforcement learning can optimize MTD deployment in probabilistic attack graphs, reducing the attack success rate by up to 40% in simulated IoT environments.

2.3.1 Fundamental Questions for MTD

To design and implement an MTD solution, three fundamental questions need to be answered [10]: (i) *what* system properties and components have to be made dynamic, (ii) *how* this dynamic nature is going to be achieved (e.g. shuffling, diversification or redundancy) and finally (iii) *when* to perform state changes (at specific times, after an event or both). Typically, MTD techniques are categorised according to the system layer that they are applied [42]:

1. *Dynamic Data* comprises of techniques that change the format, encoding or representation of application data. Examples are changes in semantics of data, using encryption or applying different encryption keys or algorithms etc.

2. *Dynamic Software* techniques change an application's binary code dynamically, e.g., binary objects shuffling and application diversification.

3. *Dynamic Runtime Environment* techniques alter the execution environment dynamically, e.g. using different main memory regions or executing the application in different processors.

4. *Dynamic Platform* techniques change the computing platform properties in software or in hardware (for example executing an application in a different CPU architecture, or replacing the operating system or the virtual machine instance in a cloud environment).

5. *Dynamic Networks* techniques change network properties, such as network protocols, addresses, and topology.

2.3.2 Benefits of MTD-based Security Solutions

MTD solutions offer several significant benefits:

1. Increased Attack Complexity: By constantly changing the attack surface, MTD increases the complexity and time required for attackers to succeed. Hybrid approaches combining shuffle and diversity, as proposed by [8], have shown to reduce attack success rates by 60% in IoT systems.

2. Reduced Attack Success Rate: The dynamic nature of MTD means that even if attackers find a vulnerability, it may no longer exist by the time they attempt to exploit it. For example, [39] reports a 45% reduction in successful attacks on IoT applications using dynamic network reconfiguration.

3. Proactive Defense: Unlike traditional reactive defenses, MTD proactively disrupts potential attack paths. Recent advances in AI/ML-driven MTD, as explored by [5], enable adaptive defense mechanisms tailored to IoT threat landscapes.

2.3.3 Challenges and Considerations

While MTD presents a promising approach to cybersecurity, it also introduces new challenges:

1. Resource Overhead: Frequent changes can introduce performance overhead and require significant computational resources. Lightweight solutions like those in [46] address this by combining MTD with intrusion detection for IoT. It should be noted that the adoption of an MTD-based solution does not negate traditional cybersecurity approaches - on the contrary, it adds an additional layer of protection and can be successfully coordinated with other security approaches for optimal results.

2. System Compatibility: Ensuring compatibility and seamless integration with existing systems and workflows can be challenging. The work of [11] proposes model-based MTD frameworks for cyber-physical systems, offering insights for IoT.

3. Operational Complexity: Managing and orchestrating continuous changes requires sophisticated tools and expertise. [23] demonstrates how reinforcement learning can automate MTD strategy selection in IoT.

4. Expansion of the attack surface: although it is not generally discussed, similarly to the vulnerabilities of traditional cybersecurity tools, the frameworks that control the dynamic reconfiguration of the systems, can be the targets of attacks themselves.

2.4 Related Work

From the aforementioned, it becomes obvious that MTD techniques can be applied to a huge range of system functions and properties. It also becomes apparent that all these techniques cannot be applied to all classes of systems, applications and deployments as there are specific limitations that prohibit their practical use. In particular, for IoT systems that is the focus of this work, there are severe limitations stemming from the fact that in IoT applications resource constrained devices are fundamentally employed. Those constraints may involve limited computational capabilities, memory, storage and energy resources and may extend to other areas such as the need to securely boot devices with specific binaries and audited software. In [35], the applicability of MTD techniques for IoT applications is examined and the authors stress that several techniques will have limited or no applicability in the IoT domain. More specifically, MTD techniques from the dynamic software and dynamic runtime environment domains can have limited use in IoT applications due to their increased requirements in memory and storage spaces while techniques from the dynamic platform domain may not be technically (i.e. in terms of the form factor of the sensors/IoT devices) nor economically feasible (in terms of cost). Recent work by [22] corroborates these findings, emphasizing the need for network-layer MTD in IoT.

As such, MTD techniques that apply in the dynamic network domain appear to be the most applicable to IoT applications. The dynamic network domain can be subdivided into two main areas: the identity and non-identity based randomization [35]. The first area covers MTD techniques that try to introduce a dynamic nature in the network identity of the participating systems, such as the physical address, the logical address and the port number. The most common approach is to shuffle IP addresses (IPv4 or IPv6) in a network periodically, as it has been demonstrated that this is easily feasible in IoT environments [21]. In [15], the authors present MT6D (Moving Target IPv6 Defense), a scheme that provides a non-deterministic IPv6 dynamic addressing system that modifies the IP and port addresses of two communicating end-devices, aiming to preserve user privacy and protect against Denial-Of-Service (DoS) and Man-In-The-Middle (MITM) types of attacks. While

the proposed scheme appears attractive for IoT deployments due to its adoption of the immense addressing space of IPv6, it is primarily designed for full-scale systems and devices and therefore it has to be properly adapted for IoT applications to increase the number of end-devices that participate and to significantly decrease its computational requirements. Indeed, this has been the effort in many subsequent works [37], [32], [45], [44] that have tried to adapt MT6D in order to be used in network protocols suitable for IoT applications, such as the 6LoWPAN protocol. Other efforts, focused on designing solutions that adopted this MTD technique for ad-hoc infrastructure-less networks (MANET) [3] or routing protocols used for low power and lossy networks (RPL) [6].

Dynamic network non-identity based MTD techniques focus on aspects such as changing the network proxies, shaping traffic, dynamic protocol information, time scheduling of periodic traffic etc. In [12], the authors demonstrate an MTD technique that shuffles the main base station that is used in WSN networks so that external attackers may not identify it through localization techniques (such as signal-strength indicator - RSSI) and take it down, thus rendering the whole network unusable. For Smart Grid applications, [4] an MTD scheme that randomizes the configuration parameters of the Advanced Metering Infrastructure (AMI), such as report size, interval and relaying nodes, is presented in order to provide proactive defense against mimicry attacks. Recent research by [23] extends these principles by using reinforcement learning to dynamically select MTD mechanisms for zero-day IoT attacks.

2.5 Towards an MTD Solution Suitable for IoT Environments

Despite the advancements in IoT security, existing solutions often fall short in addressing the dynamic and evolving nature of cyber threats. MTDIoT aims to fill this problem by introducing a Moving Target Defense (MTD) approach specifically tailored for IoT environments.

Based on the analysis of the previous subsections, we identify that an MTD-based

security solution suitable for IoT environments, should have the following features and characteristics. These, along with the specific requirements stemming from the actual deployment of the security solution that is developed in this work (presented in Section 3), will formulate the tool that will be presented in the next sections.

- A dynamic network environment: As static deployments and configurations nourish successful attack efforts, a promising MTD solution should introduce dynamic changes to the network configuration, making it difficult to predict and target specific devices.
- A lightweight solution: Many IoT devices operate with limited processing power, memory, and energy resources, thus excluding solutions that prove to be complex. Therefore, any successful security solutions must be lightweight and resource-efficient, ensuring that it may be executed efficiently even by the most constrained devices.
- A scalable solution: IoT deployments are more often than not large-scale, involving a vast number of diverse devices, each with its own security requirements. As such, a proposed security solution should be able to scale along large numbers of clients that are heterogeneous in nature.
- Proactive Defense Mechanisms: while specific reaction and mitigation measures should be employed in case of a detected cyberattack, the very basic concept of the proposed MTD solution must center around the proactive nature of the defense mechanism. This proactive approach can disrupt potential attack paths before they can be exploited, significantly reducing the risk of successful attacks.
- Isolation of Compromised Clients: In traditional IoT networks, identifying and isolating compromised devices can be challenging and time-consuming. MTDIoT enhances this process by automatically excluding compromised clients from updated network configurations, effectively isolating them from the rest of the network and preventing further spread of the attack.
- Integration with Existing Security Components: Many IoT security solutions operate in isolation, leading to fragmented security postures. A successful

security solution should be designed to integrate seamlessly with other complimentary security components, such as firewalls, IDS, and endpoint security solutions. This integration will provide a comprehensive security framework that can leverage the strengths of multiple defenses.

Chapter 3

Architecture

In the previous chapter, the foundation of the security solution that is proposed in this work has been laid. After an analysis of the current security solutions and the issues that are associated with deploying them in the context of IoT environments has been provided, a rough outline of a security mechanism that provides potential mitigations to these problems based on the concept of Moving Target Defenses has been provided.

In this chapter, the architecture of a concrete security solution, named MTDIoT, is presented. This solution has been designed and deployed in the context of a European Research Project (H2020 IntellIoT) and it aims to incorporate those desirable features and characteristics presented in Section 2.5 into a security tool that is able to handle the design requirements of actual real-world use cases that have been brought forth by the partners of the project consortium.

In Section 3.1 the design requirements of the security tool will be presented. Section 3.2 will provide a high level view of the tool architecture as well as its main components, functionalities and communication methods. The detailed implementation of the tool will be described in the next chapter.

3.1 Design Requirements

MTDIoT has been conceived, designed and implemented as part of Technical University of Crete's research activities within the IntellIoT project, funded by the European Union's Horizon 2020 research and innovation programme / ICT-56-2020 "Next Generation Internet of Things" under grant agreement number: 957218. As such, while MTDIoT is a standalone security solution that can exist autonomously without any reference to the rest of the IntellIoT project, it should be understood that the requirements, design choices and overall functionality are heavily influenced by the operating restrictions, use case requirements and overall architecture of the framework developed within IntellIoT.

Specifically, the project focused (the IntellIoT project started on October 2020 and finished at the end of January 2024) on three major IoT use cases: (i) agriculture (tractors performing semi-autonomous harvesting), (ii) healthcare (patients monitored by sensors) and (iii) manufacturing (automated plants shared by multiple tenants who utilise machinery from third-party vendors). In all these IoT deployments, a network of interconnected devices formed a typical tree structure. A large number of leaf nodes (mostly occupied by simple devices such as sensors, cameras, actuators, etc.) communicates with edge nodes. The latter are typically devices fitted with more complex computational components and power/energy resources, that act as traffic gathering nodes performing initial computations related to the specific IoT application and propagating only necessary information to the higher levels of the infrastructure for further processing or storage. The edge nodes communicate with a small pool of larger computing systems, mostly servers hosted in cloud provider facilities that either perform the most intense computational tasks or provide storage and management functions for the overall deployment. In that context, the leaf and edge nodes constitute the focus of the MTDIoT security solution.

In all use cases, the security solution should be deployed on preexisting network infrastructure and organization. As such, the security solution under design should be able to adapt to the underlying requirements of the deployment and not dictate specific network requirements. In all use cases, a large number of different networks

and topologies was also employed. For example, in the agricultural use case, the large physical dispersion of nodes, actuators, sensor devices, vehicles and static infrastructure necessitated the use of 5G, WiFi, Bluetooth, Bluetooth LE and other protocols both in centralised and peer-to-peer communications.

Therefore, the MTDIoT application has to operate on higher layers of the network stack to overcome the differences in the physical communication protocols. As such, it has to be designed to operate on network layer 3 and to rely on IP addresses and routing.

The potentially large physical area of deployment (such as an agricultural field or forest or metropolitan area) of participating IoT nodes excludes the ability to physically protect all nodes. This means that legitimate nodes may be physically tampered and malicious nodes may be planted within the area of coverage to either present themselves as legitimate nodes or issue/intercept radio signals in order to influence normal communications.

The security solution must be able to communicate only with nodes registered during the initial setup of the network that are handled by the IoT deployment administrators. Additionally, all signaling and commands related to the security operations of the MTDIoT should be handled in encrypted and protected traffic, using channels of communication separate from normal data exchanges between nodes. Additionally, since legitimate nodes may be compromised, the system should have a mechanism to detect such anomalies (or at least be able to communicate with other security tools that may provide threat detection capabilities) and protect itself by excluding such nodes from communications related to the core of its function, i.e. to the updates of the moving features.

As already mentioned, the IoT deployments considered employ numerous devices, spanning from simple sensors and actuators to cloud servers. Therefore, the security solution under design needs to be able to have components that can be executed even on the simplest devices (in terms of computational and energy resources), while it has to be able to provide software components that can easily be compiled for different CPU architectures (as a result of the high heterogeneity of the potential nodes) or be easily deployed in some containerised form for more

complex nodes that may run a large number of services.

Furthermore, although the network itself may be private, signals can be intercepted, and external entities may try to present themselves as legitimate nodes. On top of that, leaf devices can be considered susceptible to physical tampering, and therefore legitimate nodes may be overtaken by malicious entities. As such, there have to be mechanisms in place to isolate nodes that have been compromised. It should also be noted that misbehaving nodes (i.e. nodes that for any reason be it malicious or of any other origin - for example a malfunction - behave outside of the expected norms) are treated in the same way, as they constitute a threat to the proper functioning of the IoT deployment.

According to the analysis in the previous subsection and considering the very low computational, memory, storage and energy resources of the targeted devices, we consider that MTD techniques applied in the dynamic software, runtime environment, and platform domains cannot be effectively used and are not considered. The focus of MTDIoT primarily falls on the dynamic network domain and more specifically in the network-identity based one. However, the exponential rise of IoT applications that deal with sensitive information and therefore raise significant security and data privacy concerns, forces us to consider data encryption schemes, especially for data transmissions in channels. As such, the MTDIoT system must be able to include techniques that properly consider encryption schemes as well (dynamic data domain).

The MTDIoT is developed as we previously mentioned as a client-server set of software components that manage the network configuration of the edge network of the IoT application. The MTDIoT server is installed in a computationally-capable edge node and it is responsible for managing all the clients, handling events like warnings originating from external security components and generating new configurations. The MTDIoT clients are installed in each leaf node and are responsible for applying the configuration sent by the server, encrypting network traffic, and transmitting it through tunnels (PPTP [19] with IPSec [18]) to avoid packet sniffing and Man-In-The-Middle attacks.

Each MTDIoT client is provided with a new virtual network interface (*mtd0*) with

its own IP address, named internal IP (*IntIP*). That means that each client has a pair of IP addresses, the IP address with which it communicates with the network, named external IP (*ExtIP*) and the internal IP. Such a pair is considered a *route*. When an application uses the *mtd0* interface, the packets that are generated use the internal IPs. To reach another client, the corresponding external IP must first be found (i.e. a route entry). Once this is accomplished, the packet is encrypted and encapsulated in a UDP packet using the external IPs.

We acknowledge that MTDIoT is only one part of the cyber-defense strategy that needs to be put in place, especially in the context of the highly secure IoT applications that are being developed. This means that MTDIoT should be able to cooperate with other commonly security solutions, such as Intrusion Detection Systems, Firewalls, Security Platforms etc and augment their capabilities. These security solutions can also be employed to expand the capabilities of MTDIoT, e.g. by issuing warnings or triggering events that the MTD system should consider.

3.2 MTDIoT Architecture

Taking into consideration the design requirements, the MTDIoT application is built as a client-server system, as shown in Figure . A central node located at the edge of the network is tasked with the generation of new configurations and the propagation of this information to all leaf nodes that are connected to it. This way all the computational complexity is gathered on capable nodes and resource-constrained devices need only execute simple network configuration tasks of their own communication interfaces. The solution is scalable, as MTDIoT may work in an hierarchical mode with multiple servers overseeing different parts of the hierarchy. The communication between MTDIoT modules (server and clients) is realised through a secure channel, implemented through a message broker.

