University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Federated Reinforcement Learning for Private and Collaborative Selection of Moving Target Defense Mechanisms for IoT Device Security

Jan Kreischer
Zurich, Switzerland
Student ID: 18-764-571

Supervisor: Dr. Alberto Huertas, Jan von der Assen
Date of Submission: June 01, 2023

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

The Internet of Things (IoT) has grown exponentially in recent years and it is predicted that the number of devices will double again to 30 billion by 2030 [24]. At the same time, the number of unpatched, vulnerable and infected devices connected to the Internet is increasing exponentially as well. Famous malware incidents from the past like Mirai have painfully illustrated how vulnerable IoT devices are on a broad scale. This work examines how Moving Target Defense (MTD) can be used in a collaborative framework for defense in depth and to thwart cyberattacks. For this purpose, a system prototype has been implemented that is capable of autonomously learning to defend a set of IoT devices (more specifically Radio Frequency Spectrum Sensors belonging to ElectroSense) from a specific set of malware by selecting and deploying Moving Target Defenses (MTDs). In scientific literature, usually individual MTDs optimized against specific attacks are presented, but no collaborative framework that combines and orchestrates a set of MTDs.

In the prototypical implementation, an individual local agent is deployed on a set of simulated device, monitoring the behavior of its host, according to 100 system parameters. In case an attack is detected, the local agent is invoked in order to select from a set of MTD to ward off the attack. If the post-MTD device behavior can be considered normal again, the local agent receives a reward, which is used to update the local policy. Thanks to the use of FL, all local agents contribute to learning one global defense policy together.

The project shows that a good attack mitigation probability can be achieved in non-federated as well as federated learning setting. Furthermore, the system also proves to be somewhat robust against locally and globally skewed sample distribution. Under certain assumptions it can also be assumed that collaborative learning of an MTD selection policy is faster and more robust than centralized learning. The findings on how FRL can be used in IT security to collaboratively learn an MTD selection policy contribute to the state of the art on MTD.

Zusammenfassung

Das Internet der Dinge ist in den letzten Jahren exponentiell gewachsen, und es wurde prognostiziert, dass sich die Anzahl der Geräte bis 2030 nochmal auf 30 Milliarden verdoppeln wird [24]. Dieses rasante Wachstum sorgt jedoch gleichzeitig für einen exponentiellen Anstieg an anfälligen und infizierten Geräten, die mit dem Internet verbunden sind. Berühmte Malware-Angriffe aus der jüngeren Vergangenheit wie Mirai haben schmerzhaft verdeutlicht, wie anfällig IoT-Geräte für Cyberangriffe sind. Diese Arbeit untersucht, wie Moving Target Defense (MTD) kollaborativ für Defense-in-Depth (DiD) und zur Abwehr von Cyberangriffen eingesetzt werden kann. Zu diesem Zweck wurde ein prototypisches System entwickelt, das in der Lage ist, autonom zu lernen, IoT Geräte (genauer gesagt Hochfrequenz-Spektrum-Sensoren die zum ElectroSense Netzwerk gehören) durch die geschickte Auswahl und den Einsatz von Moving Target Defenses (MTDs) vor bestimmten Schadsoftwares zu verteidigen. In der wissenschaftlichen Literatur werden meistens einzelne MTDs, die gegen spezifische Angriffe optimiert sind, präsentiert. Ein kollaboratives Framework, das eine Reihe von MTDs kombiniert und orchestriert, fehlt hier noch.

In der prototypischen Implementierung existiert jeweils ein lokaler Agent pro simuliertem IoT Gerät, der wiederrum das Verhalten seines Hosts anhand von 100 Systemparametern überwacht. Sollte ein Angriff detektiert werden, so wird der lokale Agent aufgerufen, um ein MTD aus der gegebenen Menge an MTDs auszuwählen, um den Angriff abzuwehren. Abhängig davon, ob das Geräteverhalten nach der Ausführung der MTD wieder als normal angesehen werden kann, erhält der lokale Agent ein Feedback Signal, welches genutzt wird, um die lokale Policy zu aktualisieren. Durch den Einsatz von Federated Learning tragen alle lokalen Agenten zum gemeinsamen Lernen einer globalen Policy bei.

Es zeigt sich, dass es möglich ist zu lernen, die gegebenen Angriffe mit hoher Wahrscheinlichkeit abzuwehren. Dies funktioniert sowohl gut zentralisiert als auch kollaborativ. Darüber hinaus erweist sich das System auch gewissermassen robust gegenüber einer lokal und global unbalancierten Klassenverteilung. Unter bestimmten Annahmen kann auch davon ausgegangen werden, dass kollaboratives Lernen einer MTD-Policy schneller und robuster ist als zentralisiertes Lernen derselbigen. Die Erkenntnisse, wie FRL in der IT-Sicherheit eingesetzt werden kann, um gemeinsam eine MTD-Policy zu Erlernen, erweitert den aktuellen Stand der Forschung.

Acknowledgments

First of all, I want to thank the Communication Systems Research Group (CSG) at UZH for enabling and hosting such interesting research projects. Throughout several modules, seminars, and projects during my master's, I got to know most members of the CSG and was able to work together with some of them. The work atmosphere was always professional and demanding, but also always supportive and humane. I don't take this for granted and I would like to thank all CSG members for their effort.

Jordan Cedeño provided us with the MTDs that were used in order to mitigate the different cyberattacks. Furthermore, this work also pays large tribute to Timo Schenk, who helped lay the foundation for the FedRL system with his work on single agent DQNs for MTD selection and with the training data he collected.

Special thanks to my main supervisor Dr. Alberto Huertas, who shaped this project through his previous research and always engaged in fruitful discussions. Pedro Miguel Sánchez and Chao Feng always helped to review the interim progress during our bi-weekly meetings and accelerated the progress by contributing with their professional knowledge and highly appreciated technical advice. I would also like to thank Jan van der Assen for promoting and co-supervising this project and always pointing to interesting scientific literature. Further, the support from Gérôme Bovet and CYD Armasuisse, who allowed me to use their facilities at the Cyber Defense Campus in Zurich for my work, cannot be overestimated. Many thanks also to Nina Ahrendt, who helped me a lot with proofreading and linguistic revision.

Special thanks also go to Prof. Dr. Burkhard Stiller, who always pushes forward on complex, brand new, highly relevant topics together with his research group. Overall, it was a very interesting and challenging six-month project, into which a lot of time, thought and energy was invested. But as the saying goes, you grow with the challenges you are facing. Therefore, I am positively excited to finally complete my studies after so many years with this thesis, and I am equally excited to face future challenges.

Summary of Notation

This work uses the mathematical notation by Sutton and Barto, 2014 [60]. The Capital letters are used for random variables and major algorithm variables. Lower case letters are used for the values of random variables and for scalar functions. Quantities that are required to be real-valued vectors are written in bold and lower case letters (even if they are random variables).

s	state
a	action
S	set of all non-terminal states
S^+	set of all states, including the terminal state
$A(s)$	set of actions possible in state s
R	set of possible rewards
t	discrete time step
T	final time step of an episode
S_t	state at t
A_t	action at t
R_t	reward at t, depending on A_{t-1} and S_{t-1}
G_t	return (cumulative discounted reward) following t
$G_t^{(n)}$	n-step return
G_t^{λ}	λ-return
π	policy, decision-making rule
$\pi(s)$	action taken in state s under deterministic policy π
$\pi(a\|s)$	probability of taking action a in state s under stochastic policy π
$p(s^{'}, r\|s, a)$	probability of transitioning to state $s^{'}$, with reward r, from s, a
$v_\pi(s)$	value of state s under policy π (expected return)
$v_*(s)$	value of state s under the optimal policy
$q_\pi(s, a)$	value of taking action a in state s under policy π
$q_*(s, a)$	value of taking action a in state s under the optimal policy
$V_t(s)$	estimate (a random variable) of $v_\pi(s)$ or $v_*(s)$
$Q_t(s, a)$	estimate (a random variable) of $q_\pi(s, a)$ or $q_*(s, a)$

$\hat{v}(s, \mathbf{w})$	approximate value of state s given a vector of weights $\mathbf{w}$
$\hat{q}(s, a, \mathbf{w})$	approximate value of state–action pair s, a given weights $\mathbf{w}$
$\mathbf{w}$, $\mathbf{w}_t$	vector of weights underlying an approximate value function
$\mathbf{x}(s)$	vector of features visible when in state s
$\mathbf{w}^\top \mathbf{x}$	inner product of vectors, $\mathbf{w}^\top \mathbf{x} = \Sigma_i w_i x_i$
σ_t	temporal-difference error at t
$E_t(s)$	eligibility trace for state s at t
$E_t(s, a)$	eligibility trace for a state–action pair
$\mathbf{e}_t$	eligibility trace vector at t
γ	discount-rate parameter
ϵ	probability of random action in ϵ-greedy policy
α, β	step-size parameters
γ	decay-rate parameter for eligibility traces

Contents

Chapter 1

Introduction

1.1 Motivation

The exponential growth of the Internet of Things (IoT) during the recent years is expected to continue in the future. In turn, the number of unpatched, vulnerable and infected devices connected to the Internet increases constantly. The economic value that is generated by the Internet of Things (IoT) is high, making it an attractive target for illegal cyber activities. The devastating effects of specific cyber attacks in recent years underpin this thesis and show that the problem still needs to be adequately addressed. Ransomware like WannaCry has been able to spread and infect devices in a rapid fashion and cause massive economic and social damage [70], [26]. According to a statistical study, the majority of cyber attacks remained undetected for and extended period of time and were only recognized after they have caused significant damages to the IT systems [53]. Since human operators cannot always react quickly enough to the spreading cyber infection, they can often only clean up the incurred damage, once networks or systems have become infected [53]. Traditional IT security systems are often susceptible to attacks based on yet undisclosed vulnerabilities, better known as zero day exploits [22]. Therefore, more dynamic and intelligent approaches, which also include defense-in-depth, are necessary [1]. This work proposes a novel approach using a combination of Federated Learning (FL) and Reinforcement Learning (RL) in order to collaboratively learn to select Moving Target Defense (MTD)s in order to counter malware attacks. Finding an optimal mapping between the high-dimensional state space and the set of possible counter actions is difficult and brute force fails here. RL is a more intelligent strategy able to learn in an online fashion, potentially allowing to react and mitigate even newly emerging attacks. In order to protect the behavioral data of each device, the selection policy is learned in federated fashion. Collaborative learning might enable to globally defend against an attack after having it seen only locally. In addition to the lack of scientific work on the use of MTDs in the context of the IoT, there is also still too little knowledge about the combination and joint use of several MTD techniques [9]. This work extends the state of the art by creating a framework combining and orchestrating multiple MTDs in order to provide protection against a set of malware.

1.2 Targets and Goals

Firstly, in a research-oriented step this work assesses the current state of IoT-security, -malware and cyber attacks on IoT. Furthermore, the current state of the art of Federated Learning (FL), Reinforcement Learning (RL) and Moving Target Defense (MTD) is gathered and forms the theoretical foundation for this work. The goal is to build a system that is capable of autonomously learning to defend a set of IoT devices (more specifically Radio Frequency Spectrum Sensors belonging to ElectroSense) from a specific set of malware by selecting and deploying MTDs.

The overarching idea is to have an individual agent deployed on a set of homogeneous client devices. Each local agent monitors the behavior of its host device and can select from a set of MTD when an attack is detected. The detection of an attack is based on the monitoring of 100 system parameters that provide information about the current status of the system. A local state anomaly detector, which is also running on each host device, continuously checks whether the current state of the device deviates significantly from the normal state. If this is the case, it is assumed that the host device is being attacked by malware and the local RL agent is invoked to select one of the MTDs to ward off the attack. After the MTD has been executed, the local state anomaly detection must classify again whether the post-state of the host device can be regarded as normal again and whether the selected MTD was successful. Depending on the success of the defense mechanisms, the local agent receives a positive or negative reward, which will be used to update the local policy. Thanks to the use of FL, all local agents contribute to learning one global defense policy together. The performance in homogeneous vs. heterogeneous client and federated vs. non-federated learning settings is evaluated, analyzed and compared.

The performance in federated vs non-federated learning settings as well as for different sample distributions, including local and global class (im)balances, was examined. The results show that a good attack mitigation probability can be achieved. Furthermore, the system also proves to be somewhat robust against locally and globally skewed class distribution. Under certain assumptions it can also be assumed that collaborative learning of an MTD selection policy is faster and more robust than centralized learning. The findings on how FRL can be used in IT security to collaboratively learn an MTD selection policy contribute to the state of the art on MTD.

1.3 Thesis Outline & Description of Work

Chapter 1 establishes the importance of IT security in the context of IoT and sets the goals and expectations of this work in relation to the current state of the art.

Chapter 2 reviews the current state of IoT and how it is influenced by different economic, technical and social factors. The potential negative effects of insufficient IoT security are illustrated using examples of several large malware attacks from the past. Furthermore, the necessary theoretical foundation, which is required to understand subsequent chapters, is established. The moving target defense principle is explained along with the attacks and mitigating MTDs featured in this work. The knowledge required to understand the used RL and FL algorithms such as DQNs and FedAvg is also documented here.

Chapter 3 provides an overview of related scientific work. A special focus is placed on works dealing with RL-based deployment of MTDs or works combining FL with RL. This provides an insight into the current state of the art and helps to identify potential difficulties in the combination of FL, RL with MTD early on.

Chapter 4 explains how the training data was generated and collected. Furthermore, two metrics to quantify global and local class balance are defined. The five different sample distribution scenarios used for the experiments are introduced. The overview of all collected features is followed by an explanation of how the training data was pre-processed. It is explained how outliers are handled, which features were selected for training and how the training data was scaled.

In Chapter 5, the three different prototypes at the heart of this project are explained. For each of the three different prototypes, the architecture and implementation are explained in detail. Additionally, the state anomaly detector used to generate the reward signal for the RL agent is described. Multiple FL experiments based on the various sample distributions from Chapter 4 follow. The explanation and assessment of the seen results conclude the chapter.