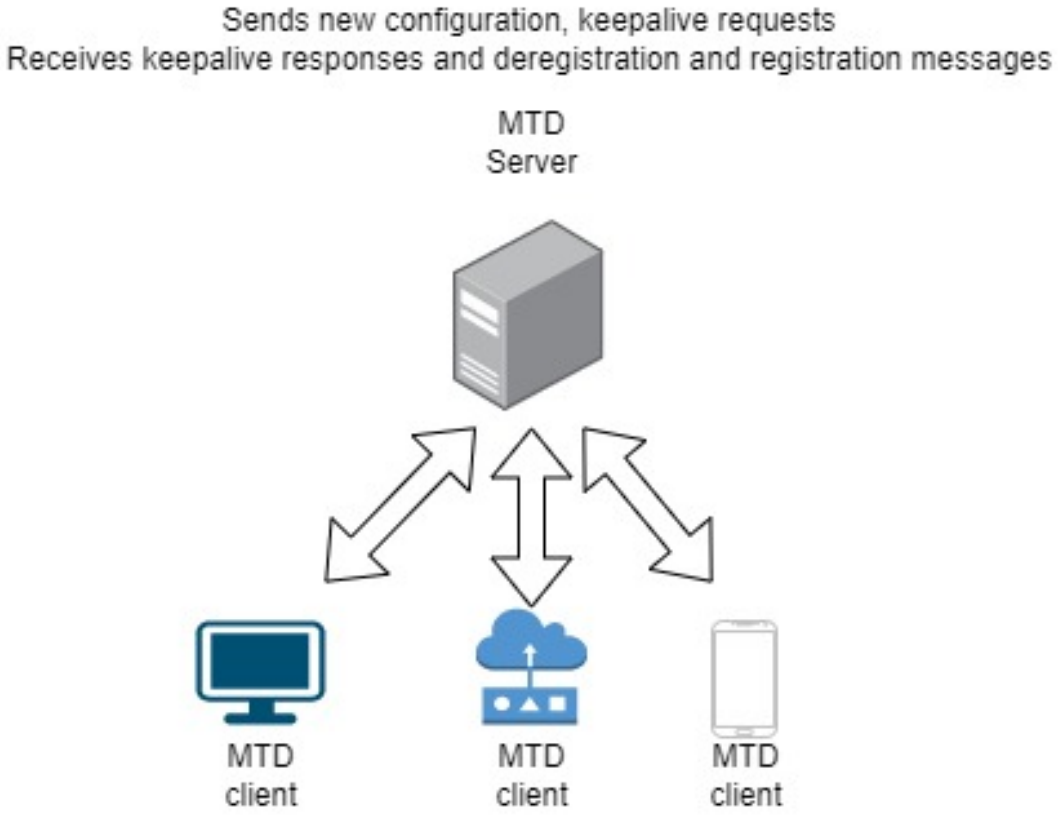

Figure 3.1: MTD Architecture

3.2.1 Specification of Dynamic Configurations

In designing MTDIoT, three properties need to be defined (*what, how, when*).

1. The first property (*what*) deals with the system properties that need to be made dynamic. MTDIoT supports changing the IP addresses of the nodes as well as the port numbers used for intercommunication. Additionally, MTDIoT specifies the encryption algorithm used for transmitting messages and handles encryption key changes.

2. The second property (*how*) specifies valid states and how to move between them. The MTDIoT server reserves a pool of legitimate IP address as well as ports that cover the range beyond well-known and used ports. The server

specifies and initial configuration that is dispatched to all clients and they switch to new configurations using a shuffling approach (randomization), making sure in the process that new configurations are valid and sufficiently different from recent changes. Concerning the encrypted communications, server generates encryption keys for the clients and dispatches them through a secure channel.

3. To make changes in the configurations (the *when* property), a hybrid approach is used. As a proactive defense mechanism, the MTDIoT embraces a timed process. In fixed or random time intervals, a new configuration is generated and dispatched to all participating nodes. However, since the system may be connected to other security modules, new configurations can also be generated when an event takes place. Most common events considered are warnings or alert messages from attack detection systems, security assurance modules or human security administrators. In that context, MTDIoT functions as a reactive defense mechanism, since these events are indicators of an active attack that has been detected. Since MTDIoT is able to use the events as a triggering mechanism, it is often the case that offending nodes have been identified. If this is true, MTDIoT is able to isolate these malicious nodes by not propagating to them the new configuration settings. As such, they cannot update their network configurations and therefore they cannot communicate with other nodes anymore.

3.2.2 MTD Server

The MTDIoT server is comprised of three main software components, as shown in Figure 3.2. The first component handles the registration and deregistration of nodes in the system. This enables the system to dynamically add or remove nodes and maintain a correct functionality by properly allocating names, configurations and resources to all connected entities. All registered clients, along with the necessary information that is required for each one of them (e.g. internal names for the message broker system) are stored in an internal database. This database is used by the third major component of the server in order to generate new configurations and dispatch them to all clients.

The configuration generator component is initiated with a static configuration at startup and updates this configuration upon fixed or random time intervals. It can be interconnected through the same message broker to other security tools and receive warnings in a specific format. Upon reception of such a warning, the configuration generation process is triggered. Compared to the timer events, warnings may include information about offending nodes (called error-block messages). In that case, a new configuration is generated with the offending nodes removed from the routing table. The configuration information is not transmitted to them and as such, they maintain stale information about the network state and become practically isolated, unable to establish connections with any other node.

It should be mentioned that these kind of warnings may be temporal. For example, a node may be erroneously marked as malicious or a malfunctioning node (appearing as malicious because of its unpredictable behavior) may be restored. The system supports error-allow messages to restore marked clients and reintroduce them to the network.

Lastly, the server maintains a state for each client. Because of the dynamic nature of the IoT applications, it is frequent that leaf nodes may have intermittent connections to the rest of the system. For example, this may be attributed to the nature of the network links, to limited energy resources that have to be replenished, to power management features of low-end devices, etc. For these reasons, MTDIoT clients dispatch specific messages about their liveliness state that the server uses to draft an updated image of the overall network.

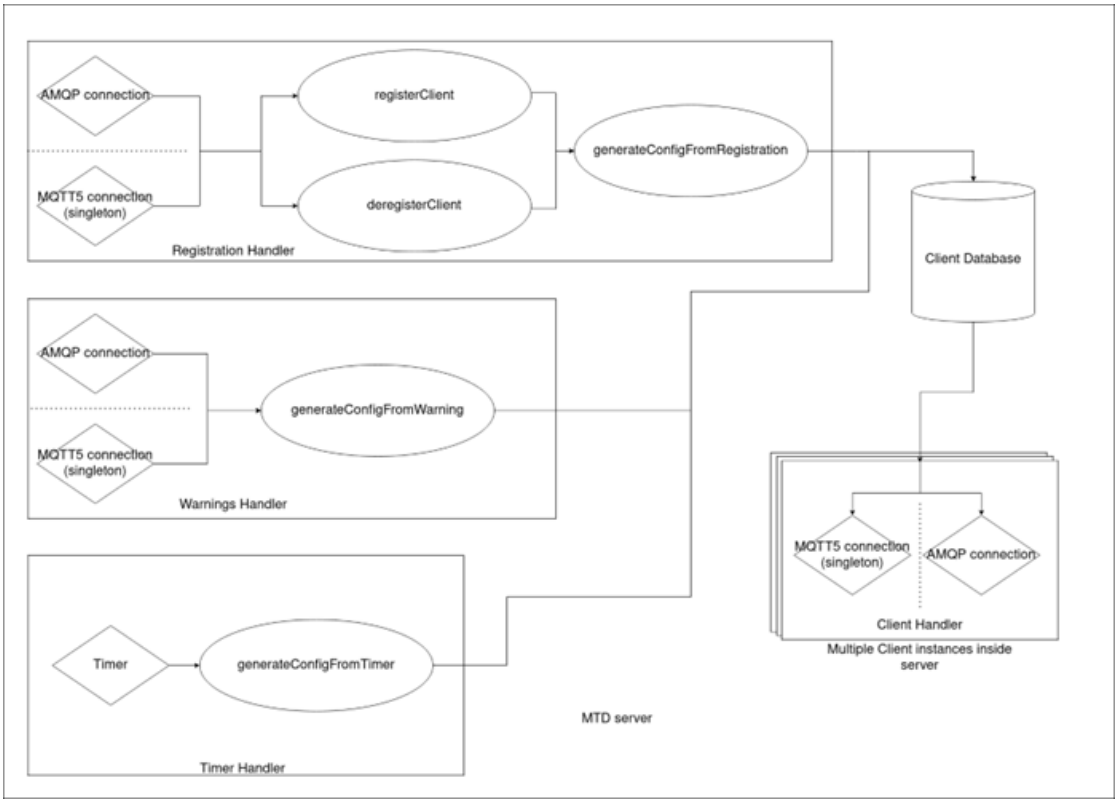

Figure 3.2: MTDIoT Server

3.2.3 MTD Client

The MTD clients are responsible for managing the network configuration, maintaining an encrypted connection between each other using the routing table sent by the MTD server and applying any changes the server sends. An MTD client lifetime is comprised by three distinct as shown in Figure 3.3.

The first phase (initialization phase) handles the registration with an MTD server. Pre-shared credentials are used to uniquely identify each client. Legitimate clients can then initialize their routing table. The second phase involves the normal operation of the system. MTD clients are equipped with a configuration handler and a VPN module. These two are responsible for receiving and enforcing the periodic MTD server updates. Each new update needs to be applied without breaking benign ongoing communications between nodes. For this reason, the last

valid routing configuration is allowed to remain active for a small period of time in order for the changes to propagate and be enforced throughout the network.

The last phase (deregistration phase), is carried out to deregister the client from the MTD server. When an MTD client is gracefully shutting down, it must notify the MTD server, so that it becomes deregistered from the system. This reduces both the CPU and network load for the MTD server. To cover the case of forced termination or when the node operation is disrupted, a keep alive mechanism is used. When a registered client fails to adhere to the keep alive mechanism, it is automatically deregistered by the MTD server. This works both ways, if the MTD server is unresponsive, the MTD client can deduce that it has lost connection.

As mentioned, the MTDIoT system supports shuffling between different encryption algorithms and keys. The MTDIoT client handles this process transparently from the application layer. An included cipher receives unencrypted data from the application running on the local device, encrypts them according to the rules specified by the MTDIoT server and then dispatches them to other nodes. The inverse functionality is realised upon reception of encrypted messages. The cipher is responsible for keeping the active algorithm and key in sync with the server. At any point there are two states of the cipher, the current and the previous state. The current state is used when encrypting packets and the default state when decrypting packets. If the decryption with the current state fails, it falls back to the previous state. That is expected behaviour when an update to the cipher occurs, since in-flight packets encrypted with the previous key and/or algorithm might reach the client after the update. The previous state is kept active for a limited period of time after each update.

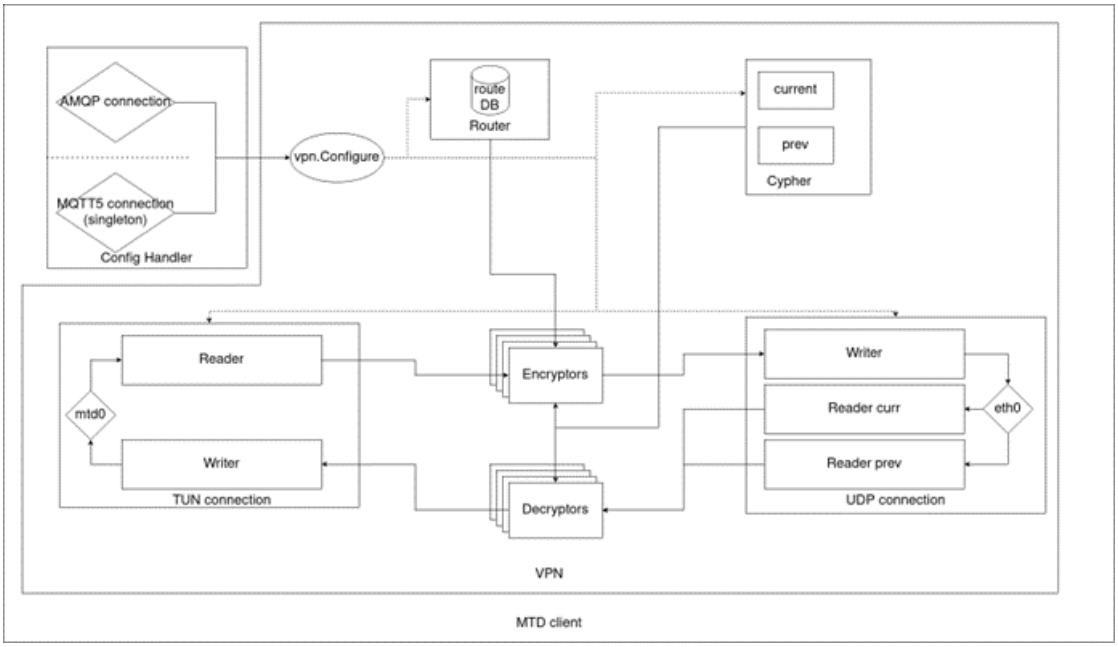

Figure 3.3: MTDIoT Client

Chapter 4

Implementation

4.1 Implementation

The MTDIoT components are implemented in Golang, making it easy to build for new architectures. They can be installed as native applications or Docker containers, depending on the deployment needs. The client component runs efficiently on low-spec devices that have the ability to execute a Linux network stack. Regardless of the deployment strategy, elevated rights are needed for the client in order to create the mtd0 interface and manage server changes.

4.2 Golang in CyberSecurity

GoLang, or Go, is a statically typed, compiled programming language developed by Google, renowned for its simplicity, efficiency, and concurrency support. In recent years, Go has gained traction in the cybersecurity community due to its robust performance, ease of use, and extensive standard library, making it well-suited for developing secure and scalable applications.

Go's simplicity and readability facilitate rapid development and maintenance of cybersecurity tools and frameworks. Its built-in concurrency features enable efficient parallel processing, essential for tasks like network scanning, threat detection,

and log analysis. Moreover, Go's static typing and memory safety features enhance code reliability and mitigate common security vulnerabilities, such as buffer overflows and type mismatches.

Numerous cybersecurity projects and frameworks leverage Go, including penetration testing tools, network security utilities, cryptography libraries, and threat intelligence platforms. The language's growing ecosystem and community support contribute to its popularity as a preferred choice for building resilient and performant cybersecurity solutions.

GoLang's crypto package provides robust support for encryption and decryption through implementations of various cryptographic algorithms and protocols. Key aspects include symmetric and asymmetric encryption, hashing, digital signatures, and secure key management, essential for securing sensitive data and communications.

Symmetric encryption algorithms, such as AES (Advanced Encryption Standard), utilize a single key for both encryption and decryption, offering fast and efficient cryptographic operations suitable for securing data at rest and in transit. GoLang's crypto package includes standardized implementations of AES and other symmetric encryption algorithms, ensuring interoperability and compatibility with cryptographic standards.

Asymmetric encryption algorithms, like RSA and ECDSA, employ public and private key pairs for encryption and decryption, enabling secure communication and digital signatures. GoLang's crypto package provides robust implementations of asymmetric encryption algorithms, along with utilities for generating and managing cryptographic key pairs securely.

Additionally, GoLang supports hashing algorithms like SHA-256 and
cryptographic primitives for generating secure random numbers, computing message digests, and deriving cryptographic keys. By leveraging Go's crypto package, developers can implement robust encryption and decryption mechanisms to protect sensitive data and communications effectively.

4.3 MTD Server

The MTD Server is the core component responsible for managing dynamic network configurations and client interactions in our IoT environments. It handles client registration, reconfiguration, and mitigation of potential security threats by dynamically altering network parameters such as IP addresses, encryption settings, and routing tables. Below is a detailed breakdown of its implementation.