Chapter 6 summarizes the results of this work and concludes by discussing the limitations. Based on the final project outcome, potential future work is outlined. The project schedule that has been followed, is displayed by the Gantt chart in the Appendix B of this work.

Chapter 2

Background

The IoT can be seen as a large disseminated set of heterogenous physical devices, vehicles, buildings connected to the Internet as illustrated by Figure 2.1. The IoT is a major growth factor and seen as enabling technology for Industry 4.0 [31]. According to forecasts, the number of Internet connected devices will increase from 13 billion (in 2022) to 29 billion (in 2029) [25]. This technology is increasingly used in security critical domains like medical science (e.g continuous glucose monitoring), industrial applications and critical infrastructures. Maintaining privacy and mitigating security threats are core IoT issues that need to be adressed [58]. Recent malware attacks like Mirai have shown how vulnerable IoT devices can be, even to unsophisticated attacks brute-forcing default passwords [31].

Figure 2.1: Schematic Visualization of the Internet of Things [11]

This chapter is intended to lay the necessary theoretical foundation required to understand the following work. First of all, it provides an overview of the current IoT landscape and its security vulnerabilities to adversarial actors. Then, the most common types of malware affecting IoT devices and specific malware attacks from the past, including countermeasures, will be discussed. This is followed by an explanation of how Moving Target Defense (MTD) can be used as a countermeasure fostering defense-in-depth. Timo Schenk has shown that it is possible to use Reinforcement Learning (RL) to learn a policy to select MTDs in order to mitigate detected malware attacks [68]. Therefore, the necessary concepts to understand the usage of RL are introduced and explained. Building on top of Timo Schenk work, the goal is to extend it with federated learning for collaborative and private training in order to decrease the training time.

2.1 The Current State of IoT Security

Each new technology also comes paired with its own and possibly new security risks. The IoT is no exception to this rule. Since IoT devices seamlessly integrate deeply into many areas of our lives, they collect sensitive data and potentially uncover it. Although they are supposed to make life more comfortable and effortless, they pose a potential threat to privacy due to their constant Internet connection and lack of security [31]. Several notable works declare security and privacy to be the two biggest challenges in the sphere of IoT [22]. Captured IoT devices can be used to carry out further cyber attacks, since they are permanently connected to the Internet. Several massive Distributed Denial of Service (DDoS) attacks, for example, on the KrebsOnSecurity Blog and DNS provider Dyn with more than 500 GBps/s in 2016, were launched by botnets mainly comprising IoT devices [22].

A multitude of technical, economic, and social reasons complicate security in context of IoT. First of all, the extreme heterogeneity caused by the high fragmentation of device manufacturers, device types and versions results in a complex security landscape. Every manufacturer has to take care of the security of their devices and applications (as well as all different versions of the same device) themselves. IoT devices should make life more convenient, meaning that security mechanisms must not reduce usability. Since usability and security often pose conflicting requirements, the compromise is often made at the expense of security [31]. Since most IoT devices are cheap and consumer grade, many manufacturers decide to use their financial resources differently, for example, to lower the cost or to increase the margin instead of increasing device security.

Although most IoT devices sit inside private networks protected behind firewalls and NAT where they are shielded from external access from the Internet, they are still vulnerable. To be able to access the devices from anywhere around the world, they need to be exposed to the Internet. For this purpose, mechanisms like Universal Plug and Play (UPnP), that automatically open ports on the firewall, were developed. This creates a new entry point into the network and thus increases the attack surface. The end users of the IoT devices also have different security needs and knowledge and are differently able to protect their devices. In order to make it easier for non-technical users, insecure but convenient configurations such as enabling UPnP by default are favored by manufacturers over more complex and secure ones [56]. Often it would be enough to observe a few very simple rules to counteract this, such as changing standard credentials, using a network firewall, disabling UPnP and regularly installing software updates. However, most end users are not IT security experts and for them, successfully installing, running and using their devices is their main objective. Investing time and thought into the security of their devices plays a subordinate role for many inexperienced end users [62]. Therefore, manufacturers should start to embrace security by design in order to support their least tech-savvy customers. Automated measures such as those proposed and developed in this work offer the advantage that they potentially require little or no human intervention, making them attractive even for inexperienced users. As has been shown, the current state of IoT security is inadequate and must be significantly improved in order to make it as difficult as possible for malicious actors.

2.2 Malware affecting IoT Devices

IoT devices are special purpose computers equipped with sensors and/or actuators, able to communicate over the Internet. Therefore, they are just as vulnerable to malware as general purpose computers [58]. Software that has been specifically designed to maliciously alter a computers functional behaviour is denoted as malware [13]. As shown below, malware can be classified into families based on their various characteristics.

2.2.1 IoT Malware Types

- **Trojans**: Malware that disguises itself as or hides itself in legitimate software is called Trojan. Since they don't usually spread on their own, they depend on someone giving them access. Usually a regular user causes the malware infection by unknowingly installing and launching the Trojan [50].

- **Viruses**: A virus is a self-replicating and self-distributing type of malware that can perform various malicious tasks on an infected machine. The virus attaches itself to a host program in order to get onto the victim's computer. As soon as the host program is executed for the first time, the virus also becomes active and performs its evil actions and tries to infect other computers in the network [50].

- **Worms**: In contrast to the former two categories, a worm is a type of malware that does not require a user to launch it to cause the infection. Since worms usually tend to spread by exploiting security vulnerabilities to access host computers, it can spread autonomously throughout the network. Their potential to infect a large number of computers in a short period of time makes worms dangerous [50].

- **Ransomware**: is a type of malware used by cyber-criminals to infect systems and hold them or their data as ransom. Usually the data on the system is encrypted and in exchange for the decryption key a ransom payment in crypto-currency has to be made. This type of malware has been a growing problem in recent years [50].

- **Cryptojacking**: This type of malware exploits infected devices to mine crypto-currencies. A network of infected devices runs crypto-mining software for Bitcoin or Ethereum while trying to stay undetected. The fact that electricity and the computing power are paid for by the device owner, makes cryptojacking lucrative for cyber-criminals. [64].

- **Rootkits**: A malware that provides the attacker with remote administrative access to the target device is denoted as a rootkit. Hereby it tries to remain undetected, while providing the attacker with privileged or even root access to the system. The rootkit often comes bundled with different tools, allowing the attacker to perform a wide variety of malicious actions [63].

- **Botnet**: comprises many infected devices that are used together to perform other malicious activities such as large scale DDoS attacks. A command and control server is used to instruct all slave devices on what to do [50].

2.2.2 IoT Malware Attacks

Already since the advent of IoT, it has been target of many small and large attacks. Three particularly well-known attacks with strong impact from the last few years are outlined here in order to illustrate the extent of the problem.

The Stuxnet Worm

Most industrial IoT systems consider only partial security, relying on the premise of "isolated" networks, and controlled access environments. In such an environment it is closely monitored who can enter the facility and only these authorized people can access the internal network's IT resources. However, this did not stop the famous IoT Stuxnet worm from infiltrating Irans's uranium enrichment plant in 2010 [32]. The worm targeted an industrial control software (called Siemens Step7) and ultimately damaged or destroyed 984 uranium-enrichment centrifuges, leaving lasting marks on the Iranian nuclear program.

The Stuxnet incident cleared up several popular but wrong assumptions. First of all, until 2010 it was considered highly unlikely that a cyberattack would target a highly specialized software application. Usually, exploits of mass market software were preferred due to their prevalence. Additionally, a "safe" environment (implying disconnected from the Internet and with limited personnel access) was considered protection enough. Although the Stuxnet incident was particularly targeted, which is something that rarely occurs, it shows how serious the consequences of an attack can be.

The Mirai Botnet

Mirai is a malware, first discovered in May 2016, scanning the Internet for unsecured IoT devices like networked cameras, digital video recorders, and home routers. Mirai then propagates by using already infected devices for continuously scanning the Internet for new devices to infect. A simple attack of at least 62 common default usernames and passwords was used in 2016 to gain access to IoT devices with unchanged default credentials. Through further propagation, Mirai has managed to enslave over 500,000 IoT devices into the botnet. Interestingly, it contains a table of IP address ranges (e.g US Postal Service, the Department of Defense) to avoid. The behaviour of enslaved devices is controlled by a C&C server [80].

The IoT-based botnet was utilized to launch several large scale DDoS attacks against KrebsOnSecurity, Dyn, OVH. With traffic rates exceeding 1 TBps these were larger than ever seen before. Mirai can use multiple attack vectors simultaneously to attack a specific target (e.g Syn Flood TCP/ UDP on network layer and low and slow attack on application layer). It is conceivable that these could be used for large-scale IoT-based botnet DDoS attacks on critical infrastructure. Since the Mirai malware infects IoT devices that have common factory default usernames and passwords, the most obvious method for securing IoT devices is to change the default credentials [22].

The Operation Prowli

The so-called Operation Prowli was a traffic manipulation and cryptocurrency mining campaign. The attacks were based on various attack techniques including common known vulnerabilities, password brute-forcing and exploiting weak configurations. It managed to compromise more than 40,000 machines from various industries that exposed different services to the internet (Drupal, Wordpress, Servers with an open SSH port). The main goal of the campaign was monetizing the computing power and control of the devices. Using the devices under control in order to mine crypto currencies was the first source of income. Typically, mining crypto-currency is a resource intensive operation as the attacker has to pay for both hardware and the electricity consumed. However, none of these expenses incurred to the attackers behind Prowli. The second stream of revenue was getting payed for generating traffic for fraudulent websites. The incident illustrates how profitable it can be to exploit IoT devices for revenue generation [21].

2.3 Moving Target Defense

This section discusses MTD as a novel cyber security paradigm able to complement or outperform established approaches. Subsequently, a brief introduction to the MTD design principle explains which elements constitute a complete MTD definition.

The static nature of most IT systems has many practical advantages, such as reduced complexity and easier maintainability. However, this enables an attacker to systematically recon valuable information over a longer period of time, before launching a well-informed attack. Given enough time, an attacker will likely be able to gather enough information and identify enough vulnerabilities to ultimately be able to compromise the system. MTD as a cyber-security paradigm has been first proposed in 2009 [20]. The paradigm acknowledges that vulnerabilities can be present in any system and that there is an information asymmetry favoring the attacker. The idea is to thwart cyber attacks by periodically shifting system parameters and thereby constantly changing the attack surface [27]. Against a system equipped with MTDs, an attacker only has a limited time to find and exploit vulnerabilities, since these may no longer exist in the next system state [51].

If security mechanisms are only implemented in one location (e.g. at the edge of a system/ network), once this line of defense has been crossed, the attack is very likely to succeed, since it has reached the vulnerable core. Defense-in-Depth (DiD) is an information security principle addressing this issue by thoroughly distributing or layering the security mechanisms throughout the computer network. By creating a digital version of a shell game for the attacker. Therefore, MTD can also be seen as part of a DiD strategy. However, there are also a few disadvantages that should not go unmentioned. For example, it results in an increased resource consumption due to the repeated execution of MTDs. Furthermore, they not only increase the complexity for the attacker but also for harmless users [51].

2.3.1 MTD Design Principle

According to the MTD design principle by Cai et al., which is widely accepted in scientific literature, three fundamental questions must be answered in order to completely specify a technique: **WHAT**, **WHEN**, and **HOW** to move the system parameters [7].

- **WHAT** to move defines which features of the system will be altered over time by the applied MTD technique (e.g IP address, port numbers), thus changing the attack surface of the system. These variable properties are denoted as Moving Parameters (MP) and for each one there exists a defined set of values that it can take on [7].

- **WHEN** to move is about the temporality of changing the Moving Parameters (MPs), i.e. the frequency of moving. Finding an optimal moving frequency is important to balance security and system performance. It would be optimal to have a high moving frequency when the system is under attack and no adaptation otherwise [7]. According to Navas et al., the decision process that triggers the MP value to change can either be time-based, event-based or hybrid [51].

- **HOW** to move defines how the set of valid values for each MP gets defined. Furthermore, it specifies how the next MP value is determined and how the transition from the old to the new value takes place. Shuffling, Diversification, and Redundancy are the main three methods used [23].

The three most common techniques to move system parameters are the shuffle technique, the diversity approach and the redundancy tactic [23]. The Shuffle technique rearranges the system configuration continuously by periodically and randomly reassigning the MP values. The Diversity approach tries to provide functionally equivalent implementations (e.g., operating systems, web servers) that can then be dynamically exchanged for one another at runtime. The redundancy tactic aims at providing multiple functional backups of each network component (e.g. nodes, routes) [23].

2.3.2 MTD Techniques

Since the number of existing MTDs for Cyber Physical System (CPS) is too large, this work focuses on a subset of techniques that were developed by Jordan Cedeño at the UZH [4]. The extensible MTDs framework is capable of dealing with four C&C based malwares, one crypto ransomware and two user-level rootkits [4].

MTD against Command and Control (C&C) malware: This MTD that is effective against C&C malware assumes that the connection between the infected host and the C&C server has already been established. The aim is to disrupt the communication channel and thereby prevent the victim device from receiving commands to be executed. The private/local IP address acts as a moving parameter and is shifted to a new unoccupied

value [4]. An arp-scan of the local network helps to determine which IP addresses have already been taken and which are still unoccupied. To validate a successful transition, the MTD tries to establish a successful Internet connection by pinging a Google DNS. If this is the case, then the IP address change is successfully completed. This MTD technique denoted as private IP address shuffling.

MTD against Rootkits: A rootkit tries to stay undetected and operates by unlinking or manipulating "*/etc/ld.so.preload*" in order use the dynamic linker to preload malicious versions of shared libraries. Therefore, this MTD needs to be executed after the rootkit has been installed and tampered with /etc/ld.so.preload. In order to sanitize the system from the rootkit it is necessary to replace /etc/ld.so.preload with a sanitized version. This is enough to break rootkits that work by unlinking /etc/ld.so.preload and linking to another file of their own which points to their malicious shared libraries [4].