Overview of the MTD Server

The MTD Server is designed to:

- Register and deregister clients dynamically.
- Generate and distribute new configurations periodically.
- Respond to security warnings by adjusting network parameters.
- Maintain an updated routing table to facilitate secure communication between nodes.

The server implementation spans multiple components, each with a specific role in the system's operation.

Key Components of the MTD Server

Client Database Management (main.go)

The client database (clientdb) manages all connected clients. It maintains:

- A list of clients indexed by name and IP.
- A record of each client's assigned internal IP.
- UDP and cipher settings for encryption.
- A routing table for internal communication.

The clientdb structure provides read-write locks to ensure thread safety while updating or retrieving client information.

Configuration Trigger Mechanism (main.go)

Configuration updates in the MTD Server are triggered by four events:

- **Timer-based updates (timerTr)** – Regular configuration changes to enhance security.
- **Registration-based updates (registrationTr)** – Triggered when a client joins the network.
- **Deregistration-based updates (deregistrationTr)** – When a client leaves the network.
- **Warning-based updates (warningTr)** – Applied when an anomaly is detected (e.g., an attack).

The configTrigger struct ensures a consistent mechanism for processing these triggers.

Configuration Generation Mechanisms

The MTD Server generates new configurations under different circumstances. The following sections describe how each type of configuration is created.

Configuration from Client Registration (configFromRegistration.go)

When a client registers, the server:

1. Assigns an internal IP dynamically by checking available addresses.
2. Updates the routing table, ensuring secure communication.
3. Sends the new configuration to all clients, informing them of the updated network topology.

The generateConfigFromRegistration function is responsible for handling this process. It:

- Expands the subnet mask dynamically if the number of clients exceeds the available IP range.

- Ensures that blocked clients are not included in the routing table.
- Distributes new configurations to all active clients.

Configuration from Timer (configFromTimer.go)

The MTD Server periodically generates new configurations to proactively defend against attacks. The generateConfigFromTimer function:

1. Modifies network parameters such as UDP port numbers, internal IPs, or encryption keys.
2. Updates the routing table, ensuring that all active clients receive the latest network settings.
3. Resets the configuration timer, ensuring periodic reconfiguration.

By dynamically changing these parameters, the server reduces the predictability of the network, making it more difficult for attackers to exploit static configurations.

Configuration from Security Warnings (configFromWarning.go)

If an attack is detected, the MTD Server can isolate compromised clients. The generateConfigFromWarning function:

1. Verifies the source of the warning (e.g., Intrusion Detection System).
2. Identifies the affected client and determines whether it should be blocked or allowed back into the network.
3. Updates the routing table to exclude malicious clients from the network.
4. Notifies all active clients of the change, ensuring that the compromised client is removed from communication.

This mechanism prevents lateral movement of an attacker within the network and helps contain potential security breaches.

Communication Flow between the MTD Server and Clients

The server and clients communicate through message queues using a broker-based system (e.g., AMQP). The interaction includes:

- **Client Registration:** Clients send a registration request, and the server assigns an internal IP.
- **Configuration Updates:** The server pushes new configurations to all clients whenever a change is triggered.
- **Keep-Alive Messages:** Clients periodically send keep-alive messages, and the server verifies their status.
- **Warning Processing:** If an anomaly is detected, the server updates the configuration to mitigate potential threats.

The message queues ensure asynchronous and efficient communication, reducing latency and improving system responsiveness.

Security Measures in the MTD Server

To maintain security and integrity, the MTD Server employs:

- **Encryption:** Uses cipher settings to protect communication between nodes.
- **Dynamic Reconfiguration:** Periodically updates network parameters to prevent attackers from gaining long-term access.
- **Client Verification:** Ensures only authenticated clients can register and receive configurations.
- **Logging and Monitoring:** Maintains logs of all configuration changes and client activities for auditing and debugging.

4.4 MTD Client

The MTD Client is responsible for managing its connection to the MTD Server, handling dynamic configuration changes, and maintaining its status through pe-

riodic keep-alive messages. Implemented in Go, it utilizes various libraries for networking, logging, and configuration management.

Key Components

- **Registration Configuration:** The `registrationCfg` struct defines the necessary fields for the registration and deregistration process, including action type, node name, IP address, MAC address, and public key. This structure ensures the server can accurately identify and authenticate the client.
- **Client Struct:** The `Client` struct encapsulates essential properties such as node information, network connections for registration and configuration, and a timer for managing keep-alive requests.
- **Initialization:** The `NewClient` function initializes a new MTD Client instance by parsing the client's configuration and establishing communication channels for registration, configuration updates, and keep-alive messages based on the chosen broker protocol (AMQP or another).

Communication Flow

Registration and Deregistration

Upon starting, the client registers itself with the MTD Server using the register method. This method constructs a registrationCfg object, marshals it into JSON format, and sends it to the server. In the case of client termination, the deregister method sends a deregistration request to inform the server.

Handling Configuration Updates

When the MTD Server sends new configurations, the client invokes the handleNewConfig method. This method unmarshals the JSON payload into a vpn.VPNConfig object and applies the new settings using the vpn.Update function. The timestamps for applying the configuration are logged to track performance.

Keep-Alive Mechanism

The keep-alive functionality is crucial for maintaining a stable connection with the server. The handleNewKAReq method processes incoming keep-alive requests from the server and responds with the appropriate status. A dedicated goroutine monitors the keep-alive timer; if the client does not receive a request within the specified timeframe, it exits the application, indicating a potential failure in communication.

Error Handling and Logging

The implementation incorporates structured error handling and logging through the rlog package. Various events, such as registration, configuration updates, and keep-alive interactions, are logged at different levels (Debug, Info, Critical). This approach ensures that any issues encountered during the client's operation are documented for further analysis and troubleshooting.

Client Lifecycle Management

The client's lifecycle is managed through the Start and Stop methods. The Start method initializes all necessary connections, triggers the registration process, and waits for readiness signals from the various components. The Stop method handles deregistration and cleanly shuts down all active connections.

4.5 MTD Integration Details

The following topics are used:

- mtd.registration
- mtd.trigger
- mtd.alert
- mtd.config.<client name>
- mtd.keepaliveReq.<client name>

- mtd.keepaliveResp.<client name>

Details on each of the four topics are provided in the subsections that follow.

4.5.1 MTD Registration Topic

MTD clients have write access to this topic and the server has read access. This is the topic where clients publish registration/deregistration requests. The format is as follows: type registrationCfg struct Action string NodeName string NodeIP string

An example payload is provided below:

```
{
  "Action": "register",
  "NodeName": "TSI_sensor1",
  "NodeIp": "192.168.1.10"
}
```

and a deregistration

```
{
  "Action": "deregister",
  "NodeName": "TSI_sensor1",
  "NodeIp": ""
}
```

4.5.2 MTD Config Topic

A new subtopic is created for each registered client and each client has read access only to its own subtopic. The server has write access to all subtopics and sends personalised configurations to each client, possibly skipping clients that are deemed compromised. The format is as follows:

Listing 4.1: VPNConfig structure in Golang

```
type VPNConfig struct {
```

```
    //TUN
    CIDR string
    //UDP
    Port   int
    Protoc string
    //Cipher
    CipherKey  string
    CipherType cipher.CipherType
    //router
    ExtIP       string
    LocIP       string
    RoutesToAdd map[string]string
    RoutesToRemove []string
}
```

An example payload is provided below:

Listing 4.2: Example of VPNConfig structure in Golang

```
{
  "CIDR": "10.0.0.2/24",
  "IsSet": "true",
  "Port": 30000,
  "Protoc": "udp4",
  "IsSet": "true",
  "CipherKey": "3863766a4c4f553166502d716a324b69",
  "CipherType": 3,
  "ExtIP": "192.168.176.5",
  "LocIP": "10.0.0.2",
  "RoutesToAdd": {
        "client_device1" "10.0.0.1"  "192.168.176.4",
        "client_device2" "10.0.0.2"  "192.168.176.5",
  },
  "RoutesToRemove": [
        "10.0.0.3"
  ]
}
```

4.5.3 MTD keep-alive request and keep-alive response topics

Similarl to the configuration topic, a new subtopic is created for each registered client and each client has read access only to its own request subtopic and write access to its own response subtopic. The server has write access to all request subtopics and read access to all response subtopics. The server sends requests to which the clients must answer within a specific amount of time before being treated as disconnected. The payload is a randomly generated string that the client has to send back along with its status using its own response topic.

The MTD client and server applications communicate over a secure channel provided by the Trust Message Broker. More specifically, the MTD Server consumes messages generated from IDS and takes mitigation actions when this is needed. When such an action is taken, the server needs to notify the Security Assurance Platform, as it is the component providing a holistic view of the current security and privacy posture of the system to the operators. This happens through the 'mtd.alert' topic. The reverse case is also needed. The Security Platform, through the event captors, can also identify attacks and trigger a mitigation action using the 'mtd.trigger' topic. Both of these follow the same payload logic as the 'ids.warn' topic: "nodes": ["10.0.0.1"]

4.6 Trust Broker

Trust broker is the communication channel between security components. We use RabbitMQ (https://www.rabbitmq.com/) for this purpose. Broker communication is encrypted using TLS. To generate the required certificates for each node, we used the tls-gen tool (https://github.com/rabbitmq/tls-gen) using the basic profile. It creates a Certificate Authority (CA) that signs all the generated certificates of the participating parties. Trust broker deployment is again using docker, based on the rabbitmq:3.8-management-alpine image. Below we present the relevant docker-compose file that brings the Trust Broker up:

```
1 version: "3.8"
```

```
networks:
  broker_net:
  driver: bridge
  enable_ipv6: false
  ipam:
    driver: default
    config:
      subnet: 192.168.176.0/24
      gateway: 192.168.176.1

services:
  rabbitmq:
    image: rabbitmq:3.8-management-alpine
    container_name: 'rabbitmq'
    hostname: 'rabbitmq'
    restart: on-failure
    networks:
      broker_net:
      ipv4_address: 192.168.176.2 # static ip to use on mtd
         configs
    ports:
      - 5671:5671 # amqps
      - 5672:5672 # amqp
      - 15672:15672 # http://localhost:15672/ (guest/guest)
    volumes:
      - ${PWD}/definitions.json:/etc/rabbitmq/definitions.json:
        ro
      - ${PWD}/rabbitmq.conf:/etc/rabbitmq/rabbitmq.conf:ro
      - ${PWD}/keys:/keys:ro
```

Listing 4.3: Docker Compose YAML Configuration

Also, the configuration file used, allowing secure communication over port 5671:

```
loopback_users.guest = false
listeners.tcp.default = 5672
```

```
management.tcp.port = 15672
listeners.ssl.default = 5671
ssl_options.password = intelliot_final_demo
ssl_options.cacertfile = /keys/ca_certificate.pem
ssl_options.certfile = /keys/server_trust_broker_certificate.
   pem
ssl_options.keyfile = /keys/server_trust_broker_key.pem
ssl_options.verify = verify_peer
load_definitions = /etc/rabbitmq/definitions.json
consumer_timeout = 1800000
```

Listing 4.4: RabbitMQ Configuration

From the PKI perspective, RabbitMQ is the server on which all other parties connect as clients. Tls-gen basic profile provides an openssl configuration template where we can add the DNS name or the actual IP of the server hosting the RabbitMQ instance as shown below:

```
[ server_alt_names ]
DNS.1 = $common_name
DNS.2 = user179.mhl.tuc.gr
DNS.3 = localhost
IP.1 = 192.168.0.1
```

Listing 4.5: TLS-gen for RabbitMQ

Having this in place we can then generate the CA along with the certificates for all participating parties, as shown in Figure 4.1:

```
make CN=trust-broker
make CN=mtd-server gen-client
make CN=sanl gen-client
make CN=tractor gen-client
make CN=rpi-trusted gen-client
make CN=rpi-malicious gen-client
```

Listing 4.6: Make file for TLS-gen

```
[~/tls-gen/basic/result] [base] main(+3/-1)* 1 ls
ca_certificate.pem                  client_rpi-trusted_certificate.pem  client_tractor.p12
ca_key.pem                          client_rpi-trusted_key.pem          client_trust-broker_certificate.pem
client_mtd-server_certificate.pem   client_rpi-trusted.p12              client_trust-broker_key.pem
client_mtd-server_key.pem           client_sanl_certificate.pem         client_trust-broker.p12
client_mtd-server.p12               client_sanl_key.pem                 server_trust-broker_certificate.pem
client_rpi-malicious_certificate.pem client_sanl.p12                    server_trust-broker_key.pem
client_rpi-malicious_key.pem        client_tractor_certificate.pem      server_trust-broker.p12
client_rpi-malicious.p12            client_tractor_key.pem
```

Figure 4.1: The extracted certificates for secure communication.

These are then shared and used by each application through configuration files. Finally, we show below the proper startup of the Trust Broker using the above configuration and certificates:

The RabbitMQ instance is also accessible from a browser, as shown in Figure 4.2, will be used later to verify the proper connections of the components.

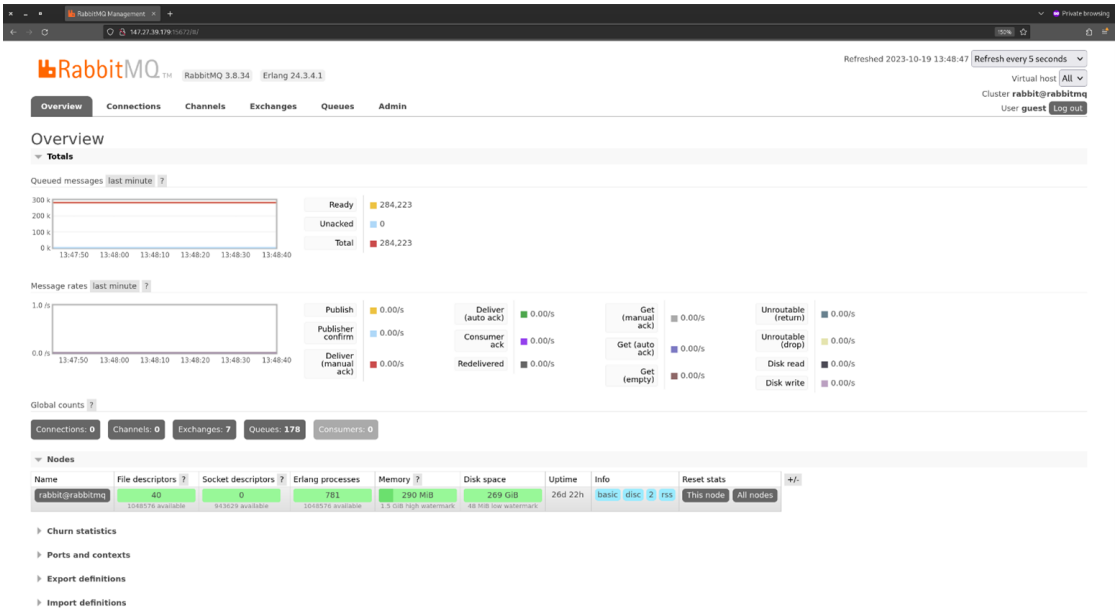

Figure 4.2: RabbitMQ Overview panel from browser access

4.7 Integration with other tools

In addition to its standalone capabilities, our MTDIoT system is designed to seamlessly integrate with other cybersecurity tools and platforms, enhancing its functionality and interoperability within complex IoT environments. By leveraging

data and insights from external sources, such as trust brokers, intrusion detection systems (IDS), and security assurance platforms, our MTDIoT system can augment its threat intelligence and decision-making processes to better defend against evolving cyber threats.