MTDs against Ransomware: Jordan Cedeño includes two promising MTD approaches effective against ransomware. Their common goal is to minimize the number of valuable user files that the ransomware is able to encrypt, and to terminate and isolate the encrypting process. Generally, the encrypting process is the one accessing and manipulating the highest number of files amongst all processes. The MTD's purpose is to bridge the time passing between ransomware infection and the point where the encrypting process can be identified and terminated. One approach focuses on keeping predefined data safe from encryption by changing its file extensions, while the other focuses on trapping the encrypting ransomware with dummy files, so that it gets stuck encrypting worthless data instead of valuable user data [4].

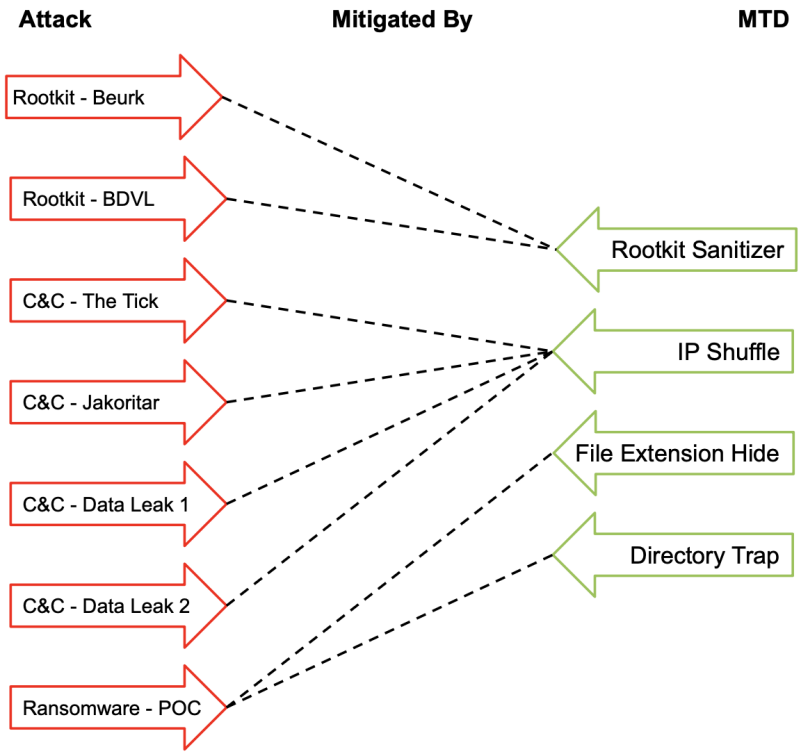


Figure 2.2: Mapping between Attacks and mitigating MTDs

2.3.3 Crowdsensing: ElectroSense

According to Navas et al., the amount of research conducted on MTDs for IoT device protection is sparse and most works have a broader focus on general purpose systems. Furthermore, there needs to be more work evaluating the effectiveness of MTDs in the context of real world scenarios [51] [9]. Therefore, this work examines the effectiveness of novel IoT-MTD techniques in the context of real world scenarios and applications like ElectroSense, a crowdsensing system "to collect and analyse spectrum data" [15]. The goal of the initiative is to make spectrum data openly accessible to increase the understanding of spectrum utilization over time and space [59].

The collaborative spectrum monitoring network consists of many inexpensive distributed spectrum sensing devices, for example a Raspberry Pi with a radio frontend for collecting spectrum information. The collected data is then sent to and processed by the backend. The collected information is provided back to the community as a service through an open Application Programming Interface (API) [59].

2.4 Reinforcement Learning

In this section, the necessary theoretical foundations regarding Reinforcement Learning (RL) is provided. This allows the reader to understand the conducted research in subsequent chapters. First, the idea behind RL and how it differs from other forms of machine learning is explained. The connection between RL and Markov decision processes (MDPs) is motivated and it is explained how RL manages to solve challenging sequential decision-making problems.

According to Russell and Norvig, intelligence is "*an emergent property of the interaction between an agent and its environment*" [61]. Consequently, reinforcement learning is built on the idea of having an agent that adapts its behaviour based on the feedback that it receives as reward from interacting with its environment [60]. The interaction with the environment provides a wealth of information about the effect of actions taken that can be used to refine the strategy. The agent's objective is to maximize the total cumulative reward received over multiple interaction, which encodes a long-term objective [67]. Therefore, RL can be viewed as the computational approach to learning from interaction. In human analogy, rewards fall into the categories of pleasure (positive reward) or pain (negative reward). Therefore, RL is closest to the learning process performed by biological agents, like humans and other animals.

Reinforcement learning is different from traditional machine learning approaches. Supervised learning is using training data with ground truth labeled examples provided by a knowledgeable external supervisor [60]. However, for interactive problems (e.g playing video games) it is often impractical to obtain examples of all situations paired with desired behaviour that the agent might have to act in. Therefore, the agent must be able to learn from making its own experience [60]. Furthermore, it is also different from unsupervised learning, which is typically about finding the structure hidden in collections of unlabeled data. Reinforcement learning is trying to maximize a reward signal instead of trying to find hidden structure. Therefore, reinforcement learning, as a semi-supervised learning methodology, is considered to be a third machine learning paradigm alongside supervised learning and unsupervised learning [60].

Since the system to be controlled is expected to be stochastic, the agent has to operate and optimize its strategy under significant uncertainty about the environment [60]. Problems with such characteristics can be best described as Markov decision processes (MDPs) [67]. An MDP is specified by a triplet $M = (X, A, P)$ where X denotes the countable set of states and A denotes the countable set of actions. The standard approach to solving MDPs is Dynamic Programming (DP). However, DP is a feasible solution method only for MDPs with very few states and actions. Therefore, RL algorithms can be seen as a transformation of infeasible dynamic programming approaches into practical methods that can be applied to large-scale problems [67].

2.4.1 The elements of Reinforcement Learning

In RL, the learner and decision-maker is called agent. The environment denotes everything that the agent can observe and it is represented in form of a state vector. At time step t, the environment that the agent is interacting with is in a state S_t. The agent receives the system state S_t as input and then calculates an action A_t which is sent back to the system. Action A_t performed by the agent causes the environment to transition to S_{t+1} and the agent receives the reward R_{t+1} [67]. The cycle is then repeated. Besides the agent and the environment, a reinforcement learning system comprises four additional main subelements: a reward signal, a policy, a value function and, optionally, a model of the environment.

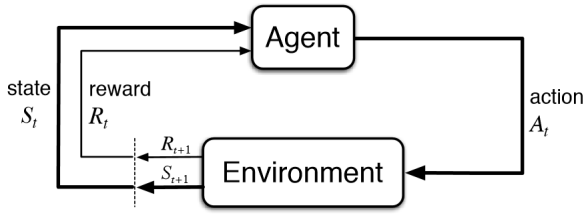


Figure 2.3: Interplay between Agent and Environment [60]

Every RL agent needs to have explicit goals, that they aim to achieve by interacting with the environment, for example, reward maximization. The reward signals can be seen as a stochastic function of the state of the environment and the actions taken. In mathematical terms, the agent seeks to maximize the expected cumulative reward it receives in the long run. The expected cumulative reward is defined as $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k_0}^{\infty} \gamma^k R_{t+k+1}$. R_t denotes the reward received at time step t. The value γ is the so called discount factor, $0 \leq \gamma \leq 1$, causing the infinite sum to have finite value [60]. The parameter is needed for continuing tasks where the interaction does not break naturally into episodes, but potentially goes on without limit [60].

In order to maximize the expected return G_t, the agent needs to optimize its behaviour that is encoded in a so-called policy. Under deterministic policy π, the action a taken in state s is denoted as $\pi(s)$. A stochastic policy is a mapping from states to probabilities of selecting each possible action: $\pi(a|s)\ \forall s \in States, a \in Actions$. The policy $\pi(a|s)$ defines the probability of taking action a in state s. Depending on the scenario, a policy can be a simple lookup table that contains the appropriate action for the requested state. Alternatively, a more complex method such as a neural network performing the mapping between state and action can be used. Through learning from interaction with the environment these probabilities are shifted towards actions which lead to higher cumulative rewards [60].

Almost all RL algorithms involve estimating value functions. A function allowing the agent to estimate how good it is to be in a certain state s is called state-value function $v_\pi(s)$ for policy π (Equation 2.1). A function allowing the agent to estimate how well it is for it to perform action a in a given state s for a policy π is called action-value function

$q_\pi(s, a)$ (Equation 2.2). These estimations are done with respect to a particular policy π. If such a function is used, the decision-maker is called value-based agent [60].

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \tag{2.1}$$

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{2.2}$$

The fourth optional element of an RL algorithm is a model of the environment. Based on this, the algorithm can be classified as either model-free or model-based. Such a model mimics the behaviour of the environment and can be queried by the algorithm for inferences about the expected environment response. For example, during learning or acting the algorithm might use the model to predict the resultant next state and next reward by providing a state and action. Alternatively, the full distribution of next states and next rewards might be requested. This information can be used for planning ahead for situations before they actually occur.

Algorithms that do not make use of a model or purely sample from experience such as Monte Carlo Control, SARSA, Q-learning, Actor-Critic are called model-free. This is due to the fact that they only use real samples from experience memory and never purely artificially generated examples or predictions. On the other hand, planning algorithms that make use of an environment model belong to the so-called model-based approaches. The archetypical model-based algorithms are Dynamic Programming (Policy Iteration and Value Iteration).

There are two key differences between DP and RL, which allow RL algorithms to deal with large, high-dimensional state- and action-spaces. Instead of having a complete model of the environment and transition probabilities, RL uses samples to compactly represent the dynamics of the control problem. Furthermore, function approximation is used in order to compactly represent action- and value-functions.

2.4.2 How to derive optimal policies

Different solution methods exist in order to find optimal policies. Due to the complexity of the problem and the high-dimensional environment state, this work focuses on temporal-difference learning methods that approximate the action-value function via a Deep Neural Network (DNN). The method of choice is Deep Q-Networks (DQNs) aboutwhich more details can be found in 2.4.5.

Finding a trade-off between Exploration and Exploitation

A challenge that arises for reinforcement learning, in contrast to the other machine learning paradigms, is that a trade-off between exploration and exploitation has to be made. When

maintaining expectations of action values, then at any point in time there is one with the highest expected value, which is called greedy action. The simplest strategy is to always select the action with the highest expected rewards, called greedy action selection (Equation 2.3). This strategy exploits current knowledge and doesn't spend time exploring actions currently perceived as inferior [60].

$$A_t = \underset{a}{argmax}\ Q_t(a) \tag{2.3}$$

When consciously choosing a non-greedy actions it is called exploration. It allows to more precisely determine the action values of the non-greedy actions. The epsilon-greedy policy (Equation 2.4) selects the best action (*i.e associated with the highest estimated reward*) most of the time with probability $(1-\epsilon) \in [0,1]$. However, sometimes a random action is chosen with small probability ϵ. The aim is to balance between exploration and exploitation. By allowing to have some room for trying new things and questioning old knowledge, ϵ-greedy methods eventually perform better [60].

$$A_t = \begin{cases} \underset{a}{argmax}\ Q_t(a) & with\ probability\ (1-\epsilon) \\ random\ action\ a & with\ probability\ \epsilon \end{cases} \tag{2.4}$$

2.4.3 Bellman Optimality Equation

A fundamental property of value functions used by reinforcement learning algorithms is the recursive relationship between the value of a state and the value of its possible successor states. The Bellman equation decomposes the value function into two parts, the immediate reward r and the discounted future values of successor states $\gamma v_\pi(s')$ when following a certain policy π (Equation 2.5). Analogously, the Bellman equation can also be defined for action-value functions $q_\pi(s,a)$.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] \tag{2.5}$$

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')] \tag{2.6}$$

The Bellman equation averages over all action-selection and transition-reward probabilities, weighting each by its probability of occurring. The computation of the value function is decomposed into recursive subproblems and finding their optimal solution. The state-value $v_\pi(s)$ or action-value function can then be computed as the the unique solution of the respective defined Bellman equation, making it one central element of RL.

2.4.4 Temporal Difference Learning

Amongst RL's model-free methods is temporal difference (TD) learning, an unsupervised learning technique capable of learning to predict the total discounted future reward (Equation 2.7).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} \tag{2.7}$$

TD learning understands that the total future reward consists of immediate reward r_{t+1} plus future discounted reward $\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. One characteristic of RL environments are sparse rewards, which means that often a long sequence of zero-reward actions occurs before a non-zero reward is received from performing an action. For example, in Tic-Tac-Toe (and many others) the reward is zero until the winner is determined in the last (terminal) state. This makes it more difficult to quantify the contribution of each individual action to the final reward. The concept of how Temporal Difference (TD) methods try to iteratively calculate the true action values is illustrated in Equation 2.8.

$$NewEstimate \leftarrow OldEstimate + StepSize * (Target - OldEstimate) \tag{2.8}$$

Instead of trying to compute the total future reward directly, TD uses the sum of immediate reward r_{t+1} plus an estimation of the future reward at the next moment in time $Q(s_{t+1}, a_{t+1})$ as target. This value acts as a new reference point and is compared against the previous estimation $Q(s_t, a_t)$. The algorithm calculates the deviation from the old estimate to the target, called temporal difference (TD) error, and adjusts the old estimate in direction of the new prediction. This behaviour of updating the value function based on one or more new estimated values is referred to as bootstrapping.

$$V(s_t) = V(s_t), +\alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \tag{2.9}$$

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \tag{2.10}$$

This method can be defined analogously for state-value functions (Equation 2.9) as well as for action-value functions (Equation 2.10). Iteratively executing this process moves the entire chain of predictions gradually towards the true values. TD learning builds the foundation of two important algorithms - Q-Learning and SARSA - which will be explained in the following subsection.

Q-Learning

Q-Learning is a model-free off-policy TD algorithm that is able to approximate the optimal action-value function $Q \approx q*$. It is an iterative algorithm that updates the action value Q(S, A) in direction of the temporal difference error computed as the difference between

the immediate reward R plus discounted future reward $\gamma Q(S', A')$ minus the current expected value $Q(S, A)$ (Equation 2.11). Figure 1 shows the procedural version of the algorithm.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(s_t, A_t)) \tag{2.11}$$

Algorithm 1: Q-Learning Algorithm [60]

Initialize $Q(s, a), \forall s \in S, a \in A(s)$, arbitrarily, and Q(*terminal-state*, $\cdot$) = 0
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    $S \leftarrow S'$;
  until S is terminal

SARSA

The acronym SARSA stands for State, Action, Reward, State, Action and the name stems from the (s, a, r, s', a') tuples taken by the algorithm as input. It is an on-policy TD algorithm for estimating Q $\approx$ q* that derives from Q-Learning by using an on-policy update rule to learn the Q-values. Comparing x with y, one can see that Q-learning always selects the greedy action $argmax_{a'} Q_t(s^{'}, a^{'})$ for the next state $s^{'}$ irrespective of the initial policy. SARSA stays on-policy and estimates the return for the next state-action pair $Q(s^{'}, a^{'})$ by following current policy's action $a^{'}$.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \tag{2.12}$$

Algorithm 2: SARSA Algorithm [60]

Initialize $Q(s, a), \forall s \in S, a \in A(s)$, arbitrarily, and Q(*terminal-state*, $\cdot$) = 0
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    $S \leftarrow S'; A \leftarrow A'$;
  until S is terminal

2.4.5 How Deep Reinforcement Learning can help

Deep reinforcement learning is achieved by integrating Neural Networks (NNs) used by Deep Learning (DL) into the framework of RL [37]. Mnih et al. proved in 2013 that deep RL systems can learn to play Atari games based on raw pixels as visual inputs [48]. The agent's ability to learn directly from high-dimensional sensor inputs, was a breakthrough in the field of RL. Several notable works of deep RL agents, like AlphaGo (Silver et al. [66]) beating the former world champion Lee Sedol or AlphaStar mastering the real-time strategy game StarCraft (Liu et al. [42]), have followed. Also it has shown huge potential for real world applications like autonomous driving (Pan et al. [55]), robotics (Levine et al. [38]), finance (Deng et al. [12]), automated surgery (Nguyen et al. [52]).

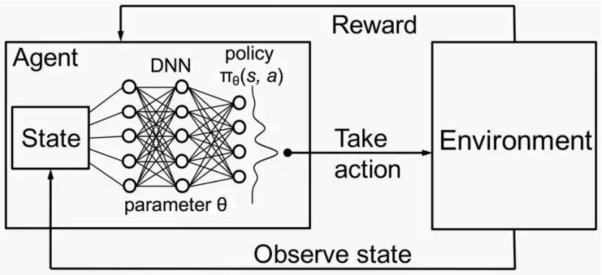


Figure 2.4: Schematic Representation of Deep Reinforcement Learning [72]

Algorithm 3: Deep Q-Learning with Experience Replay [60]

Initialize replay memory D with capacity N
Initialize online and target action-value functions Q^O and Q^T with random weights
Initialize exploration factor ϵ with a small value close to 1
for *episode in episodes* **do**
  Initialize s_t
  T = length(episode)
  for *t in* {*1, ... T*} **do**
    With probability ϵ select a random action a_t
    Otherwise select $a_t = max_a(Q^O(s_t, a; \theta))$
    step: Execute action a_t and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions (s_t, a_t, r_t, s_{t+1}) from D
    Calculate targets:
    $y_j = \left\{ \begin{array}{ll} r_j & \text{for terminal } s_{t+1} \\ r_j + \gamma max_{a'}(Q^T(s_{t+1}, a'; \theta)) & \text{for non-terminal } s_{t+1} \end{array} \right\}$
    Perform a batch gradient descent step using $(y_j - Q^O(s_t, a_j; \theta))^2$
    $s_t \leftarrow s_{t+1}$
    Perform $\epsilon - decay$ to minimize exploration over time
    if *tot_steps mod update_freq == 0* **then**
      $Q^T \leftarrow Q^O$ *// Update target network* Q^T
    end
  end
end

The memory requirement of tabular RL methods grows exponentially with the input state size. Therefore, methods like Q-Learning or SARSA are inefficient to solve problems involving high dimensional or continuous state spaces [52]. As shown in Figure 2.4 the Deep Neural Network (DNN) receives the state vector s as input and predicts the expected value of taking each action a_i in state s. By using a DNN in order to learn abstract representation from data, it is capable of dealing with high-dimensional scenarios [37]. In 2015 DeepMind developed a deep version of Q-learning called DQN (Deep Q-Network). Although this algorithm is very powerful, it is also often very data hungry and potentially unstable [52]. The DQN algorithm in Figure 3 is extended with two additional techniques, namely experience replay and the use of a separate target network, to stabilize learning and improve performance [47]. How this works is explained in the following subsection.

Experience Replay & Target Networks

Experience replay denotes the idea of storing the past N experiences $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ in a buffer memory for reuse. The concept was first motivated by Lin for three main reasons. Temporal difference learning is especially slow when the credit assignment needs to be propagated back through a long sequence of actions. By accelerating the credit propagation process, it is supposed to accelerate the learning. Secondly, some experiences might be too rare or too costly to obtain through trial-and-error and then only used once to adjust the networks before being thrown away. Furthermore, it helps to break the correlation between subsequent samples [17]. For the previously mentioned reasons, experience replay is an important improvement and extension of DQNs that is being used in this work.

Another issue of DQNs is that according to the Bellman equation (2.6) the action-value Q(s, a) is updated based on immediate reward plus the estimated target value Q(s', a'). However, since there is only one step between state s and s', their action values underlie some correlation. Hence, adapting the values for Q(s', a') indirectly influences neighbouring action-values Q(s, a), potentially making training very unstable. The trick to stabilize learning is to include two neural nets for the Q-value training and estimation. The so called target network is used for the Q(s', a') estimation in the Bellman equation. The Q-values predicted by the target Q-network are used to back propagate through and train the online Q-network. However, the target network is not directly trained and kept constant for a defined number of episodes. Its parameters are periodically overwritten with the updated parameters of the online Q-network. Only the later network is trained to learn the optimal state-action value function. The idea is that by using the Q values estimated by the semi-static target network for the update rule, the training of the online network becomes more stable [78].

2.5 Federated Learning

Nowadays, billions of smartphones and IoT devices constantly collect and generate valuable data. Sensitive information like photos, text messages, and health data mostly resides in isolated silos. New regulations, like the General Data Protection Regulation (GDPR), increase user data protection and make the centralization of data more difficult. Contradictory, traditional machine learning algorithms run on a single node and require the data to be stored in scope [43]. These are contradicting requirements and would thus render the data unusable. However, companies would still like to harness these dispersed data pools to build better products and smarter models [45]. Due to this conflict, Bharati et al. claim that privacy and security of data has become "*the most pressing issue that has to be addressed*" [6]. Federated learning was invented in order to make decentralized protected data accessible as training data for machine learning, while maintaining user privacy [45]. FL makes it possible to align both requirements since the dispersed data can be used to train a global model, while the sovereignty of local data can be retained [54].

For traditional machine learning, in a first step the training data x_i has to be accumulated into one data set $X = \{x_i | i \in N\}$ residing on a single server. The training, evaluation and validation of the model is then performed centrally. In a last step, the trained model is then deployed somewhere for on-device inference [45]. However, centralized training might infringe the privacy of user data. A federated learning network comprises multiple Edge Devices (EDs). In contrast with FL, one model M_i is trained on every edge device C_i and the client data remains stored privately on the device. Hereby, every participating node remains the single owner of its data. In a second step, the models are sent to a central server that aggregates them into a global model using a certain aggregation strategy (e.g federated averaging). This process is run for multiple rounds until the global model has converged. Then this global model can be deployed for on-device inference. By exploiting the computing power of edge devices, the requirements for the central infrastructure can potentially be lowered [3].

2.5.1 Definition of Federated Learning

Let there be N data owners $\{F_1, \dots F_N\}$, also called workers. Their shared objective is to train a common machine learning model based on their respective individual data sets $\{D_1, \dots D_N\}$. A conventional approach would be to aggregate all individual data sets into $D = D_1 \cup \dots \cup D_N$ and then use it to train a model M_{AGR}. In case the data aggregation is unfeasible, a federated learning can be used in order to collaboratively train a common model M_{FED} through a process where no data owner F_i exposes any of its data D_i to others. The performance V_{FED} of M_{FED} should be close to the performance of M_{AGR}, V_{AGR} [77].

2.5.2 Types of Federated Learning

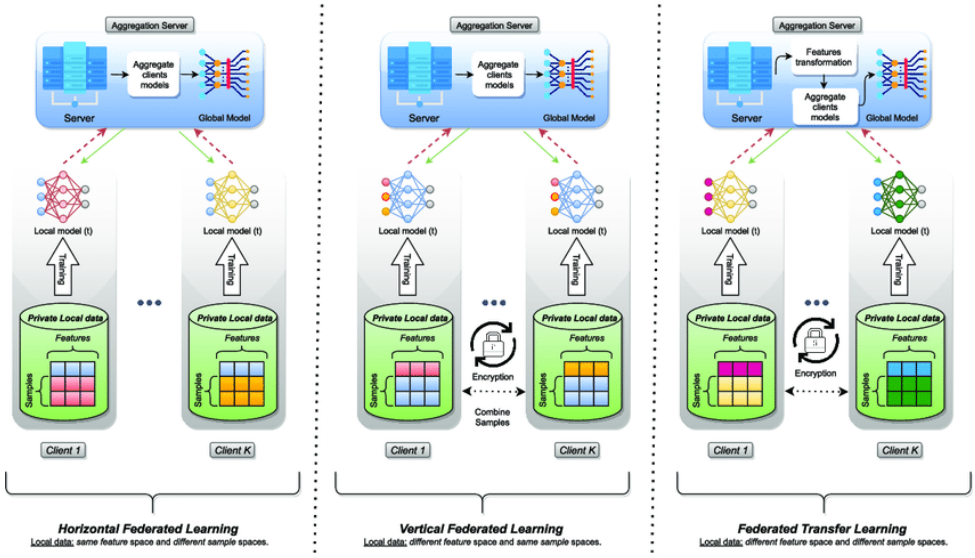


Figure 2.5: Three types of federated learning [18]

As shown in Figure 2.5, depending on how the data is distributed across particpiating worker nodes, FL may be classified as horizontal FL, vertical FL, or federated transfer learning (FTL) [54] [6]. Let D_i denote the data matrix of worker F_i where every row represents an observation and each column represents a feature. The individual data sets D_i comprise the feature space denoted as X and the label space Y. Furthermore, every sample is uniquely identified by an id from the sample ID space denoted I. The individual worker's data set is defined as $D_i = (I_i, X_i, Y_i)$ analogously to the complete training data set $D = (I, X, Y)$. The feature and sample space may differ amongst the various parties [77].

Figure 2.5 shows the three different federated learning paradigms. In the case of horizontal federated learning, the data sets share the same feature space but differ in sample space. The name originates from the fact that the data is partitioned horizontally in feature space. On the other hand, vertical federated learning is implemented when different workers have collected distinct features about the same subjects. In this case, the data sets share the same sample space but differ in feature space. The name originates from the data set being vertically partitioned. It is denoted as federated transfer learning, when neither the feature nor the sampling space is homogeneous across different workers [77] [54].

- **Horizontal federated learning**: same feature space, different sampling space
  $\Leftrightarrow (X_i = X_j, Y_i = Y_j, I_i \neq I_j, \ \forall D_i, D_j, i \neq j)$
- **Vertical federated learning**: different feature space, same sampling space
  $\Leftrightarrow (X_i \neq X_j, Y_i \neq Y_j, I_i = I_j, \ \forall D_i, D_j, i \neq j)$
- **Federated transfer learning**: different feature space, different sampling space
  $\Leftrightarrow (X_i \neq X_j, Y_i \neq Y_j, I_i \neq I_j, \forall D_i, D_j, i \neq j)$

2.5.3 Security & Privacy of Federated Learning

The most immediate privacy benefit of Federated Learning (FL) is that training data does not need to be centralized and thus remains under full control of the data owner. As a result, much less information is disclosed [3]. However, the parameter updates from an optimization algorithm like Stochastic Gradient Descent (SGD) that are exchanged between the individual clients and the aggregation server might indirectly lead to partial information leakage [77]. As shown by Phong et al., malicious clients might be able to infer important information like class membership of data subsets from these leaked gradients [57]. To prevent this, several techniques like Differential Privacy, k-anonymity and diversification have been developed. By adding a small amount of random noise to the client-side data the client's exact contribution during training gets obfuscated [77]. Although, FL has some inherent security advantages, recent works have uncovered several vulnerabilities. Yang et al. developed a gradient inversion attack, allowing an adversary to infer sensitive information about a client's private data set [77]. Data poisoning as well as model poisoning attacks have been shown to be effective in disturbing the FL process even in robust training settings [16]. This vulnerability arises from the large attack surface exposed by incorporating a large number of clients in the process [3]. The project focus is not on giving security or privacy guarantees, since honest clients can be assumed in the experimental environment. This work tries to combine the self-learning ability of RL with the high-level security and privacy benefits of FL. This work makes use of the (probably) best known weight aggregation strategy called Federated Averaging as shown by Algorithm 4. Hereby, the K clients are indexed by k, C denotes the ratio of clients to select, B is the minibatch size and E is the number of local epochs.

Algorithm 4: FederatedAveraging (FedAvg) by McMahan et al. [46]

execute *federated_training()*:
    initialize w_0;
    for *each round* $t = 1, 2, \dots$ **do**
        m $\leftarrow max(C * K, 1)$;
        $S_t \leftarrow$ (random set of m clients);
        for *each client* $k \in S_t$ ***in parallel*** **do**
            $w_{t+1}^k \leftarrow$ client_update(k, w_t);
        end
        $m_t \leftarrow \Sigma_{k \in S_t} n_k$;
        $w_{t+1} \leftarrow \Sigma_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$;
    end
function *client_update(k, w)*:
    // *Run on client k*;
    B $\leftarrow$ (split P_k into batches of size B);
    for *each local epoch i from 1 to E* **do**
        for *batch* $b \in B$ **do**
            $w \leftarrow w - \eta \nabla l(w; b)$;
        end
    end
    return w to server;

Chapter 3

Related Work

Machine Learning (ML) algorithms such as Deep Reinforcement Learning (DRL) and FL have been deployed both on the attacking and defending side of cybersecurity. In order to gain a better understanding of the domain, a first assessment of the current state of the art from scientific literature was necessary. The considered work either had to be related to RL based deployment of MTDs or combine FL with RL. Table 3.1 provides an overview of related work and classifies it along seven important dimensions. The table contains the references in order in which they appear in the text below. The index number links to the textual description while the **Ref.** column connects to the associated reference. Each work tries to mitigate certain **Threats** focusing on a subset of **Devices** in an certain **Application Domain**. The **Env.** column classifies whether the experiments were performed in a *real (R)*, *hybrid (H)*, or *simulated (S)* environment. The last three columns indicate whether **RL**, **FL** or **MTDs** are used in the respective work (✓) or not (x).