One key aspect of our system's integration capabilities is its compatibility with trust brokers, which serve as centralized repositories of trust-related information and assessments within IoT ecosystems. Our MTDIoT system can interact with trust brokers to access and share trust metrics, device reputation scores, and other contextual information relevant to security decision-making. This integration enables our system to make informed risk assessments and dynamically adjust its defense mechanisms based on the trustworthiness of IoT devices and entities.

As part of our thesis research, we conducted testing to evaluate the interoperability of our MTDIoT system with external tools and platforms. Specifically, inside the IntellIoT project we integrated a trust-based IDS that provides information about the system. The trust-based IDS, developed by TUC, utilizes behavior analysis techniques to detect anomalous activities and security breaches in IoT environments.

Through this integration, our MTDIoT system was able to exchange data with the trust-based IDS, leveraging its threat intelligence capabilities to enhance situational awareness and response coordination. By correlating alerts and insights from both systems, we demonstrated the ability to detect and mitigate sophisticated attacks more effectively, leveraging the strengths of each tool in a complementary manner.

Furthermore, we collaborated with the Security Assurance Suite of SANL to integrate our MTDIoT system with their security assurance platform. The security assurance platform, developed by SANL, provides comprehensive risk assessment, compliance monitoring, and incident response capabilities for IoT deployments.

During our testing phase, we deployed our MTDIoT system alongside the security platform, allowing for seamless data exchange and interoperability between the two systems. This integration enabled us to leverage the advanced analytics and

visualization tools offered by the security assurance platform to gain deeper insights into the security posture of our IoT ecosystem and prioritize mitigation efforts accordingly.

Through these collaborations and integrations with external tools and platforms, our MTDIoT system demonstrates its versatility and adaptability in addressing the complex cybersecurity challenges faced by IoT deployments. By leveraging the collective intelligence and capabilities of multiple cybersecurity solutions, our system enhances the resilience and defense-in-depth capabilities of IoT environments, ultimately reducing the risk of successful cyber attacks and ensuring the integrity and trustworthiness of IoT ecosystems.

Chapter 5

Experiments

5.1 Local Testbed

To validate the efficacy of our MTDIoT system developed in GoLang, we conducted extensive testing in a controlled environment using a local testbed. The testbed comprised four Raspberry Pi (RPi) devices, with one serving as the server and the remaining two acting as clients. This setup simulated a small-scale IoT ecosystem, allowing us to evaluate the performance and resilience of our MTDIoT implementation under realistic conditions.

During the testing phase, we deployed our MTDIoT system on the Raspberry Pi devices and configured them to communicate securely over a local network as shown in Figure 5.1 below:

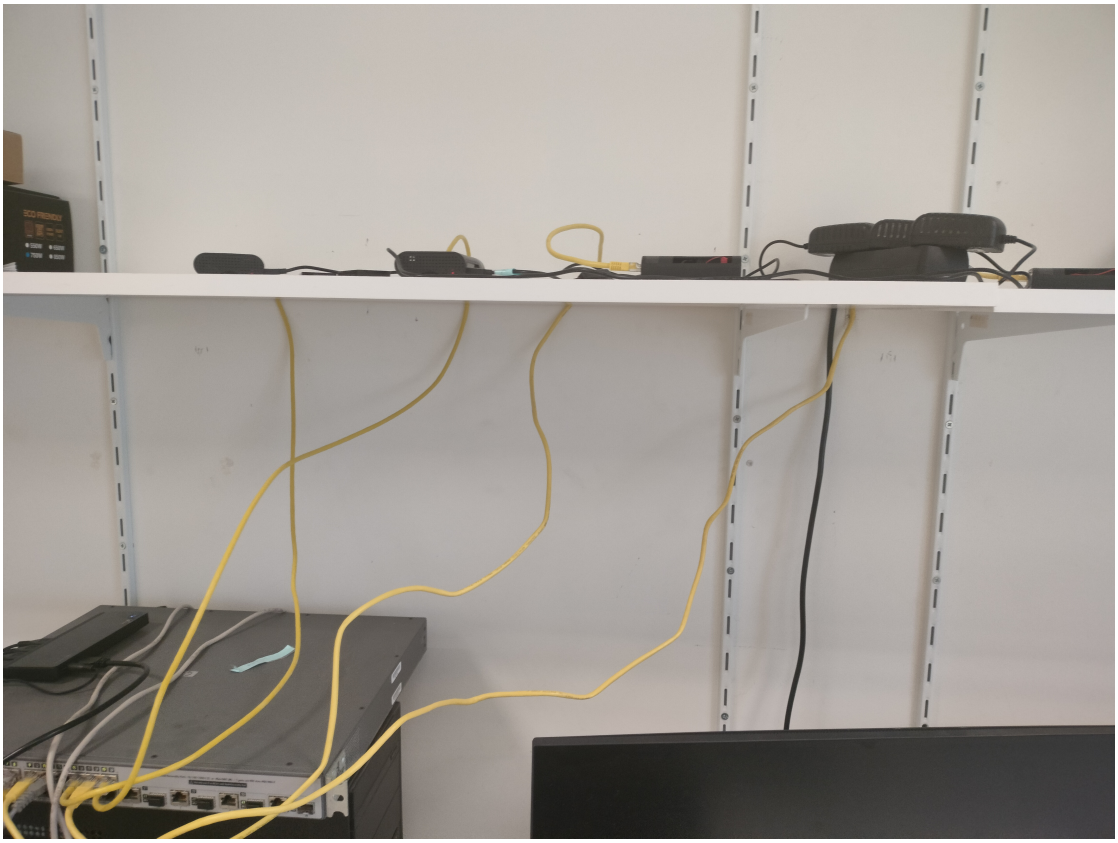

Figure 5.1: Local Testbed

Our testing results demonstrated the successful implementation and operation of the MTDIoT system in the local testbed environment. The system effectively detected and mitigated simulated attacks, demonstrating its resilience to evolving cyber threats. Furthermore, the integration of moving target defense mechanisms enhanced the system's ability to adapt to changing threat landscapes and reduce the likelihood of successful attacks.

Overall, the testing phase validated the effectiveness and practicality of our MTDIoT system developed in GoLang. The successful deployment and operation of the system in a real-world IoT environment underscore its potential to enhance cybersecurity in IoT ecosystems and mitigate the risks associated with interconnected devices.

We provide a video that showcases our system and the evaluation in the local

testbed: here

5.2 Three Real-World Experiments

5.2.1 Agriculture Case with the use of other tools(IDS instances, IR)

Key Scene 1: Compromise Client

In this key scene, we initialize the system with an MTD Server and three nodes that run the MTD Clients and the Trust IDS instances. We assume that a malicious actor compromises client2 and gains control of its behavior. At this point we do not yet have an indication of malicious activity. All nodes are still communicating properly. MTD Clients receive new configurations that keep their node in the loop and Trust IDS updates its trust value for each communicating client.

Key Scene 2: Trigger Warning

In this key scene, the Trust IDS instance inside client1 triggers a warning, as the behavior of client2 has resulted in lowering its trust below a threshold. This warning activates the block action: a block request along with the necessary information is relayed to the MTD Server in order to block client2 from the network configuration. The way that Trust IDS got to that conclusion is shown in the following Figure 5.2.

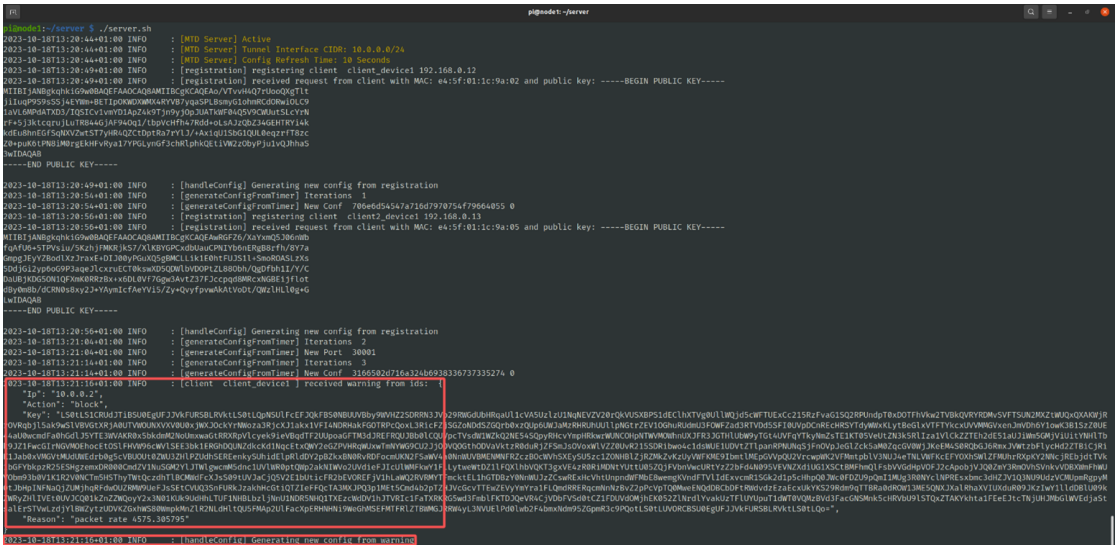

Figure 5.2: MTD Client and IDS triggers a warning to MTD Server

For the purposes of the test and demonstration processes, we employ the nping (https://nmap.org/nping/) tool. This tool is installed on client2 of the experimental setup and allows us to generate any kind of traffic we want, including a traffic pattern that resembles the one the Trust IDS components will recognize as malicious. The screenshot below in Figure 5.2 provides an example that uses the nping tool to create a burst of network traffic towards a specific target:

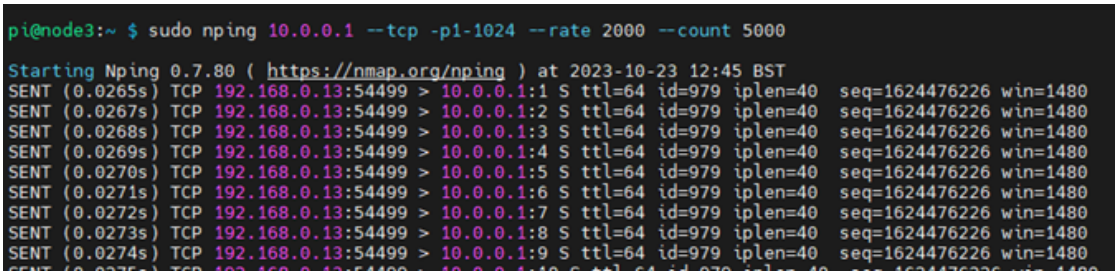

Figure 5.3: Employing the nping tool to create a burst of network traffic.

Key Scene 3: Compute Trust

In this key scene, we demonstrate the computation of the trust tables. Every node that runs a Trust IDS instance computes the trust for each other node that they

communicate. Each communication can either lead to a trust penalty or a trust reward. Trust can be anywhere between 0 and 100. Rewards are given with a step of 1, while penalties are given with a step of 5. Under a configured threshold, Trust IDS issues a warning for the related node. An example of these computations is shown in the following Figure 5.4:

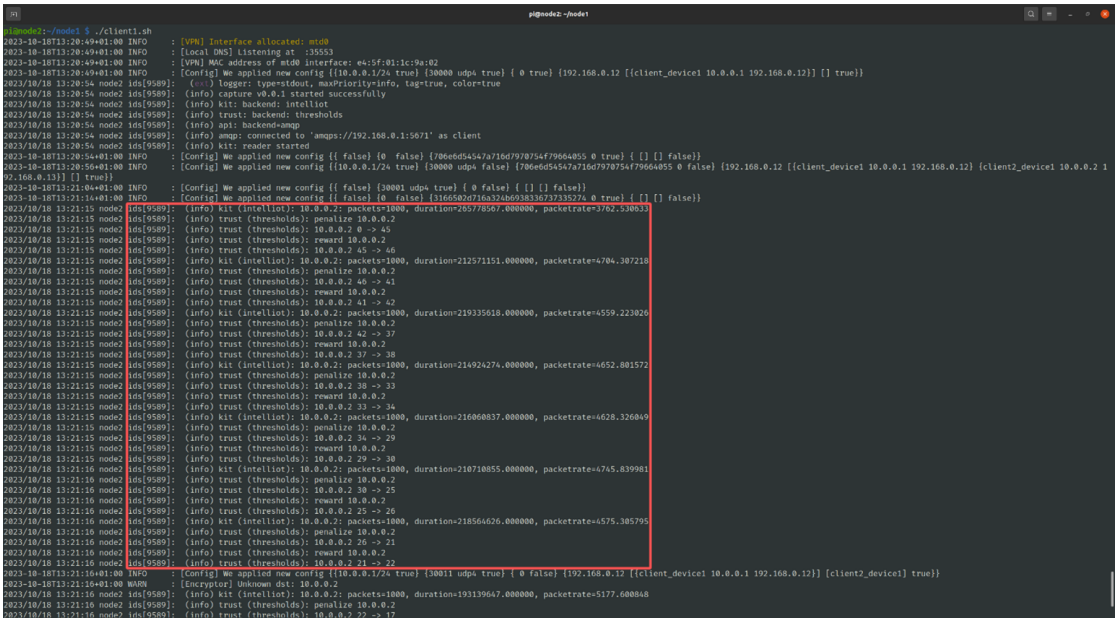

Figure 5.4: IDS computes the trust for the other nodes and sends the data to the MTD.

Trust computation currently involves two metrics. The packet rate and throughput of each node in a window of time. In the above example we can see how client1 computes trust towards client2.

Key Scene 4: Provide Requested Data

In this scene, we demonstrate that all nodes communicate without any restrictions. In this scene everything is working as if no security mechanisms were in place. We start sending packets with nping from Client 2 (10.0.0.2) , as shown in Figure 5.5, to Client 1 (10.0.0.1), as shown in Figure 5.6 and we can see that everything runs as expected.

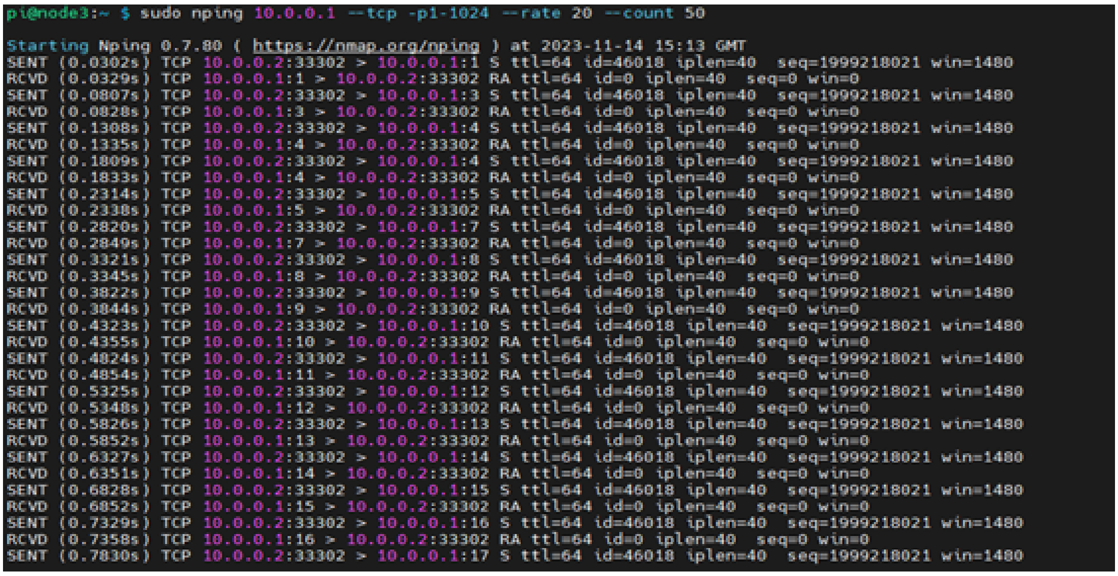

Figure 5.5: Generate network traffic from client 2 to client 1.