Table 3.1: Classification of Related Works

Ref.	Publ.	Application Domain	Device Focus	Threat	Env.	RL Usage	MTD Usage	FL Usage
1. [49]	2021	Anomaly Detection	IoT	MitM, DDoS	R	x	x	✓
2. [29]	2013	Network Security	Web Servers	DDoS	S	x	✓	x
3. [71]	2020	IT Security	IoT	DDoS, Spoofing, Jamming	S	✓	x	x
4. [40]	2020	Optimal Control	IoT	None	R	✓	x	✓
5. [14]	2019	Policy Planning	Servers	Probing Adersarial Actor	S	✓	✓	x
6. [36]	2021	Intrusion Prevention	IP Networks	DoS, Network Scanning	H	✓	✓	x
7. [76]	2022	Routing Randomization	SDNs	Eaves-dropping	R	✓	✓	x
8. [2]	2018	IT Security	IP Networks	DDoS, Botnets	R	✓	✓	x
9. [39]	2023	System Security	CPS	From NVD	R	✓	✓	x
10. [82]	2022	IoV	IoV	DDoS	S	✓	✓	x
11. [19]	2021	Network Security	CPS	DDoS	S	✓	✓	x
12. [10]	2021	Policy Planning	SDNs	XSS, SQLI	R	✓	✓	x
13. [79]	2021	Network Security	IoV	Various	H	✓	✓	x
14. [65]	2020	Web-App. Security	Clients & Servers	Various	S	✓	✓	x
15. [68]	2022	System Security	IoT	C&C, Rootkits, Ransomware	H	✓	✓	x

Mothukuri et al. propose an FL-based anomaly detection approach aimed at detecting and preventing intrusions in IoT networks. The implementation was done on top of the

PySyft framework and evaluated on the Modbus ICS data set. For each considered attack a separate global detection model exists. A random forest decision tree Ensembler combines the predicted attack specific probabilities into a final prediction. Experimental validation has shown that the proposed approach outperforms the classic centralized machine learning versions in privacy preservation of user data and provides higher accuracy in attack detection [49].

The MOTAG system by Jia, Sun, and Stavrou uses dynamic, hidden proxies as an MTD to mitigate network flooding attacks. Once a DDoS attack is launched against a proxy, the client proxy assignment is adapted to protect the application server availability. To retain service access, the legitimate and authenticated clients get re-assigned to yet hidden proxies [29]. To quarantine insider assisted attacks, Chai et al. developed a deep Q-learning-based shuffle MTD capable of iteratively finding optimal user-to-proxy assignments intended to isolate and block malicious clients [8].

Uprety and Rawat provide a comprehensive survey of RL methods for IoT security. Several notable examples of RL systems defending IoT devices from various attacks like Jamming, Denial-of-Service, Spoofing are listed [71].

Lim et al. advise a federated RL framework that allows multiple heterogeneous IoT devices to collaboratively learn an optimal control policy by sharing parts of their learning experience. A federated version of the actor–critic proximal policy optimization algorithm was able reduce the training time required to learn an optimal control policy for multiple connected rotary inverted pendulums by about 38%. The devices to be controlled were of the same type but had slightly different dynamics, making it unfeasible to replicate a mature policy from one device to the other. The collaboration amongst devices proved to accelerate the learning process, mitigate training instability and increase generalization [40].

Eghtesad, Vorobeychik, and Laszka propose a deep reinforcement learning based approach to finding optimal strategies for defenders and adversaries in a game-theoretic moving target defense model. In this environment two players - a defender and an adversary - compete for control over a set of servers. Since finding optimal strategies for deployment of MTDs analytically is difficult in a high dimensional state-action-space, the authors chose a Deep-Q-Network Learning (DQL) (Mnih et al. 2013) method to find a good policy. The policies found using Deep-Q-Learning outperform heuristic strategies for attacker and defender. It was proven that finding such policies is computationally feasible even in analytically complex environments. However, the experiments only consider a single defending agent in a synthetic environment.

The 'DIVERGENCE' framework by [36] is a DRL-based traffic inspection and intrusion prevention system. It includes an ip-address-shuffling-based MTD that is used to prevent intrusions and to defend against detected attacks. Since the network traffic is usually too large to be completely analysed, a deep deterministic policy gradient (DDPG) algorithm is trained to reroute suspicious traffic to the Intrusion Detection System (IDS) for inspection. The underlying SDN allows the MTD for easy IP/MAC address management, and routing path modification [36].

Xu et al. propose a routing randomization MTD with packet level granularity that is effective against eavesdropping attacks. Deep Deterministic Policy Gradient (DDPG) is used to generate random routing schemes which are then used in a SDN to route each individual packet independently through another routing path. Empirical results show that this effectively prevents sequential packets from getting intercepted, thus making eavesdropping more difficult. To avoid initial poor defense performance due to slow convergence, the model is pre-trained on a dual environment before being fine-tuned [76].

Albanese, Jajodia, and Venkatesan use MTDs to combat botnets that use stealthy communication methods to evade detection. Traffic inspection systems placed at central locations in the network were supposed to identify potentially infected hosts by uncovering connections to a C&C server. The MTD periodically altered the placement of detectors in order to force the stealthy botnet to periodically find unmonitored routes. In comparison to different detector placement strategies, the RL-based one performed best [2].

Li and Zheng propose a robust MTD framework capable of defending against unknown attacks. In contrast to other works, no a priori distribution is assumed over neither the attacker type nor type of attack. Such an assumption might lead to a suboptimal solution if the distribution shifts between the training of the MTD policy and its application. Since it is often unfeasible to collect a large number of samples covering a large variety of different attacks in security critical domains, a meta-policy is pre-trained in a simulated environment and then fine-tuned to new attacks based on only a small number of samples [39].

Zhang et al. focus on the emerging Internet of Vehicles (IoV) that relies on concurrent and dynamic wireless vehicle-to-infrastructure and vehicle-to-vehicle communication. They propose an intelligent MTD scheme on network level that is capable of defending against DDoS attacks by identifying and separating benign from adversarial vehicles. This is achieved by periodically mutating the SDN configuration based on DRL and then evaluating the trust of each vehicular client [82].

Gao and Wang propose an RL based mobile MTD strategy capable of balancing system security and system performance. The goal of the defender is to thwart DDoS attacks by launching a network shuffling MTD before the attacker completes the reconnaissance phase. However, taking defensive action without an attack happening has a negative effect on system performance and, hence, should be avoided. Q-learning is used to iteratively optimize and adapt to the attackers evolving strategy. Experiments in a simulated environment have shown that this allows the defender to find a strategy balancing between system security and performance [19].

Chowdhary et al. present a multi-agent reinforcment learning experiment that contains an attacker and a defender capable of deploying MTDs in a SDN managed cloud environment. The environment is fully observable for the defender whose goal is to defend against the attacker's actions such as reconnaissance, targeted attacks, etc. The attacker and defender's interactions are modeled in a two-player game theoretic model. The rewards obtained by each player are modelled according to the Common Vulnerability Scoring System (CVSS), taking into account the difficulty of compromising a vulnerability, the effort spent by each actor and the success rate of MTDs. Taking observable performance effects of the individual MTDs into account for reward calculation is novel. The empirical

evaluation yields a higher reward for defenders when using a reinforcement learning derived policy against an adaptive adversary [10].

Yoon et al. focus on strengthening the security on in-vehicle networks for autonomous driving with respect to existing vulnerabilities. They propose a multi-agent deep reinforcement learning framework called DESOLATER. The agent is capable of slicing the SDN-based in-vehicle network into isolated logical entities. For each slice a dedicated DRL agent is used to derive the optimal triggering interval for periodically shuffling the IP/MAC addresses by the SDN slice controller. The objective is to minimize security vulnerability while maintaining high Quality of Service (QOS). The viability of the proposed technique was implemented and evaluated based on an in-vehicular SDN testbed [79].

Sengupta and Kambhampati model MTDs as a two-player leader-follower game between a defender and an adversary. They incorporate uncertainty over attacker types into their game-theoretical model called Bayesian Stackelberg Markov Games (BSMGs). Furthermore, they also deliver a solution method capable of finding the reward optimal policy in presence of a rational and strategic adversary - Bayesian Strong Stackelberg Q-learning (BSS-Q). The approach tries to cope with the insufficiency of existing models yielding sub-optimal movement strategies when there is incomplete information about a rational adversary. Experiments with MTDs conducted in two application scenarios showed that policies learned using BSS-Q outperformed existing baselines [65].

This work pays tribute to Timo Schenk's work on optimizing MTD deployment on IoT devices using RL. The aim of this work was to train an agent capable of thwarting various cyber attacks by deploying MTDs on the target system. An online RL agent was trained in a real world application scenario (Crowdsensing with ElectroSense) resulting in a high probability of mitigating the launched cyber attack [68]. This work seeks to extend the previous one to include collaborative and private policy training using FL [28].

The related works can be divided into three main subcategories. The first category comprises works dealing with either MTD, RL or FL in isolation. A small number of works combining RL and FL form the second category. A large number of works using RL for MTD deployment or optimization could be identified, constituting the third category. A scientific work combining all three technologies has not been identified during the literature review.

Chapter 4

Data Collection and Feature Engineering

4.1 Data Collection

The data collection was performed by Timo Schenk during his master thesis. For the sake of completeness the process is briefly described here. More detailed information about the data collection can be found in his work [68]. The goal is to cover a wide range of system features so that a large number of different attacks is reflected and can be recognized. The data has been collected in order to be able to pre-train an agent in an offline manner, which is required due to learning time constraints. The two linux commands, '*perf*' and '*top*' were used to collect current status information about the system. Certain hardware events like instructions executed, number of soft/hard CPU interrupts or cache-misses are recorded in dedicated CPU hardware registers. perf provides access to these native kernel performance monitoring capabilities [33]. Out of the six different event types, only performance counters of type software events and kernel tracepoint events are considered. The data was collected on a Raspberry Pi 3 Model B with 1,2 GHz QuadCore 64Bit CPU. A recording period of 5 seconds was selected for data collection with perf [68]. Complementary information about the dynamic real-time overview over active linux processes, like CPU & memory usage, was recorded using *top* [44]. In total, each sample contains 100 recorded features, and the full list can be found in Table 4.4.

Table 4.1: Data Set 01 (Unfiltered Sample Distribution)

Behavior	**Type**	**#Samples**
NORMAL	decision state	14702
RANSOMWARE_POC	decision state	9381
ROOTKIT_BDVL	decision state	5698
ROOTKIT_BEURK	decision state	7358
CNC_THETICK	decision state	7704
CNC_BACKDOOR_JAKORITAR	decision state	4312
CNC_OPT1	decision state	5687
CNC_OPT2	decision state	4162
TOTAL		59004

The state samples are used to generate episodes of state transitions that mimic the state transitions an online agent could encounter when interacting with his environment and deploying MTDs. The more realistic the crafted data set is, the easier it is to transfer the offline pre-trained agent and policy to an online setting. It is important to mention that the performance counters collected by perf are affected by all processes and components that are running simultaneously. Therefore, in order to build a realistic prototype, as many real parts of the final overall system should be running during data collection. Data set 01 shown in Table 4.1 was collected as raw behavior samples from an ElectroSense device that was under attack by one of the seven malwares with no FL or RL system components (e.g Agent, MTD deployment) running and influencing the system state.

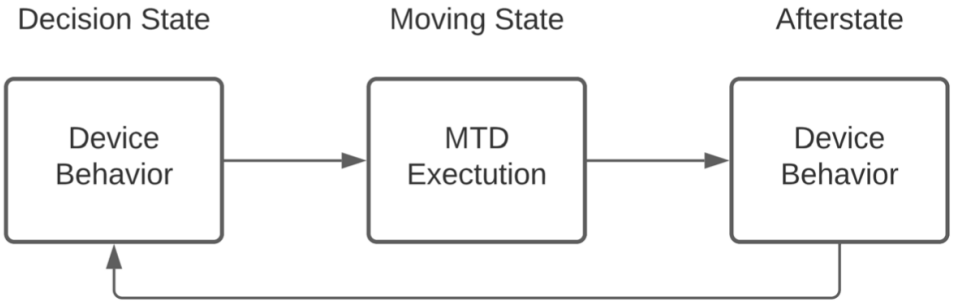


Figure 4.1: Decision and Afterstate in MTD deployment [68]

A second, more realistic data set comprising decision and afterstate samples was collected by Timo Schenk. As shown in Figure 4.1 the so-called decision state is the first state of an episode when no MTD has been executed on the device but the malware attack has potentially already begun. To collect the decision states, an RPi 3 Model B+ was set up as an ElectroSense sensor and all required dependencies were installed. Each of the seven malware attacks successively targeted a freshly setup device in order to record the device behavior under attack. In order to monitor the afterstates shown in Table 4.3, first of all the device had to be setup and brought into one of the respective eight decision states (NORMAL, ..., CNC_OPT2). Then, one of the four MTDs was executed and subsequently the performance counters signaling the device behavior were recorded again. This procedure was performed for each available behavior × MTD combination.