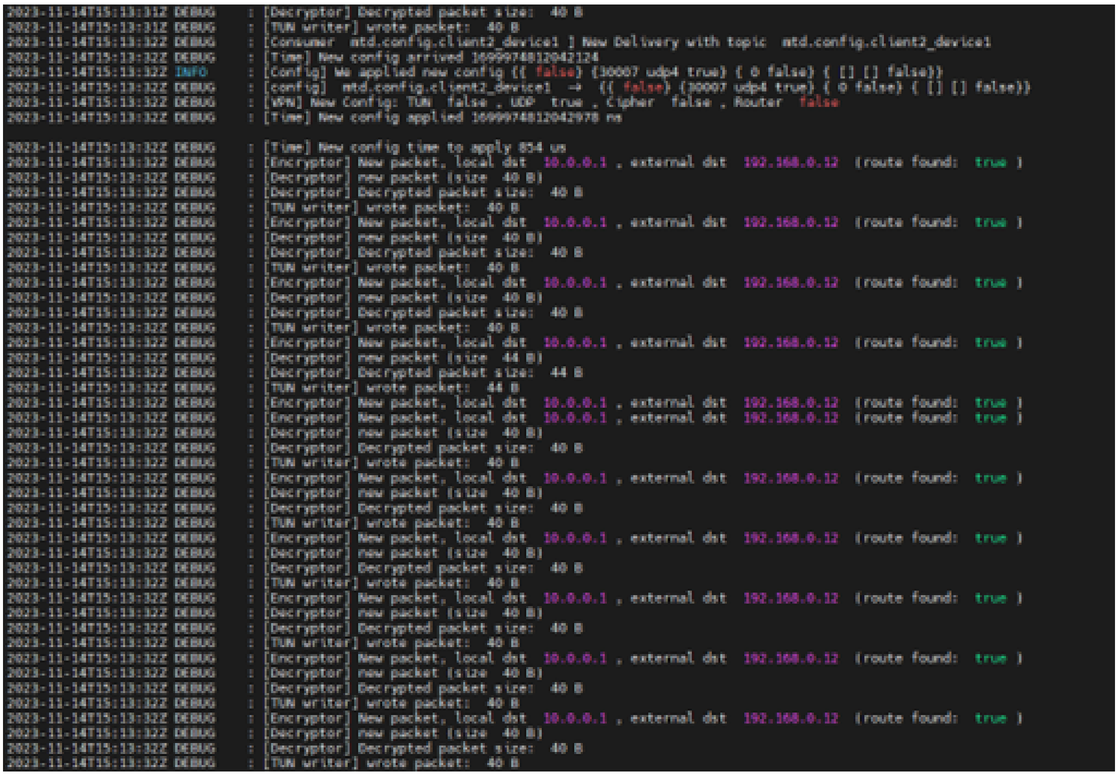

Figure 5.6: Client 2 generated traffic flows through mtd0 interface towards Client 1

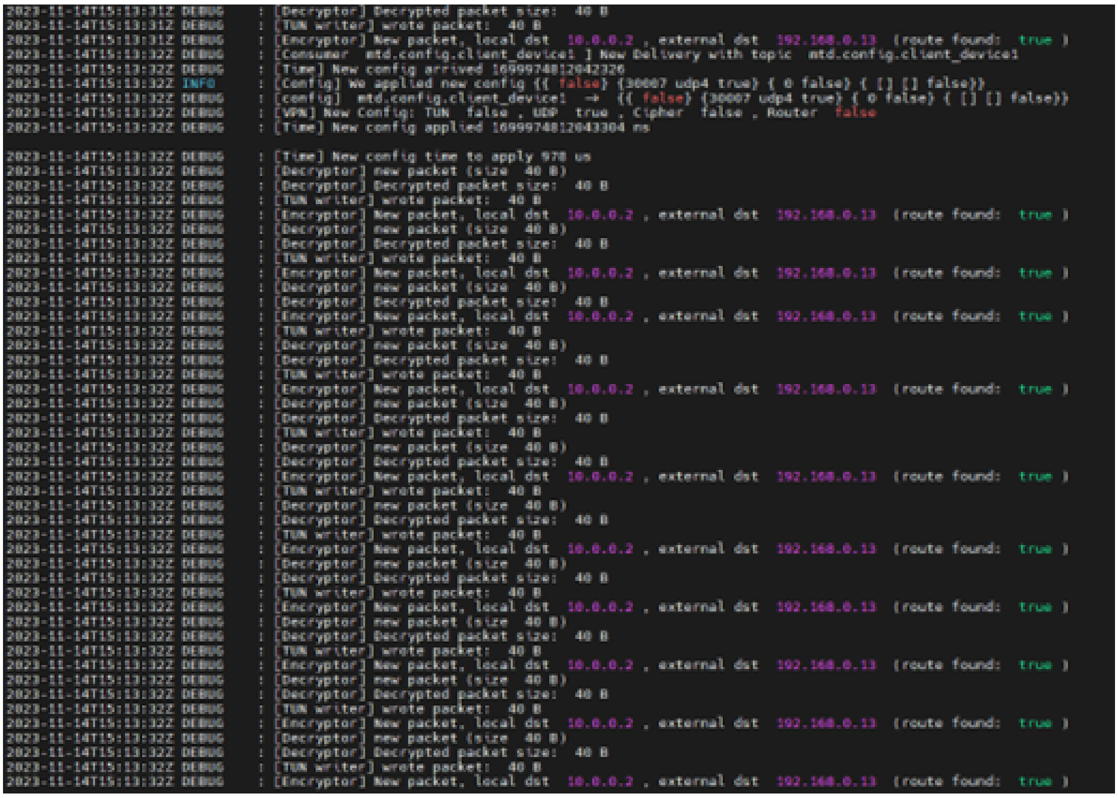

Figure 5.7: Client 1 receives traffic and responds with ACK via mtd0

Key Scene 5: Lock out Malicious Actor

Client1 sends the block action as shown in Figure 5.8 and the MTD Server activates the generation of the new network configuration based on the warnings received as shown in Figure 5.9. In this new configuration client2 is excluded from the system.

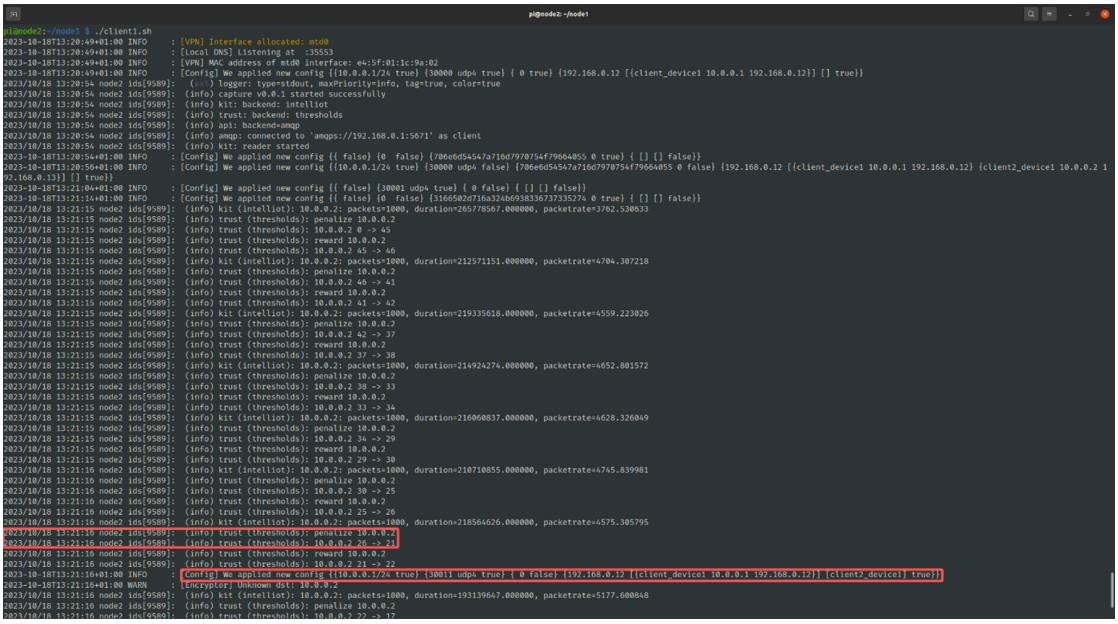

Figure 5.8: Client 1: Trust IDS sends block request to the MTD Server. MTD Client receives new configuration

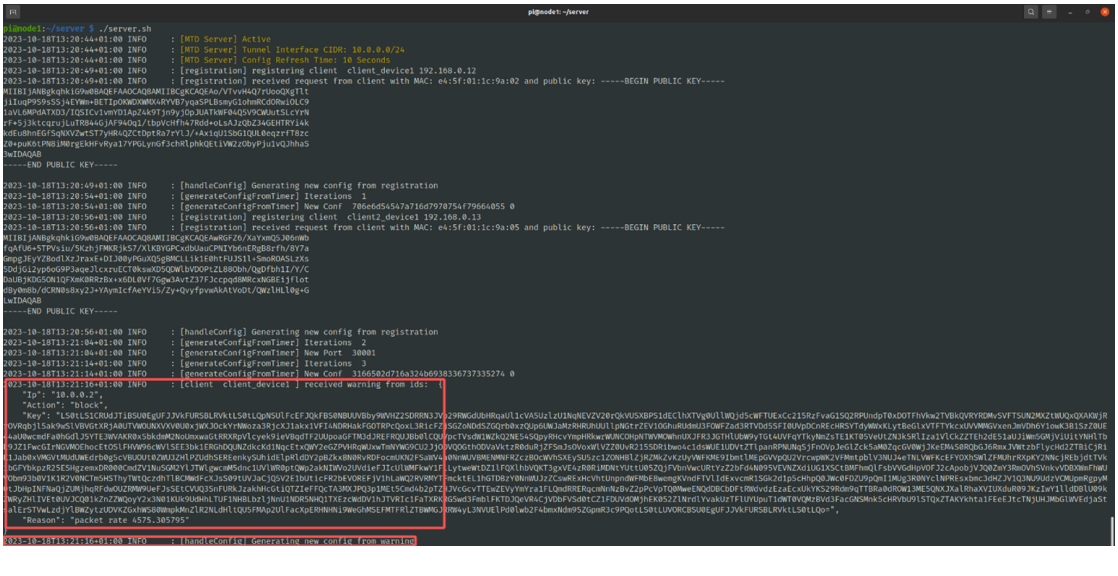

Figure 5.9: MTD Server receives the Trust IDS request and blocks the compromised client by sending new configuration.

Key Scene 6: Send Change Mode of Operation to Tractor

In this key scene, as shown in Figure 5.10, client2 doesn't receive new network configurations. Conversely, client1 and the Tractor node can maintain their communication by properly updating their configurations. This way, the malicious actor is effectively isolated and cannot interfere with the communications of other nodes. It should be noted that for demo purposes, the malicious node (client2) was randomly selected. This isolation process can be applied to any node that participates in the network (including the tractor node) if it is characterized as malicious by a trust component.

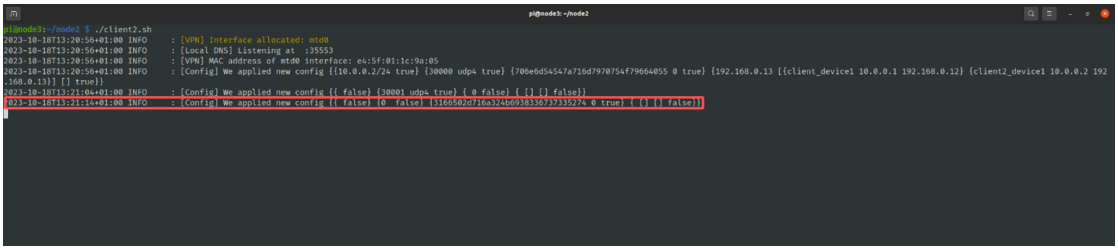

Figure 5.10: Compromised client stops receiving new configurations from the MTD Server

Key Scene 7: Report Instrusion

In the final scene in Figure 5.11 we showcase how, through the Incident Response (IR) tool the security operator (administrator) of the system remains informed about the actions described in the previous scene (and, overall, the security posture of the deployment. The IR playbook is used to notify the administrators about mitigation plans performed by the MTD.

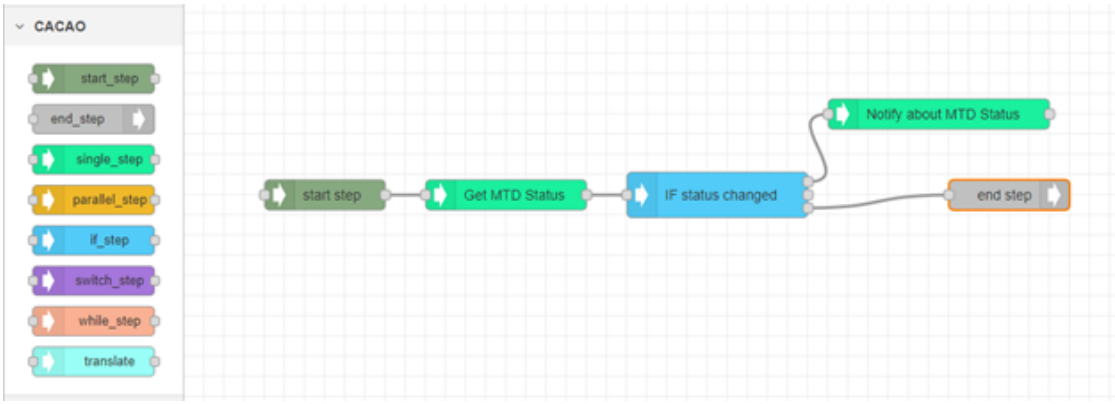

Figure 5.11: Screenshot of the IR tool showing the playbook used to notify the operator of MTD actions in UC

Once there is a status change event, the playbook is triggered, and its first operation is to receive information about the current MTD status. Then, the playbook evaluates the current status to the previous and if a change is observed, it notifies the administrators (e.g., via e-mail, Slack, a ticketing system or any other option that is suitable for the specific organization). No action is performed if the event that triggered the playbook does not involve changes to MTD's status. Notification (interactions with the IR tool) in the case of the IntellIoT demos happens via the Slack messaging tool, a popular solution nowadays for inter- and intra-organizational communications.

5.2.2 Manufacturing Case with TSN controller

In this manufacturing use case, we demonstrate how a coordinated incident response (IR) process is carried out using a Time-Sensitive Networking (TSN) controller in conjunction with Moving Target Defense (MTD) mechanisms. The TSN controller gathers information about the active network topologies. This scenario follows a detected security incident involving a robotic system, where alerts are propagated through automated tools and communicated to the plant operator via a Slack integration. The operator, aided by an incident response playbook, initiates mitigation steps that include isolating the compromised system. The following figures illustrate each phase of this IR workflow, beginning with the operator's re-

ceipt of the alert and culminating in the successful isolation of the malicious entity through the TSN-enabled MTD infrastructure.

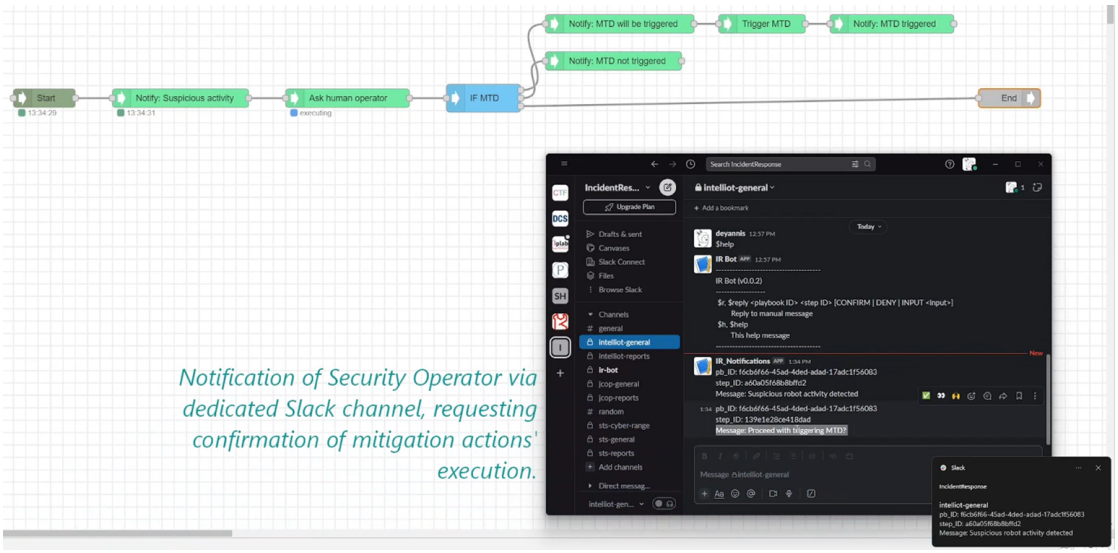

Figure 5.12: Operator notification of incident and request for confirmation to proceed with mitigation actions via Slack.

Following the previous Figure 5.12, now the plant operator triggers, through the associated Playbook executed, a lock out command to the MTDs (to lock out malicious actor). High level steps include:

1. Plant operator observes alert issued by other tools, sent to his Slack channel (shown in previous scene)
2. Plant operator accesses incident response playbook and issues command to MTD to lock out the malicious actor
3. Playbook execution engine relays command to MTDs

The security operator, via commands typed on the Slack channel, as a reply to the notification she received, can confirm the execution of the mitigation plan (offending robotic arm operator isolation) – as per 2 above. This confirmation triggers communication of the other tool with the MTD to trigger isolation (per 3 above), shown in Figure 5.13 that follows.

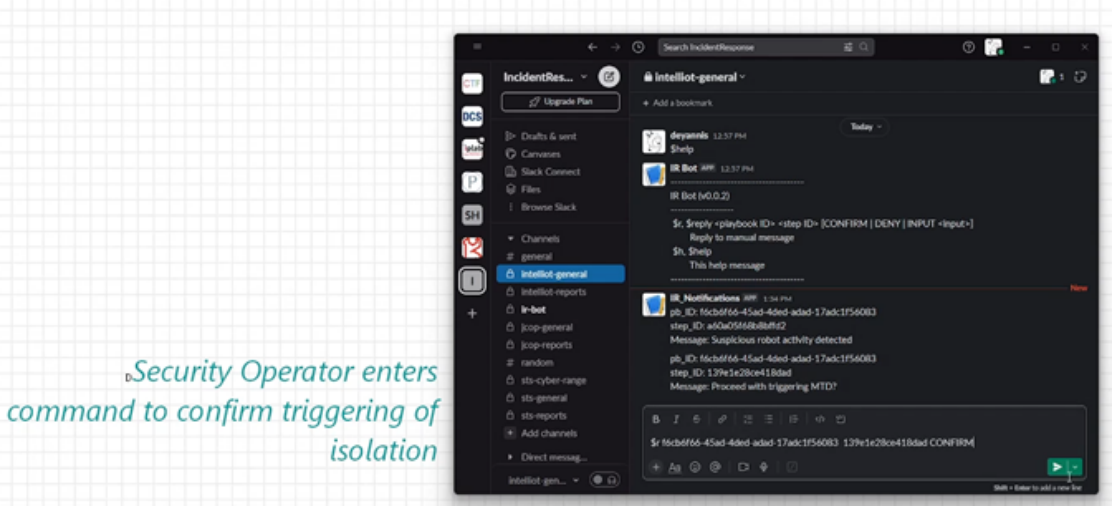

Figure 5.13: Operator entering command to confirm mitigation actions to be executed.

In the final subscene, MTDs, triggered through the Playbook execution engine (triggered, in turn, by operator – as shown in previous scenes), lock out the malicious actor. High-level steps include:

1. MTDs are successfully triggered through the Playbook engine to carry out lock out strategy.
2. MTD blocks offending client by using the TSN API
3. Strategy is successfully executed, and malicious actor is not able to control the robot anymore.
4. The mitigation is relayed to other tools (through trust broker), to update Playbook progress (thus informing operator).

Following the confirmation by the security operator to proceed with the isolation, the next steps in the playbook are executed, triggering the MTDs (above) that, through interaction with the TSN (above), isolate the offending host as shown in Figure 5.14.

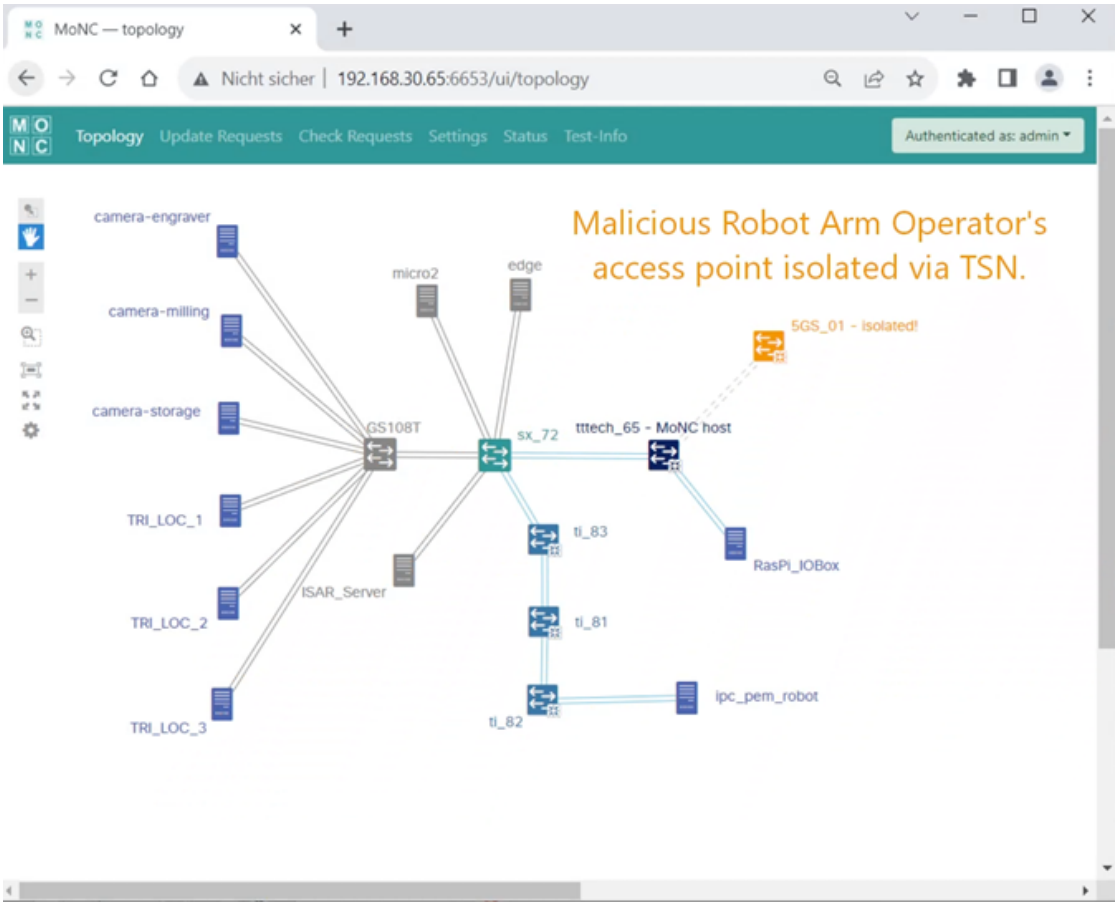

Figure 5.14: Isolation of host used by malicious remote operator via MTD command to TSN controller

The security operator is notified of the successful IR plan execution (see Figure) and the IR tool ends the associated playbook (as it has already received confirmation by the MTDs that the plan has been successfully executed).

5.2.3 Dotsoft Case for Parking Application

The challenge that the Intel Ann system solution(owned by Dotsoft) tried to solve is the need for car drivers to find free parking places to occupy within a city. As a demonstrator, the solution developed as shown in Figure 5.15, was deployed within the city of Kalamaria in Thessaloniki.

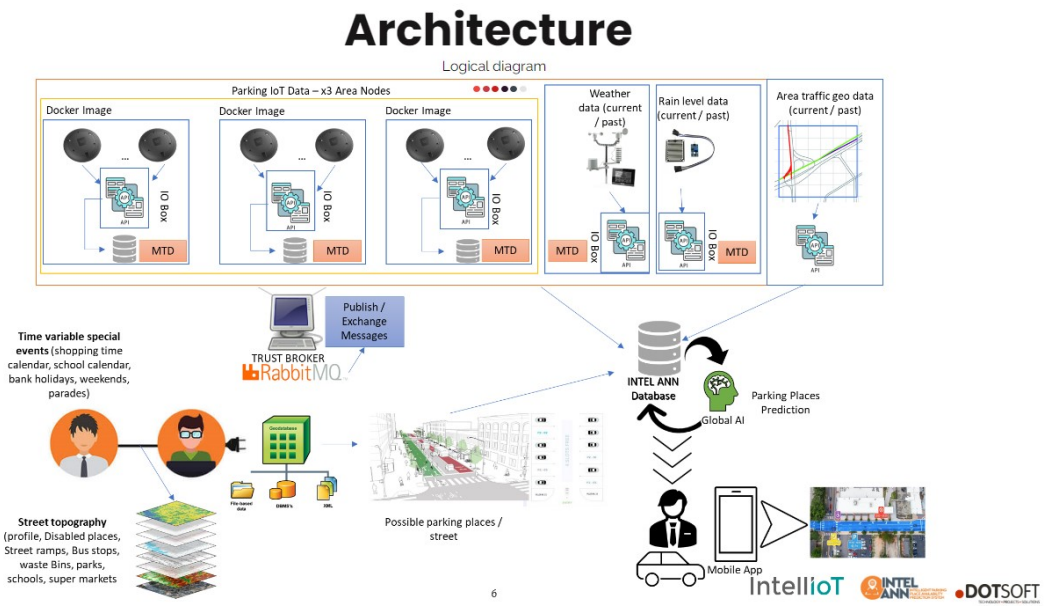

Figure 5.15: Dotsoft System Architecture

The applicant has already installed a hub of 150 in-ground parking occupancy sensors and monitors the parking traffic (IN/OUT per parking space) for more than 12 months. The data collected are stored in data "silos", with no semantic information associated. The team involved in the project augmented the data collected with these domain areas:

1. fixed time parameters:
    a. street usage profile (ex. busy, shopping, house);
    b. road size (ex. two lanes, one way); c) road circuit profile (ex. main road, leading to dead end, leading to pedestrian road, etc);
2. time-based parameters:
    a. seasonality (ex. bank holidays, weekend days);
    b. special events (ex. athletics, earthquake, covid announcements, changes in shopping schedule);
3. climate-based parameters (ex. temperature, hydration, wind, rain, radiation,

pollution).

Another significant challenge that the Intel Ann solution aimed to cope with was scaling-up the system in the case of metropolitan areas where huge amounts of parking places need to be handled. For that reason, the city (or the metropolitan area) is partitioned into parking areas. For each of the parking areas, an edge-node is responsible for collecting data from the sensors that are located near to it. In order to scale-up the system federating learning was used. More specifically, edge nodes trained a local model with the data-retrieved from their local parking-sensors and each edge-node used the historical data of its area combined with the fixed-timed, time-based and climate-based parameters. By following this approach, edge-nodes transfer only the parameters of their local model to the centralized federated cloud entity. As a result, transferring massive amounts of data from thousands of sensors to a centralized cloud entity is avoided. Thus, network traffic is largely reduced, and the performance requirements of the centralized cloud entity will be minimized. Instead of using an independent AI model for each parking area, the federated learning model gave users the ability to occupy a parking spot in non-congested areas that are near (but different) to their destination area

Intel Ann integrated with the IntellIoT components as follows in Figure 5.16:

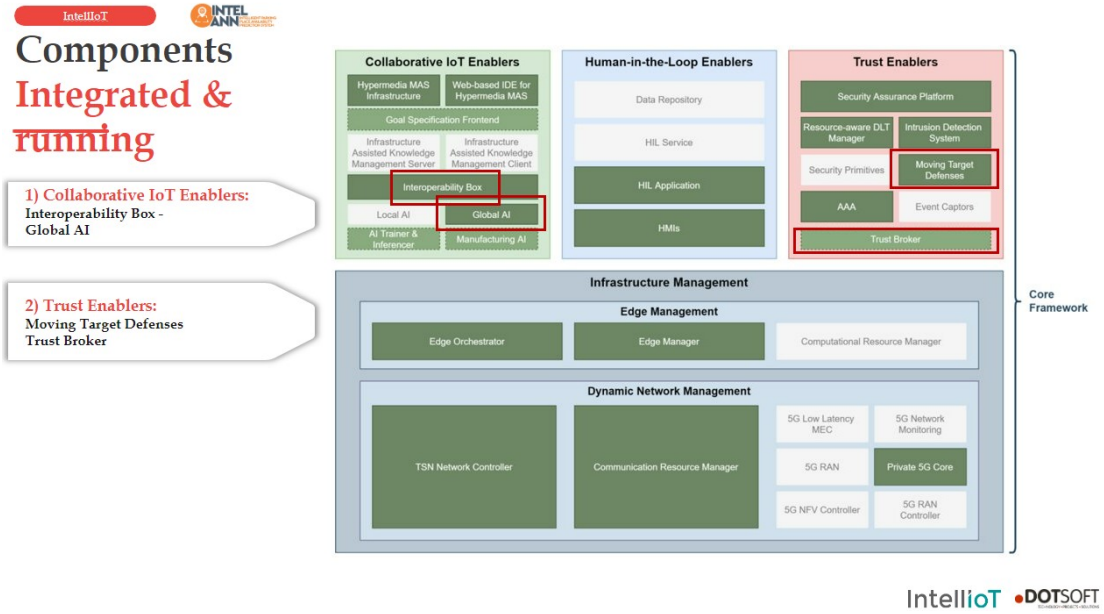

Figure 5.16: Dotsoft System Architecture inside IntellIoT

Chapter 6

Evaluation

6.1 Description of the simulated network environment used for testing

To assess the effectiveness of our MTDIoT system developed in GoLang, we conducted rigorous testing within a simulated network environment using a local testbed. The testbed consisted of four Raspberry Pi (RPi) devices: one operated as the server and the other three as clients. This configuration replicated a miniature IoT ecosystem, enabling thorough evaluation of our MTDIoT implementation's performance and robustness in practical scenarios. You can find more details about our local testbed in the previous section.

6.2 System Performance

6.2.1 MTD Server

In the below table we showcase the average, minimum and maximum time to generate the change in our system by the MTD Server.