Table 4.2: Data Set 02 (Decision State Samples)

Initial Behavior	**State**	**Executed MTD**	**#Samples**
NORMAL	decision	None	4178
RANSOMWARE_POC	decision	None	1804
ROOTKIT_BDVL	decision	None	1658
CNC_BACKDOOR_JAKORITAR	decision	None	2018
ROOTKIT_BEURK	decision	None	2012
CNC_THETICK	decision	None	1507
CNC_OPT1	decision	None	2080
CNC_OPT2	decision	None	2075
TOTAL			17332

Table 4.3: Data Set 02 (After State Samples)

Initial Behavior	**State**	**Executed MTD**	**#Samples**
NORMAL	after	RANSOMWARE_DIRTRAP	2084
NORMAL	after	RANSOMWARE_FILE_EXT_HIDE	1971
NORMAL	after	ROOTKIT_SANITIZER	1971
NORMAL	after	CNC_IP_SHUFFLE	2031
RANSOMWARE_POC	after	RANSOMWARE_DIRTRAP	2095
RANSOMWARE_POC	after	RANSOMWARE_FILE_EXT_HIDE	2092
RANSOMWARE_POC	after	CNC_IP_SHUFFLE	636
RANSOMWARE_POC	after	ROOTKIT_SANITIZER	1812
ROOTKIT_BDVL	after	RANSOMWARE_DIRTRAP	1392
ROOTKIT_BDVL	after	RANSOMWARE_FILE_EXT_HIDE	624
ROOTKIT_BDVL	after	CNC_IP_SHUFFLE	657
ROOTKIT_BDVL	after	ROOTKIT_SANITIZER	1995
CNC_BACKDOOR_JAKORITAR	after	RANSOMWARE_DIRTRAP	2017
CNC_BACKDOOR_JAKORITAR	after	RANSOMWARE_FILE_EXT_HIDE	2013
CNC_BACKDOOR_JAKORITAR	after	CNC_IP_SHUFFLE	2024
CNC_BACKDOOR_JAKORITAR	after	ROOTKIT_SANITIZER	2085
ROOTKIT_BEURK	after	RANSOMWARE_DIRTRAP	1969
ROOTKIT_BEURK	after	RANSOMWARE_FILE_EXT_HIDE	1990
ROOTKIT_BEURK	after	CNC_IP_SHUFFLE	1975
ROOTKIT_BEURK	after	ROOTKIT_SANITIZER	2081
CNC_THETICK	after	RANSOMWARE_DIRTRAP	2095
CNC_THETICK	after	RANSOMWARE_FILE_EXT_HIDE	2087
CNC_THETICK	after	CNC_IP_SHUFFLE	2086
CNC_THETICK	after	ROOTKIT_SANITIZER	2093
CNC_OPT1	after	RANSOMWARE_DIRTRAP	2091
CNC_OPT1	after	RANSOMWARE_FILE_EXT_HIDE	2085
CNC_OPT1	after	CNC_IP_SHUFFLE	2079
CNC_OPT1	after	ROOTKIT_SANITIZER	2081
CNC_OPT2	after	RANSOMWARE_DIRTRAP	2072
CNC_OPT2	after	RANSOMWARE_FILE_EXT_HIDE	2095
CNC_OPT2	after	CNC_IP_SHUFFLE	2089
CNC_OPT2	after	ROOTKIT_SANITIZER	2082
TOTAL			60549

4.2 Data Balance/ Imbalance

When having n devices participate in the learning process, n local data sets also exist. Therefore, the intra- and inter-class balance of the n data sets must now be considered to determine if the data distribution is balanced/ imbalanced. In the context of federated learning, depending on the extent to which the data distributions of the individual n local data sets deviate from one another, a distinction between IID vs non-IID is made. In statistics, a set of random variables is denoted as Independent and Identically Distributed (IID) if each random variable follows the same probability distribution as the others and are all mutually independent. Real world data often does not fully satisfy the classic IID assumptions that each sample has to be independently drawn from the same distribution. Non-IID data is common when collected or generated by user devices. For example, facial images collected by cameras represent the demographics of each camera's location. When collecting individual handwritten digits and letters, these will slightly differ for each writer, making them non-IID [69].

Wang et al. briefly introduced local and global imbalance as the two most relevant types of class imbalance that can influence FL [73]. However, they did not provide formulas to quantify global and local imbalance. Global class imbalance looks at the entirety of training data and measures how evenly each label class appears globally. If each label appears globally at the same frequency, it is considered balanced. Additionally, the local distribution of label classes between individual clients should be considered. It is denoted as local imbalance when the sample distribution deviates significantly between individual clients. Xiao and Wang define two metrics that can be used to measure the global and local class imbalance [75].

$$MID = \sum_{c=1}^{C} \frac{n_c}{N} log_C \frac{Cn_c}{N} \tag{4.1}$$

The Multiclass Imbalance Degree (MID) of a data set with N data samples and C possible classes where n_c denotes the number of samples with label c is defined as shown in Equation 4.1. However, the Multiclass Imbalance Degree (MID) is insensitive to the size of data sets. MID eliminates the impact of the size of data set and ranges between 0 and 1. MID equal to 0 implies a strictly balanced data set. The larger the MID, the more imbalanced the data set is. In the experiments at hand, the MID value expresses how class imbalanced the global data set D is [75] [83].

$$WCS = \frac{1}{||L||_1||L||_2} \sum_{i=1}^{p} \frac{||l_i||_1}{||l_i||_2} L * l_i \tag{4.2}$$

The Weighted Cosine Similarity (WCS) measures the class imbalance between all participating clients. It takes into consideration the relationship between local and global imbalance. For a set of P clients with local data set $D_1, \ldots, D_P$ the WCS is defined as shown in Equation 4.2. The label distribution vector of client j is $l_j = [n_j^1, ..., n_j^c, ..., n_j^C]$ where n_j^c is the number of samples with label c. The global label distribution vector

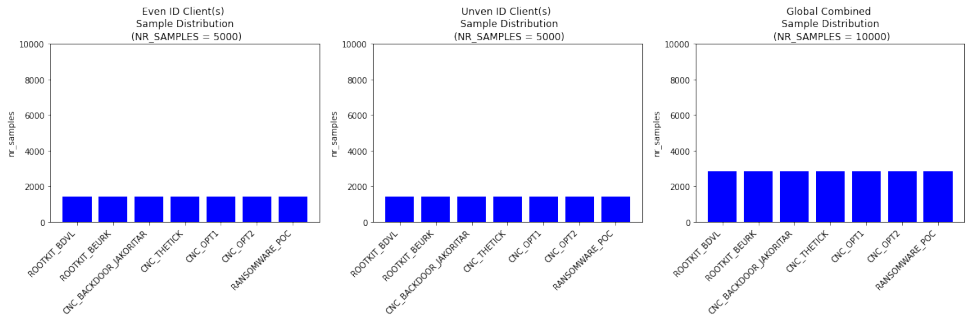
$L = [\sum_{i=1}^{P} n_i^1, \ldots, \sum_{i=1}^{P} n_i^C]$ where n^c denotes the total number of samples with class c [75].

Based on the two metrics MID, WCS, four kinds of class imbalance scenarios can be derived [75]. To better visualize these four class imbalance scenarios in the context of our scenario, suppose there are two devices $\{d1, d2\}$ and seven different attacks $\{a1, \ldots, a7\}$. In Figure 4.2 to Figure 4.5, the left and middle column diagram show for d1 and d2 the local frequency that the respective device is targeted by attack a1-a7. The rightmost chart shows the global combined attack distribution, which is the addition of the two local distributions.

4.2.1 Scenario 01 (Globally and Locally Balanced)

The global data is strictly class balanced and all local label distribution vectors follow the same distribution. This is the case for MID = 0, WCS = 1. As shown in Figure 4.2, this means that every attack is equally likely and targets each device with equal probability.

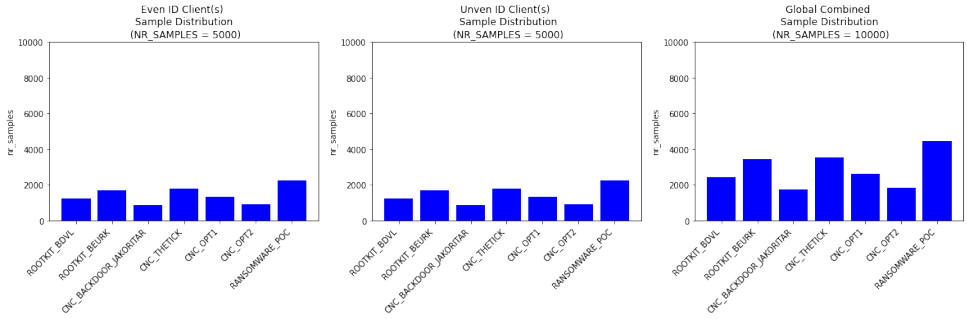
Figure 4.2: Globally and Locally Balanced Sample Distribution (MID=0.0 and WCS=1.0)



4.2.2 Scenario 02 (Globally Imbalanced & Locally Balanced)

The global data distribution is imbalanced, but the local label distribution vectors are balanced. This results in MID > 0, WCS = 1. As shown in Figure 4.3, this means that each attack on a global scale has a different probability, but the distribution of the attacks on the two devices follows the same probability distribution.

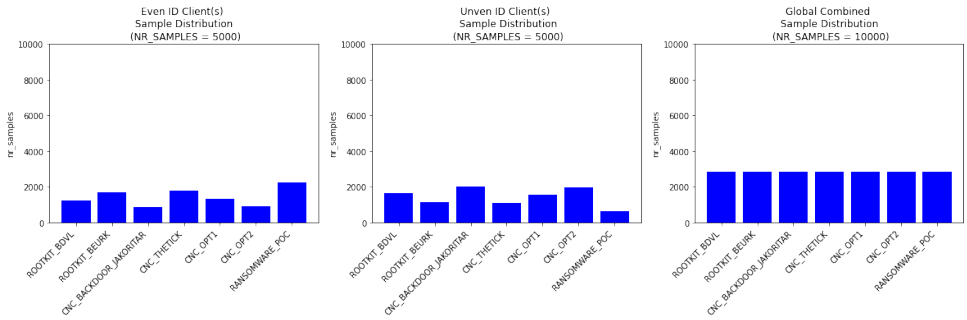
Figure 4.3: Globally Imbalanced but Locally Balanced Sample Distribution (MID=0.0267 and WCS=1.0)



4.2.3 Scenario 03 (Globally Balanced but Locally Imbalanced)

The global data is strictly class balanced but the local label distribution vectors follow different distributions. This is reflected in a MID = 0 and WCS < 1. In Figure 4.4, it can be seen that every attack is equally likely on a global scale, but they are divided differently between the two local devices or their training data respectively.

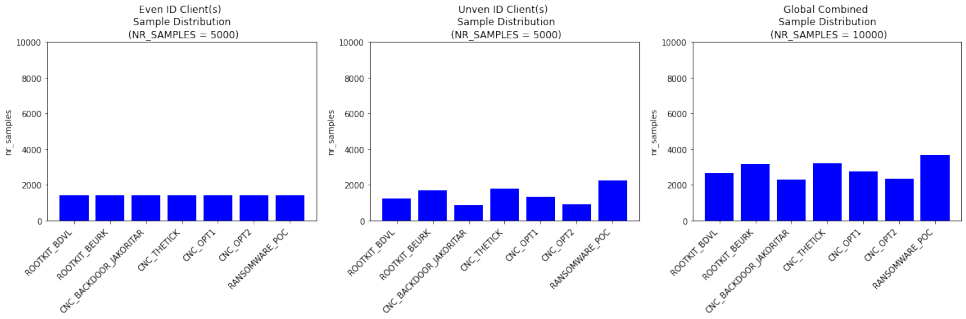
Figure 4.4: Globally Balanced but Locally Imbalancd Sample Distribution (MID=0.0 and WCS=0.9516)



4.2.4 Scenario 04 (Globally Imbalanced and Locally Imbalanced)

The global data presents to be class imbalanced as well as the local label distribution vectors (MID > 0, WCS < 1). This means that the attacks on a global scale have different probabilities and also target the individual devices with different frequency. This can be considered the strongest form of imbalance overall in FL.

Figure 4.5: Globally and Locally Imbalanced Sample Distribution (MID=0.066 and WCS=0.9878)



4.2.5 Scenario 05 (Client Exclusive or Client Distinct Classes)

Client exclusive classes mean that at least one label class is not present in one of the n local data sets. This also means that at least one of the n devices does not see at least one of the m attacks during training. As shown in Figure 4.8, a particularly extreme case of local imbalance is client distinct class setting, which means that each label class (attack) is only seen by one device during training.