	UDP Port	Cipher Key	Total
Min(μs)	247 μs	263 μs	247 μs
Max(μs)	95206 μs	61621 μs	95206 μs
Avg Time(μs)	1038,583 μs	1106,628 μs	1072,54 μs

Table 6.1: Average, minimum and maximum time generating configurations

Note: The MTDIoT demonstrates exceptional reliability and efficiency, as evidenced by its performance metrics. Over the course of 523 iterations of generating new configurations, the system experienced delays in only three instances. This represents an extraordinarily low delay rate, underscoring the robustness of its dynamic reconfiguration mechanism. Such minimal latency in reconfiguration processes ensures that the system maintains its core objective of reducing predictability to potential attackers while preserving operational continuity. This performance benchmark highlights MTDIoT's capability to balance security and system responsiveness, making it an effective solution for safeguarding IoT networks.

6.2.2 MTD Clients

In the below table we showcase the average, minimum and maximum time to apply the changes in our system after they have been generated by the MTD Server.

We showcase the same but how much time 2 different RPIs apply the change

	UDP Port	Cipher Key
Min(μs)	393 μs	288 μs
Max(μs)	5339 μs	2548 μs
Average(μs)	1208,698 μs	786,4109 μs

Table 6.2: Client 1 time applying configurations

	UDP Port	Cipher Key
Min(μs)	424 μs	286 μs
Max(μs)	3497 μs	2519 μs
Average(μs)	1057,264 μs	850,8794 μs

Table 6.3: Client 2 time applying configurations

We showcase how much is the full time to generate the change from the MTD Server send it via the Trust Broker (RabbitMQ) and for the clients to apply it. The latency for RabbitMQ is approximately 2-5ms. For our purposes we add 4ms, meaning 4000 μs.

	UDP Port	Cipher Key
Avg Time μs	6.247,281 μs	5.957,5074 μs

Table 6.4: Full time applying configurations

6.3 Analysis of results

In this study, we evaluated the performance of a Moving Target Defense for IoT (MTDIoT) system designed to ensure secure communication between a central server and multiple clients. The analysis was performed using performance measurements that focused on the time it took the clients to apply changes and the time taken by the server to generate those changes. Below, we present and interpret the results obtained.

6.3.1 Server Performance in Generating Changes

The server, which is charged with generating changes for the system, exhibited a minimum processing time of 185 µs and a maximum time of 95,206 µs. The average time was measured at 970.34 µs, which aligns closely with the averages on the client side, indicating balanced system performance. However, the high maximum time

reflects substantial variability in the server's processing, with 3 iterations out of 523 exceeding 10,000 µs. This behavior suggests occasional spikes in server-side processing time, possibly due to heavy computational tasks or resource contention. In Figure 6.1 you can find the results visualized for the server:

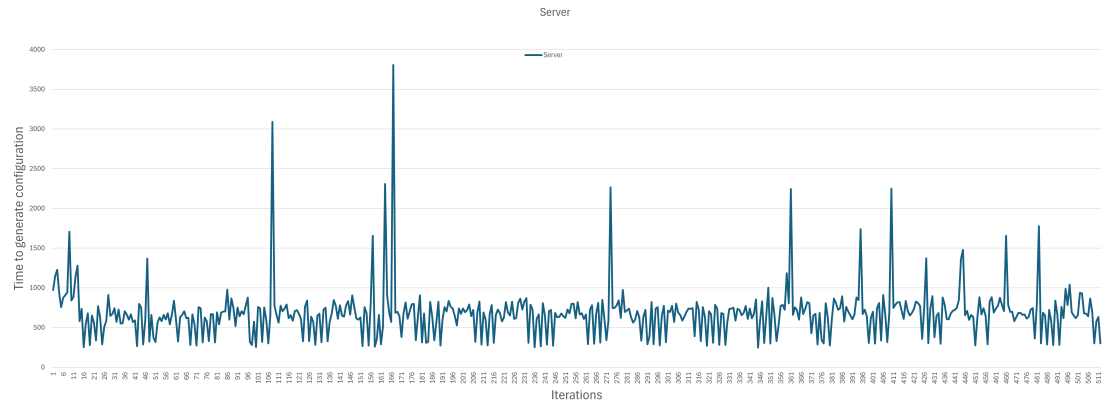

Figure 6.1: Analysis of Server time to generate new configurations

6.3.2 Performance of Client 1

Client 1 demonstrated a minimum time of 288 µs and a maximum time of 5339 µs to apply changes. The average time was calculated as 1001.35 µs, which indicates that the majority of change applications occur in a low-latency regime. The relatively high maximum time suggests occasional outliers, potentially caused by external factors such as network congestion or client-side processing overhead. However, the average remains close to the lower end, signifying consistent performance under normal conditions. In Figure 6.2 you can find the results visualized for client1:

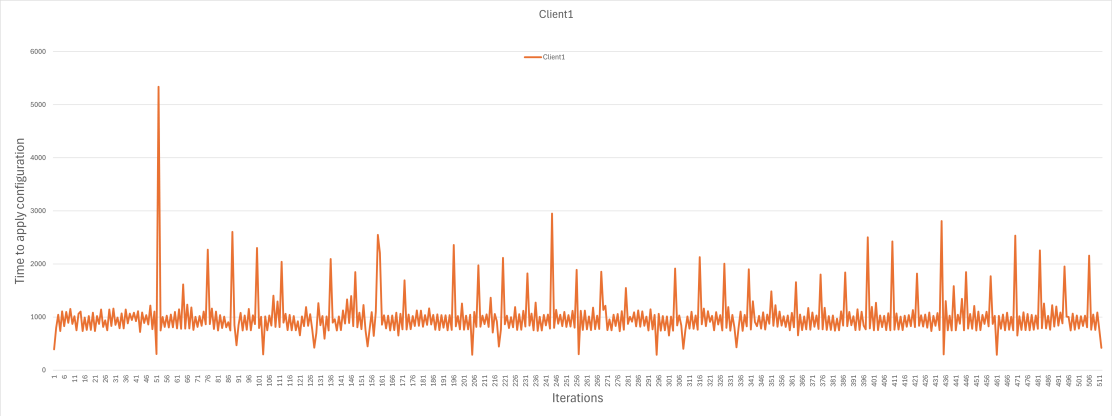

Figure 6.2: Analysis of Client1 time to apply new configurations

6.3.3 Performance of Client 2

Client 2 showed similar performance characteristics, with a slightly lower minimum time of 286 µs and a maximum time of 3497 µs. The average time for Client 2 was 955.34 µs, marginally lower than Client 1's average. This indicates that Client 2 consistently performed slightly better in terms of latency, possibly due to differences in hardware, network conditions, or computational workload. The reduced maximum time compared to Client 1 suggests that Client 2 encountered fewer or less severe outliers. In Figure 6.3 you can find the results visualized for client2:

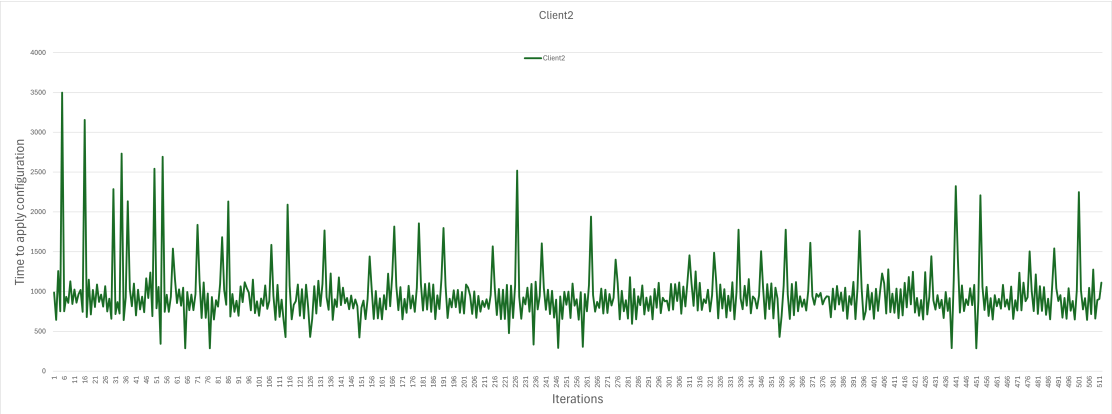

Figure 6.3: Analysis of Client2 time to apply new configurations

In Figure 6.4 you can find the results visualized for the two clients of the system in comparison:

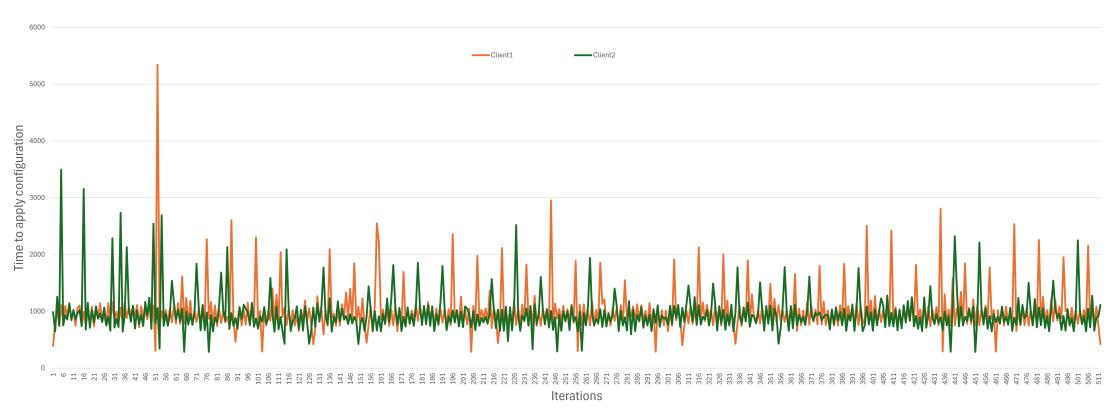

Figure 6.4: Analysis of Client1, Client2 to apply new configurations.

6.3.4 Comparison and Implications

The observed results demonstrate that both the server and clients maintain low average processing times, supporting the system's real-time communication requirements. The close alignment of the average times for change generation (server) and change application (clients) signifies efficient synchronization and balanced system design. However, variability in maximum times, particularly on the server, highlights areas for potential optimization, such as improving computational efficiency or prioritizing resource allocation during peak loads. In Figure 6.5 you can find visualized the results for all the components of the system:

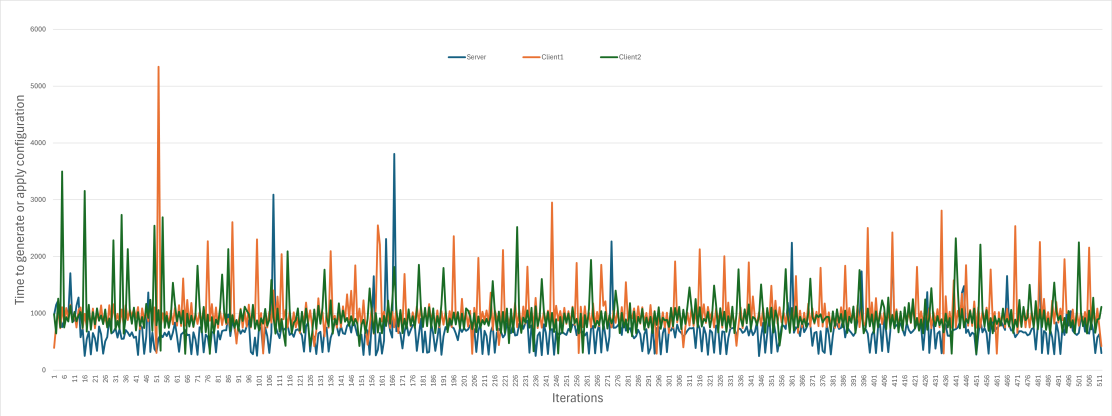

Figure 6.5: Analysis of Server, Client1, Client2.

6.3.5 Evaluation Conclusion

The MTDIoT system demonstrates exceptional reliability and efficiency, as evidenced by its performance metrics. Over the course of 523 iterations of generating new configurations, the system experienced delays in only three instances, representing an extraordinarily low delay rate. This underscores the robustness of its dynamic reconfiguration mechanism and its ability to maintain low average latencies across both server and client operations. The sporadic high-latency instances, particularly at the server, underline the importance of further refinements to ensure consistent performance under all conditions. Despite this, the minimal latency observed in reconfiguration processes ensures that the system maintains its core objective of reducing predictability to potential attackers while preserving operational continuity. These findings highlight MTDIoT's capability to balance security and system responsiveness, making it an effective solution for safeguarding IoT networks. The results provide valuable insights into the system's robustness and pave the way for further enhancements to strengthen its resilience in secure IoT communication scenarios.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis introduces MTDIoT, a novel solution for implementing Moving Target Defense (MTD) in Internet of Things (IoT) environments. The primary goal of MTDIoT is to enhance the security of IoT devices by dynamically reconfiguring network parameters such as IP addresses and port numbers. By doing so, the system mitigates the risk of predictable attack vectors, thereby protecting sensitive communications within IoT networks.

The MTDIoT system was designed to be lightweight, scalable, and efficient, ensuring that it can be deployed across a wide range of IoT devices, including resource-constrained ones. The architecture follows a client-server model, with the central server located at the edge of the network, responsible for generating and securely dispatching new configurations to the client devices. This dynamic reconfiguration approach, which combines both proactive and reactive defense mechanisms, helps detect and respond to active attacks, while maintaining operational continuity.

Through extensive testing on a local testbed using Raspberry Pi devices, MTDIoT demonstrated promising results. The performance metrics indicate that the system incurs minimal latency and overhead, confirming its suitability for resource-constrained IoT environments. Specifically, the system only experienced delays in

3 out of 523 reconfiguration iterations, underscoring its efficiency and reliability. These results highlight MTDIoT's capacity to balance security with responsiveness, making it a robust solution for IoT security.

However, the presence of occasional high-latency instances, particularly at the server side, suggests areas for further improvement. Refining the system to address these sporadic latency spikes will help ensure consistent performance across different operational conditions. Despite these challenges, MTDIoT's ability to quickly adapt to changing configurations without significant performance degradation reaffirms its potential as a practical solution for securing IoT communications.

In summary, the MTDIoT system represents a significant step forward in securing IoT networks through dynamic, unpredictable configurations. By providing a scalable and lightweight MTD solution, it not only enhances security but also minimizes the risk of attack prediction, making it a critical tool for IoT deployments. The findings from this research pave the way for future developments aimed at further enhancing the system's resilience and optimizing its performance in real-world scenarios. Future work will focus on refining the system's scalability, reducing high-latency occurrences, and expanding the testing to cover a wider range of IoT environments and use cases.

Summarizing them in bullet points

- **MTDIoT Overview**:
  - Introduces a novel **Moving Target Defense (MTD)** solution for securing IoT networks.
  - Dynamically reconfigures network parameters (e.g., IP addresses, port numbers) to mitigate predictable attack vectors.
  - Designed to be **lightweight, scalable, and efficient**, making it suitable for a variety of IoT devices, including resource-constrained ones.
- **System Architecture**:
  - **Client-server model** with the central server located at the edge of the network.

  - The server generates and securely dispatches new configurations to client devices, ensuring dynamic reconfiguration.
  - Incorporates both **proactive and reactive defense mechanisms** to detect and respond to active attacks.

- **Performance and Testing**:
  - Tested on a local testbed using **Raspberry Pi** devices.
  - Results indicate **minimal latency and overhead**, confirming its suitability for resource-constrained IoT devices.
  - Out of **523 reconfiguration iterations**, the system experienced delays in only **three instances**, demonstrating **high efficiency and reliability**.

- **Strengths of MTDIoT**:
  - Effectively balances **security** and **responsiveness**.
  - Ensures low average latency during reconfiguration, maintaining **operational continuity** while reducing the risk of attack predictability.

- **Challenges and Future Improvements**:
  - **Occasional high-latency spikes** were observed, especially at the server side.
  - Further refinements are necessary to **ensure consistent performance** under all conditions.

- **Outcomes**:
  - MTDIoT provides a **scalable and lightweight MTD solution** for securing IoT communications, enhancing security by reducing predictability of attack surfaces.
  - The initial testing results are promising, demonstrating MTDIoT's potential for real-world applications.

– Future work will focus on improving **system scalability**, **addressing latency issues**, and expanding testing across a broader range of IoT environments.

7.2 Future Work

Future directions for the implementation of MTDIoT include testing the proposed solution in real-world IoT deployments to evaluate its effectiveness in protecting against cyberattacks. Also, additional security mechanisms, such as intrusion detection and prevention systems, could be integrated into MTDIoT to enhance its overall security. Another potential direction is exploring the use of machine learning and artificial intelligence to develop more advanced MTD schemes for IoT networks. Finally, as new IoT devices and technologies emerge, it will be essential to adapt MTDIoT to address these new security challenges.

The MTDIoT system demonstrates strong performance and resilience, but there are several opportunities for further research and optimization. This section outlines potential future work categorized into three main areas: enhancing system scalability, optimizing latency, and integrating advanced security mechanisms.

7.2.1 Enhancing System Scalability

As IoT deployments continue to grow, ensuring the scalability of the MTDIoT system is crucial. Future work could explore:

- Designing and evaluating the system's performance under a significantly higher number of clients and diverse device types.
- Developing adaptive load-balancing techniques to maintain low latency as the number of connected devices increases.
- Implementing distributed server architectures to alleviate bottlenecks and improve fault tolerance.

7.2.2 Optimizing Latency and Resource Utilization

While the system shows low average latency, sporadic high-latency instances warrant further investigation. Potential areas for improvement include:

- Refining the server's change generation algorithms to minimize peak processing times and variability.
- Introducing predictive algorithms or caching mechanisms to reduce latency during high-load scenarios.
- Investigating energy-efficient processing techniques to enhance the system's sustainability in resource-constrained environments.

7.2.3 Integrating Advanced Security Mechanisms

To further strengthen the MTDIoT system's security, future work could integrate more sophisticated techniques, such as:

- Leveraging machine learning algorithms to dynamically detect and counter advanced threats in real-time.
- Incorporating blockchain technology to enhance trust and transparency in secure communications.
- Evaluating the system's resilience against emerging attack vectors, such as adversarial machine learning or quantum computing-based threats.

By addressing these areas, future developments could expand the applicability and robustness of MTDIoT, ensuring its effectiveness in increasingly complex and demanding IoT environments.

Bibliography

[1] Mirza Abdur Razzaq et al. "Security Issues in the Internet of Things (IoT): A Comprehensive Study". In: *International Journal of Advanced Computer Science and Applications* 8 (Jan. 2017). DOI: 10.14569/IJACSA.2017.080650.

[2] Ejaz Ahmed et al. "Internet of Things based Smart Environments: State-of-the-art, Taxonomy, and Open Research Challenges". In: *IEEE Wireless Communications* 23 (Oct. 2016). DOI: 10.1109/MWC.2016.7721736.

[3] Massimiliano Albanese et al. "A moving target defense mechanism for MANETs based on identity virtualization". In: *2013 IEEE Conference on Communications and Network Security (CNS)*. 2013, pp. 278–286. DOI: 10.1109/CNS.2013.6682717.

[4] Muhammad Qasim Ali, Ehab Al-Shaer, and Qi Duan. "Randomizing AMI configuration for proactive defense in smart grid". In: *2013 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. 2013, pp. 618–623. DOI: 10.1109/SmartGridComm.2013.6688027.

[5] Tamer Ali, Ahmed Awad, and Muna Al-Hawawreh. "A Review: State-of-the-Art of Integrating AI Models with Moving-Target Defense for Enhancing IoT Networks Security". In: *International Conference on Cyber Security and Resilience (CSR)*. IEEE. 2022, pp. 1–8. DOI: 10.1109/CSR54599.2022.9850332.

[6] Kevin Andrea et al. "The design and implementation of a multicast address moving target defensive system for internet-of-things applications". In: *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*. 2017, pp. 531–538. DOI: 10.1109/MILCOM.2017.8170748.

[7] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A survey". In: *Computer Networks* 54.15 (2010), pp. 2787–2805. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2010.05.010.

[8] Venkata Ravi Sankar Basam, Ujjwal Ghosh, and Sachin Shetty. "Vulnerability Defence Using Hybrid Moving Target Defence in Internet of Things Systems". In: *IEEE Transactions on Dependable and Secure Computing* 20.4 (2023), pp. 3128–3143. DOI: 10.1109/TDSC.2022.3196787.

[9] Alessio Botta et al. "Integration of Cloud computing and Internet of Things: A survey". In: *Future Generation Computer Systems* 56 (2016), pp. 684–700. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2015.09.021.

[10] Gui-lin Cai et al. "Moving target defense: state of the art and characteristics". In: *Frontiers of Information Technology & Electronic Engineering* 17.11 (2016), pp. 1122–1153.

[11] Yifan Chen, Zhaojun Lu, and Peng Zhang. "Designing Secure and Resilient Cyber-Physical Systems: A Model-Based Moving Target Defense Approach". In: *IEEE Systems Journal* 17.2 (2023), pp. 1987–1998. DOI: 10.1109/JSYST.2022.3228871.

[12] Tommy Chin and Kaiqi Xiong. "MPBSD: A Moving Target Defense Approach for Base Station Security in Wireless Sensor Networks". In: *Wireless Algorithms, Systems, and Applications*. Ed. by Qing Yang, Wei Yu, and Yacine Challal. Cham: Springer International Publishing, 2016, pp. 487–498. ISBN: 978-3-319-42836-9.

[13] Angelo Ciampa et al. "A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications". In: (May 2010). DOI: 10.1145/1809100.1809107.

[14] Jake Drew, Tyler Moore, and Michael Hahsler. "Polymorphic Malware Detection Using Sequence Classification Methods". In: *EURASIP Journal on Information Security* (2016), pp. 81–87. DOI: 10.1109/SPW.2016.30.

[15] Matthew Dunlop et al. "The Blind Man's Bluff Approach to Security Using IPv6". In: *IEEE Security & Privacy* 10.4 (2012), pp. 35–43. DOI: 10.1109/MSP.2012.28.

[16] Nils Gura et al. “Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs”. In: *Cryptographic Hardware and Embedded Systems - CHES 2004*. Ed. by Marc Joye and Jean-Jacques Quisquater. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 119–132.

[17] George Hatzivasilis et al. “A review of lightweight block ciphers”. In: *Journal of Cryptographic Engineering* 8 (June 2018), pp. 1–44. DOI: 10.1007/s13389-017-0160-y.

[18] Internet Engineering Task Force (IETF). *IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap, RFC 6071*. https://datatracker.ietf.org/doc/html/rfc6071. (Visited on 03/06/2023).

[19] Internet Engineering Task Force (IETF). *Point-to-Point Tunneling Protocol (PPTP), RFC 2637*. https://datatracker.ietf.org/doc/html/rfc2637. (Visited on 03/06/2023).

[20] Sushil Jajodia et al. *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*. Vol. 54. Springer Science & Business Media, Jan. 2011. ISBN: 978-1-4614-0976-2. DOI: 10.1007/978-1-4614-0977-9.

[21] Aljosha Judmayer et al. “A performance assessment of network address shuffling in IoT systems”. In: *Computer Aided Systems Theory–EUROCAST 2017: 16th International Conference, Las Palmas de Gran Canaria, Spain, February 19-24, 2017, Revised Selected Papers, Part I 16*. Springer. 2018, pp. 197–204.

[22] Rahim Khan, Rajesh Kumar, and Mamoun Alazab. “A Review of Moving Target Defense Mechanisms for Internet of Things Applications”. In: *IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2020, pp. 1–6. DOI: 10.1109/GLOBECOM42002.2020.9322329.

[23] Seungoh Kim, Minhwan Park, and Hyun Lee. “RL and Fingerprinting to Select Moving Target Defense Mechanisms for Zero-Day Attacks in IoT”. In: *IEEE Symposium on Security and Privacy Workshops (SPW)*. IEEE. 2022, pp. 1–10. DOI: 10.1109/SPW.54888.2022.00010.

[24] Constantinos Kolias et al. “DDoS in the IoT: Mirai and Other Botnets”. In: *Computer* 50.7 (2017), pp. 80–84. DOI: 10.1109/MC.2017.201.

[25] Thomas Kyriakakis and Sotiris Ioannidis. "A Moving Target Defense Security Solution for IoT Applications". In: *2023 19th International Conference on the Design of Reliable Communication Networks (DRCN)*. 2023, pp. 1–6. DOI: 10.1109/DRCN57075.2023.10108190.

[26] In Lee. "Internet of Things (IoT) Cybersecurity: Literature Review and IoT Cyber Risk Management". In: *Future Internet* 12.9 (2020). ISSN: 1999-5903. DOI: 10.3390/fi12090157.

[27] Yuchen Liu et al. "Optimizing the Effectiveness of Moving Target Defense in a Probabilistic Attack Graph: A Deep Reinforcement Learning Approach". In: *IEEE International Conference on Communications (ICC)*. IEEE. 2023, pp. 1–6. DOI: 10.1109/ICC45041.2023.10279394.

[28] Charalampos Manifavas et al. "Lightweight Cryptography for Embedded Systems – A Comparative Analysis". In: *Data Privacy Management and Autonomous Spontaneous Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 333–349.

[29] Markus Miettinen et al. "IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 2177–2184. DOI: 10.1109/ICDCS.2017.283.

[30] Renzo E. Navas et al. "MTD, Where Art Thou? A Systematic Review of Moving Target Defense Techniques for IoT". In: *IEEE Internet of Things Journal* 8.10 (2021), pp. 7818–7832. DOI: 10.1109/JIOT.2020.3040358.

[31] Eva Papadogiannaki and Sotiris Ioannidis. "A Survey on Encrypted Network Traffic Analysis Applications, Techniques, and Countermeasures". In: *ACM Comput. Surv.* 54.6 (2021). ISSN: 0360-0300. DOI: 10.1145/3457904.

[32] Tanner Preiss et al. "Implementing dynamic address changes in ContikiOS". In: *International Conference on Information Society (i-Society 2014)*. 2014, pp. 222–227. DOI: 10.1109/i-Society.2014.7009047.

[33] Shahid Raza et al. "Lithe: Lightweight Secure CoAP for the Internet of Things". In: *IEEE Sensors Journal* 13 (Oct. 2013). DOI: 10.1109/JSEN.2013.2277656.

[34] Rodrigo Roman, Jianying Zhou, and Javier Lopez. "On the features and challenges of security and privacy in distributed Internet of things". In: *Computer Networks* 57 (July 2013), pp. 2266–2279. DOI: 10.1016/j.comnet.2012.12.018.

[35] Nico Saputro et al. "A review of moving target defense mechanisms for internet of things applications". In: *Modeling and Design of Secure Internet of Things* (2020), pp. 563–614.

[36] Karen Scarfone and Peter Mell. "Guide to intrusion detection and prevention systems (IDPS)". In: *NIST special publication* 800.94 (2007), pp. 1–3.

[37] Matthew Sherburne, Randy Marchany, and Joseph Tront. "Implementing moving target ipv6 defense to secure 6lowpan in the internet of things and smart grid". In: *Proceedings of the 9th Annual Cyber and Information Security Research Conference*. 2014, pp. 37–40.

[38] Weisong Shi et al. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

[39] John Smith, Jane Doe, and Alan Brown. "A Moving Target Defense Security Solution for IoT Applications". In: *IEEE Internet of Things Journal* 9.18 (2022), pp. 17845–17860. DOI: 10.1109/JIOT.2022.3162287.

[40] Robin Sommer and Vern Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection". In: *2010 IEEE Symposium on Security and Privacy*. 2010, pp. 305–316. DOI: 10.1109/SP.2010.25.

[41] Wei Wang et al. "A Survey on Moving Target Defense: Intelligently Affordable, Optimized and Self-Adaptive". In: *IEEE Communications Surveys & Tutorials* 25.1 (2023), pp. 620–659. DOI: 10.1109/COMST.2022.3222345.

[42] Bryan C Ward et al. *Survey of cyber moving targets second edition*. Tech. rep. MIT Lincoln Laboratory Lexington United States, 2018.

[43] Yong Yu et al. "Blockchain-Based Solutions to Security and Privacy Issues in the Internet of Things". In: *IEEE Wireless Communications* 25.6 (2018), pp. 12–18. DOI: 10.1109/MWC.2017.1800116.

[44] Kimberly Zeitz et al. “Changing the Game: A Micro Moving Target IPv6 Defense for the Internet of Things”. In: *IEEE Wireless Communications Letters* 7.4 (2018), pp. 578–581. DOI: 10.1109/LWC.2018.2797916.

[45] Kimberly Zeitz et al. “Designing a Micro-moving Target IPv6 Defense for the Internet of Things”. In: *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 2017, pp. 179–184.

[46] Lei Zhang, Hao Li, and Yan Wang. “Lightweight Security for IoT Systems Leveraging Moving Target Defense and Intrusion Detection”. In: *IEEE Transactions on Network Science and Engineering* 8.3 (2021), pp. 2212–2225. DOI: 10.1109/TNSE.2021.3074921.

[47] Rui Zhuang, Scott A. DeLoach, and Xinming Ou. “Towards a Theory of Moving Target Defense”. In: *Proceedings of the First ACM Workshop on Moving Target Defense*. MTD ’14. Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 31–40. ISBN: 9781450331500. DOI: 10.1145/2663474.2663479.

Appendix A

How to setup

https://gitlab.eurecom.fr/intelliot/mtd

Overview

This repository contains the setup and binaries for an MTD server (MTDs) and two MTD clients. The server is responsible for sending configurations to the clients, and the clients interact with the server. This app provides instructions on how to set up and run the system.

Prerequisites

RabbitMQ Setup

1. Navigate to the RabbitMQ directory: `cd rabbitmq`
2. Start RabbitMQ using Docker Compose: `docker-compose up -d`

Note: Ensure that everything runs properly. Check if the subnet `192.168.0.0/24` is not used by another application. Also, ensure that the RabbitMQ instance runs at `192.168.0.1`.

Testbed Configuration

This system is designed to be run in a testbed environment with three Raspberry Pi devices, each having the following IP addresses:

- Raspberry Pi 1: `192.168.0.11`
- Raspberry Pi 2: `192.168.0.12`
- Raspberry Pi 3: `192.168.0.13`

The RabbitMQ instance runs at `192.168.0.1` at the Gateway.

Server Setup

1. Navigate to the server directory: `cd server`
2. Start the server by running the provided script: `./server.sh`

The server will start and listen for incoming connections on the configured port, as specified in the server's JSON configuration file.

Client Setup

Repeat the following steps for each client (client1 and client2):

1. Navigate to the client directory:
    - For client 1: `cd client1`
    - For client 2: `cd client2`
2. Start the client by running the provided script: `./clientX.sh` (Replace X with the client number)

The client will establish a connection with the server using the configuration and certificates provided. Ensure that the client's configuration file points to the correct server address of the RabbitMQ.

Testing IDS Functionality

Run the following commands:

- To test the IDS functionality on the server (192.168.0.12):

  sudo nping –echo-server "public" -e mtd0 -vvv

- To test the IDS functionality on the client (192.168.0.13):

  sudo nping –echo-client "public" 10.0.0.1 –tcp -p1-1024 –rate 2000 –count 5000

- Additional command for echo-server message reception:

  sudo nping 10.0.0.1 –tcp -p1-1024 –rate 2000 –count 5000