Figure 4.6: Weak Client Exclusive Sample Distribution

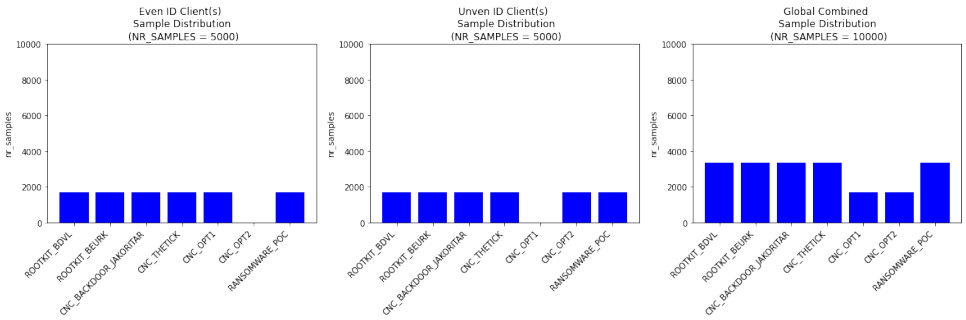


Figure 4.7: Medium Client Exclusive Sample Distribution

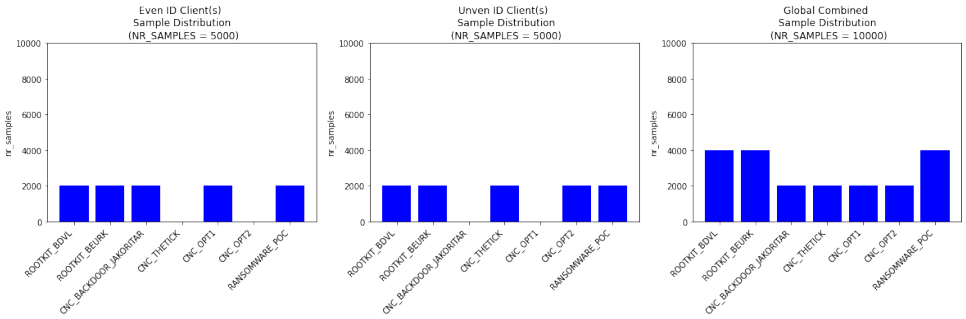


Figure 4.8: Strong Client Exclusive Sample Distribution

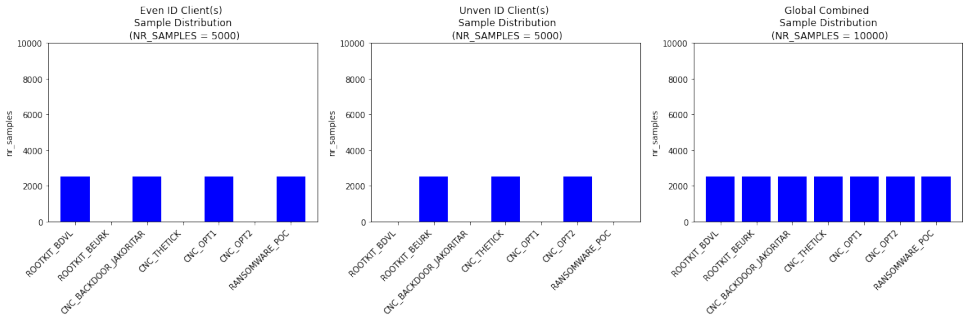
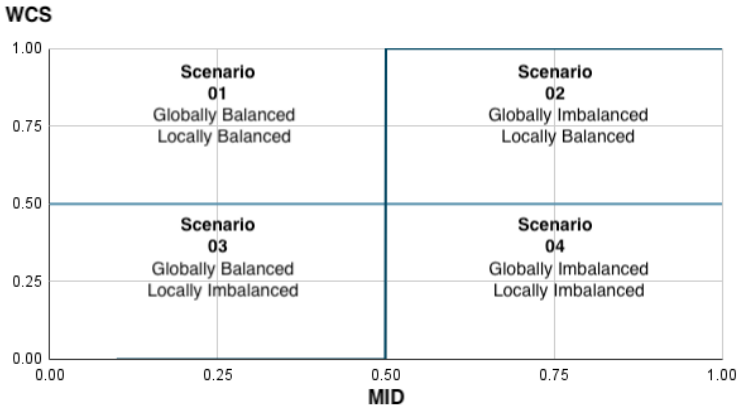


Figure 4.9: Schematic representation of the four global/local imbalance scenarios



4.3 Feature Engineering

Table 4.4: Overview of collected information and selected features

Feature	Event Source	Time Status	(Quasi) Constant	Highly Correlated	Selected as Feature
time		✓	x	x	x
timestamp		✓	x	x	x
seconds		✓	x	x	x
connectivity		x	✓	x	x
cpuUser		x	x	x	✓
cpuSystem		x	x	(✓)	✓
cpuNice		x	✓	x	x
cpuIdle		x	x	(✓)	✓
cpuIowait		x	x	(✓)	✓
cpuHardIrq		x	✓	x	x
cpuSoftIrq		x	x	(✓)	✓
tasks		x	x	(✓)	✓
tasksRunning		x	x	(✓)	✓
tasksSleeping		x	x	(✓)	✓
tasksStopped		x	✓	x	x
tasksZombie		x	x	(✓)	✓
ramFree		x	x	(✓)	✓
ramUsed		x	x	(✓)	✓
ramCache		x	x	(✓)	✓
memAvail		x	x	(✓)	✓
iface0RX		x	x	(✓)	✓
iface0TX		x	x	(✓)	✓
iface1RX		x	x	(✓)	✓
iface1TX		x	x	(✓)	✓
numEncrypted		x	x	(✓)	✓
alarmtimer_fired	alarmtimer	x	✓	x	x
alarmtimer_start	alarmtimer	x	✓	x	x
block_bio_backmerge	block	x	x	(✓)	✓
block_bio_remap	block	x	x	x	✓
block_dirty_buffer	block	x	x	(✓)	✓
block_getrq	block	x	x	x	✓
block_touch_buffer	block	x	x	(✓)	✓
block_unplug	block	x	x	x	✓
cachefiles_create	cachefiles	x	✓	x	x
cachefiles_lookup	cachefiles	x	✓	x	x
cachefiles_mark_active	cachefiles	x	✓	x	x
clk_set_rate	clk	x	x	(✓)	✓
cpu-migrations		x	x	(✓)	✓

Continued on next page

Table 4.4 – continued from previous page

Feature	Event Source	Time Status	(Quasi) Constant	Highly Correlated	Selected as Feature
cs		x	x	(✓)	✓
dma_fence_init	dma_fence	x	✓	x	x
fib_table_lookup	xib	x	x	x	✓
mm_filemap_add_to_page_cache	xilemap	x	x	x	✓
gpio_value	gpio	x	x	x	✓
ipi_raise	ipi	x	x	x	✓
irq_handler_entry	irq	x	x	(✓)	✓
softirq_entry	irq	x	x	(✓)	✓
jbd2_handle_start	jbd2	x	x	x	✓
jbd2_start_commit	jbd2	x	x	(✓)	✓
kfree	kmem	x	x	x	✓
kmalloc	kmem	x	x	x	✓
kmem_cache_alloc	kmem	x	x	(✓)	✓
kmem_cache_free	kmem	x	x	(✓)	✓
mm_page_alloc	kmem	x	x	(✓)	✓
mm_page_alloc_zone_locked	kmem	x	x	x	✓
mm_page_free	kmem	x	x	(✓)	✓
mm_page_pcpu_drain	kmem	x	x	x	✓
mmc_request_start	mmc	x	x	x	✓
net_dev_queue	net	x	x	x	✓
net_dev_xmit	net	x	x	x	✓
netif_rx	net	x	x	x	✓
page-faults		x	x	x	✓
mm_lru_insertion	pagemap	x	x	x	✓
irq_enable	preemptirq	x	x	(✓)	✓
qdisc_dequeue	qdisc	x	x	x	✓
get_random_bytes	random	x	x	x	✓
mix_pool_bytes_nolock	random	x	x	x	✓
urandom_read	random	x	x	(✓)	✓
sys_enter	raw_syscalls	x	x	(✓)	✓
sys_exit	raw_syscalls	x	x	(✓)	✓
rpm_resume	rpm	x	x	x	✓
rpm_suspend	rpm	x	x	x	✓
sched_process_exec	sched	x	x	x	✓
sched_process_free	sched	x	x	x	✓
sched_process_wait	sched	x	x	x	✓
sched_switch	sched	x	x	(✓)	✓
sched_wakeup	sched	x	x	(✓)	✓
signal_deliver	signal	x	x	x	✓
signal_generate	signal	x	x	x	✓
consume_skb	skb	x	x	(✓)	✓

Continued on next page

Table 4.4 – continued from previous page

Feature	Event Source	Time Status	(Quasi) Constant	Highly Correlated	Selected as Feature
kfree_skb	skb	x	x	x	✓
skb_copy_datagram_iovec	skb	x	x	x	✓
inet_sock_set_state	sock	x	x	(✓)	✓
task_newtask	task	x	x	x	✓
tcp_destroy_sock	tcp	x	x	x	✓
tcp_probe	tcp	x	x	x	✓
hrtimer_start	timer	x	x	(✓)	✓
timer_start	timer	x	x	x	✓
udp_fail_queue_rcv_skb	udp	x	✓	x	x
workqueue_activate_work	workqueue	x	x	x	✓
global_dirty_state	writeback	x	x	(✓)	✓
sb_clear_inode_writeback	writeback	x	x	x	✓
wbc_writepage	writeback	x	x	x	✓
writeback_dirty_inode	writeback	x	x	x	✓
writeback_dirty_inode_enqueue	writeback	x	x	x	✓
writeback_dirty_page	writeback	x	x	x	✓
writeback_mark_inode_dirty	writeback	x	x	x	✓
writeback_pages_written	writeback	x	x	x	✓
writeback_single_inode	writeback	x	x	x	✓
writeback_write_inode	writeback	x	x	x	✓
writeback_written	writeback	x	x	x	✓

4.3.1 Outlier Handling

Filtering Samples affected by External Events

The first N and the last sample for every behavior are possibly influenced by the required user interaction with the device, like logging in and starting the data collection. It is unknown whether these samples are representative of the behavior or whether they are potentially adulterated. Therefore, the data provider contains an optional flag that can be set to remove them during the data pre-processing [68]. In the course of the experiments, however, it becomes clear that the prototypes can also handle a certain degree of outliers. Since a prototype that has a certain robustness against outliers is considered advantageous in the IT security context, these samples were kept in the training data set.

Detecting and Removing Outliers

The data provider includes the option to detect outliers and to delete them from the data set. This pre-processing step can be switched on or off by setting a flag. The Z-Score method or standard score method works best for data that follows a normal distribution [68]. When enabled, the Z-Score method removes any sample that deviates more than

3 standard deviations from the mean. However, as shown inFigure 4.10, the individual feature values do not necessarily follow a normal distribution. Therefore, the Z-Score method is not used. Outliers are not removed, since the system should be somewhat robust to outliers.

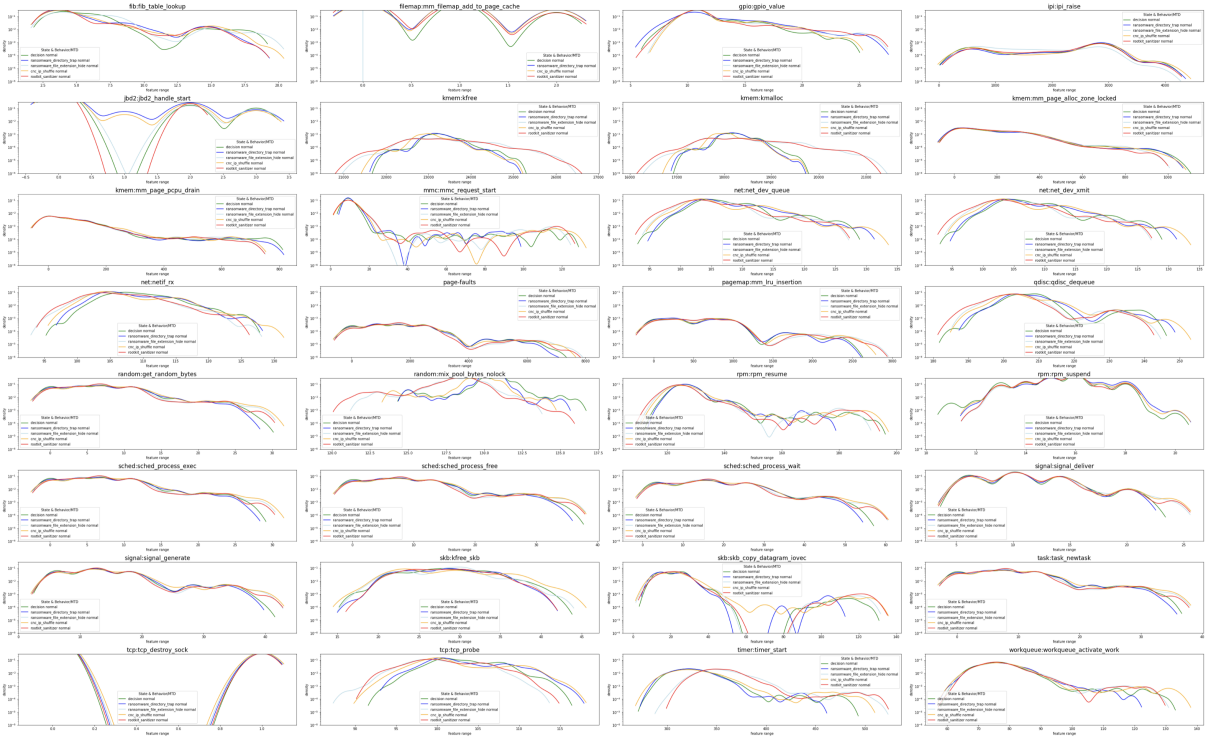


Figure 4.10: Kernel Density Estimation of Training Data

4.3.2 Feature Selection

Table 4.4 contains all features and displays which features were removed due to what reason and which were retained for training. First of all, all time status features (time, timestamp seconds) were dropped, since their variation in value is only explained by time and not connected to the device state. Secondly, all quasi constant features (*connectivity* *alarmtimer_fired*, *alarmtimer_start*, *cachefiles_create*, *cachefiles_lookup*, *cachefiles_mark_active*, *cpuHardIrq*, *cpuNice*, *dma_fence_init*, *tasksStopped*, *udp_fail_queue_rcv_skb*) were detected and removed. The threshold for this operation was set to 99%, meaning that 99% of this feature's values have to be identical. Last but not least, there would be the removal of the highly correlated features. If three random variables A, B and C are highly correlated (>95%), then two out of three could be dropped without losing too much information. However, the correlation of the features would likely differ from scenario to scenario. Therefore, since it was desired to build a model with as few prior assumptions as possible, this was not done and the variables are marked in round brackets in Table 4.4.

4.3.3 Feature Scaling

Two different feature scaling methods were tried - MinMax- and Standard-Scaling. These can either be fitted to the entire data set (all behaviors) or to just one data set. The scaler was programmed to only fit the samples representing normal behavior and then apply it to all samples. It is important to have enough representative samples for fitting the scaler as this step would also have to be performed in an online setting. If the min/max values or mean and standard deviation of the online samples would deviate significantly from those used for the offline prototype, it would probably degrade performance. However, as shown in Table 4.1 to Table 4.3, enough training data should be available for realistic scaling. Using Standard Scaler ensures that the training data has zero mean ($\mu = 0$) and unit variance ($\sigma = 1$) [69]. MinMax-Scaling, on the other hand, ensures that all features lie in the range from 0 to 1. Since the scaler fitting is performed only on the normal training data and not on the entire data set, this is a certain simplification, but it is permissible. During the course of this work it has been shown that MinMax scaling has a better effect on the test performance than the standard scaling, and therefore it was preferred. Three different boxplots of the features after min-max scaling can be found in Appendix A of this work. Each boxplot compares the range of features under normal behavior with the range of features when the device is affected by either rootkit BEURK, BDVL or C&C OPT2. The behavioral fingerprint of C&C OPT2 differs greatly (Figure A.3), while the behavioral fingerprint of rootkit bdvl differs only slightly (Figure A.2) from the normal device state. However as shown in Figure A.1 the device behavior when infected with rootkit BEURK resembles the normal device state very closely, making it probably the most difficult one amongst all attacks to recognize and mitigate.

Chapter 5

Design & Implementation

The design and implementation of the individual FL clients, which consist of a local agent, state anomaly detector and the local environment of the host device was shaped by Timo Schenk's non federated, single agent system architecture, as shown in Figure 5.1. Figure 5.1also shows that for experimental purposes a total of seven different malware attacks exist. Since the experiments run in a simulated environment, it can be ensured that at least one mitigating MTD is available for every attack. This means that it is guaranteed that the agent has the ability to ward off all existing attacks with its four MTDs. Of course, this assumption does not hold in a real network environment.

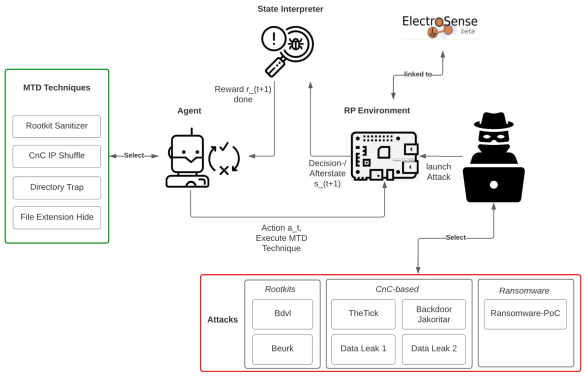


Figure 5.1: Single Agent System Architecture proposed by Timo Schenk [68]

In RL it is important to balance exploration and exploitation. A simple epsilon decay strategy was implemented on each local agent to manage this trade-off. As shown in Table 5.1, the initial value of epsilon is 1 on each device which means that initially each local RL agent only selects random actions in order to explore his options. For the first 80% of the total per client trained episodes, a small linear epsilon decay is performed at the end of each individual episode. A minimum value for epsilon of 0.01 is never undercut. This means that every local agent slowly moves from exploration of options to exploitation of already acquired knowledge. During the last 20% of episodes per client, training is carried out with the maximum exploitation ratio.

5.1 Prototype 01

The first FL prototype uses a supervisor that is capable of labeling every state-action-reward transition with perfect accuracy for the generation of the reward signal. Therefore, this prototype serves as a baseline under ideal conditions. The federated architecture extends the stand-alone architecture with a central coordinator, managing the training of all participating agents and updating the global agent. Each training round follows the same five steps as shown in Figure 5.2. At the beginning of each round, the current global model is sent to all clients (1). Each local agent is then trained for a specified number of episodes in order to obtain the next model version (2). The coordinator then fetches the locally updated model from all clients. A simple federated averaging (FedAvg) algorithm where each locally trained model is weighted equally is used for aggregation (3). The aggregated model then replaces the old global model as the new global model (4). This cycle is repeated until the number of global training rounds is reached (5). The raw device behaviors from data set 1 (see Table 4.1) are used as training data. These were divided equally into 10 strides and distributed to the 10 participating clients.

This architecture was implemented locally with four different client simulation strategies. The training of the individual clients can be performed sequentially, multi threaded (each in a single threat), as individual processes or as part of a multiprocessing pool.

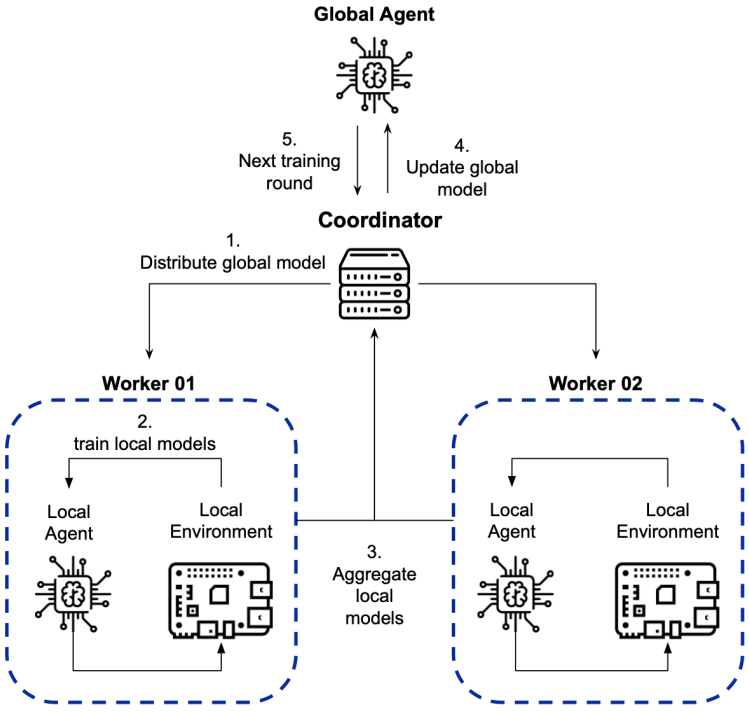


Figure 5.2: Prototype 01 / Federated Architecture

5.1.1 Experiment 1.1

The first experiment was supposed to investigate how individual clients can be simulated locally and what performance implications the FL has. In a first step, the performance of single-threaded and multi threaded client simulation was compared. A total of 10,000 episodes were used for training on $i \in \{1, \ldots, N\}$ clients distributed over 10 rounds. This means that the number of episodes per global training round decreases linearly with increasing number of clients. The initial hypothesis was that training time would decrease as the number of clients trained in parallel increased.

Listing 5.1: Python Multithreading Implementation

```
case Execution.SINGLE_THREADED:
    for client in self.clients:
        client.train_agent(nr_epochs_per_round)

case Execution.MULTI_THREADED:
    threads = []
    for client in self.clients:
        t = threading.Thread(target=Client.train_agent, args=(client,
            nr_epochs_per_round))
        t.start()
        threads.append(t)
    for t in threads:
        t.join()
```

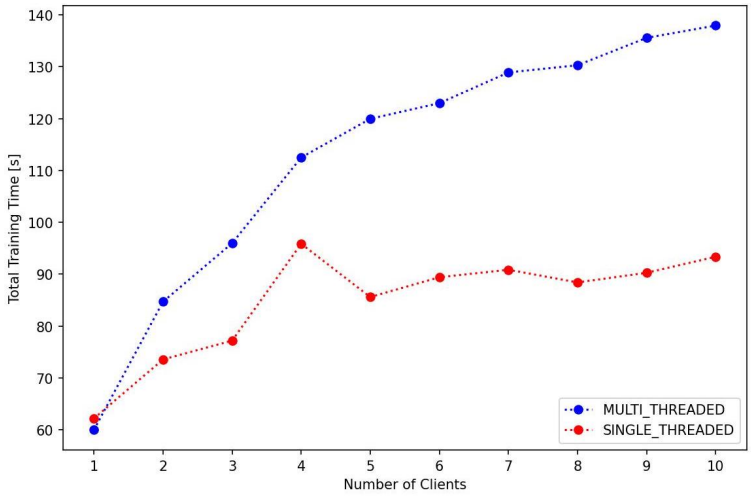


Figure 5.3: Total training time depending on the number of clients (Single vs. Multi-Threading)

Figure 5.3 shows the total training time for 10,000 episodes using different numbers of participating clients. Contrary to the initial hypothesis, it can be seen that multi-threaded training (*blue curve*) consistently takes more time than single-threaded training (*red curve*) across the whole researched range of participating clients. This unexpected behavior has prompted further investigations. Literature research uncovered that there

exists a Mutex that allows only one thread to hold the control over the Python interpreter at the time, called the Python Global Interpreter Lock (GIL). Therefore, only one thread at a time can be in a state of execution. The GIL prevents deadlocks and protects the reference count variables from race conditions [5]. This creates a performance bottleneck in CPU-bound and multi-threaded code, effectively making any CPU-bound Python program single-threaded. However, not every implementation of the Python interpreter is using the GIL, but CPython as the standard and most common reference implementation does. Hence, Python multi-threading is not an intelligent option to simulate multiple clients and at the same time reduce the training time. Therefore, different methodologies for simulating individual workers on one machine had to be implemented and evaluated.

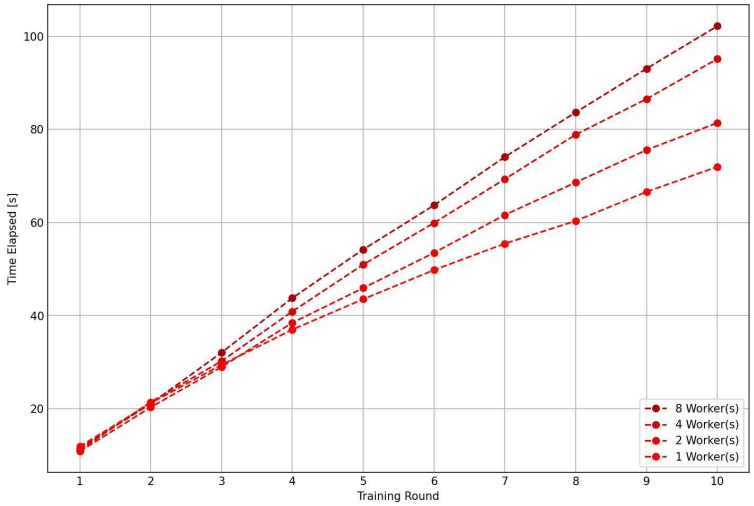


Figure 5.4: Time elapsed after nth training round (Single-Threaded)

In Figure 5.4, the time elapsed until the end of the nth global training round was recorded for $\{1, 2, 4, 8\}$ sequentially trained clients. A relevant observation is that the total training time increases with the number of clients even in single threaded execution. This can be attributed to the administrative overhead caused by managing multiple clients including the distribution of and aggregation of models. Since Python multi-threading did not turn out to be a viable option for simulating multiple local clients, multiprocessing was evaluated as an alternative. As shown in Listing 5.2, multiprocessing uses process-based parallelism. Although the Process and Pool class give the possibility to execute CPU-bound tasks in parallel, the execution is slightly different. Pool manages a pool of worker processes and waits until all processes have run until the end before returning the result. In order to aggregate all local models into a new global model, all local training processes must have been completed. Therefore *join()* needs to be called for each child process to block the parent process until the aggregation can safely be performed.

Listing 5.2: Python Multiprocessing Implementation

```
case Execution.MULTI_PROCESSING:
    threads = []
    for client in self.clients:
        thread = multiprocessing.Process(target=Client.train_agent,
            args=(client, nr_epochs_per_round))
        thread.start()
        threads.append(thread)

    for thread in threads:
        thread.join()

case Execution.MULTI_PROCESSING_POOL:
    pool = multiprocessing.Pool(processes=nr_clients)
    pool.starmap(Client.train_agent, self.clients)
    pool.close()
    pool.join()
```

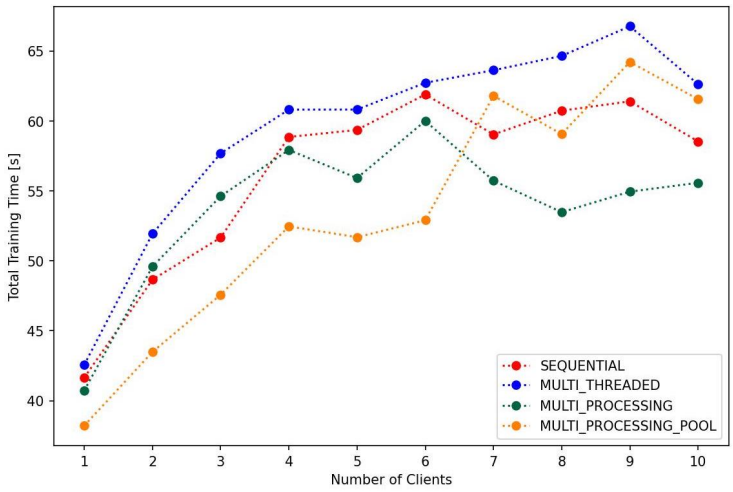


Figure 5.5: Total training time depending on the number of clients for different execution methods

Figure 5.5 compares the total training time for a range of workers across the four implemented training methodologies (Sequential, Multi-Threaded, Multi-Processing, and Multi-Processing Pool). Consistent with the first observation, the multi-threaded training times here are the highest as well. The multiprocessing with pooling achieves slightly lower training times than sequential training for small number of clients (approx. <=5). In comparison, regular multiprocessing achieves lower training times than sequential training for higher number of clients (approx. >=5). Therefore, regular multiprocessing seems to be the most promising approach for local FL training simulation. Since the pre-training time is still in the lower three-digits range, it does not pose a bottleneck for the first prototype and does not have to be further optimized yet. However, it is conceivable, especially in a later prototype with real physical clients, that the training time poses a bottleneck that can be reduced by using FL.

5.1.2 Experiment 1.2

Experiment 1.2 was about finding the best hyperparameter combination in order to build later experiments on top of them. The 30,000 training samples were divided between 10 clients and subsequently each client was trained for 3,000 episodes distributed over 30 FL rounds of 100 episodes each. In Table 5.1, the best hyperparameter combination found by manual hyperparameter search is shown in **bold**. Since the total number of customizable and optimizable hyperparameter is over 20, an exhaustive search was not possible to perform and therefore a manual hyperparameter search focusing on the DQN architecture and the most important training hyperparameter was performed.

Table 5.1: Hyperparameter Search Combinations

Server Hyperparameter	
NR_CLIENTS	10
NR_ROUNDS	30
NR_EPISODES_PER_ROUND	100
NR_EPISODES_PER_CLIENT	3000
TOTAL_NR_EPISODES	30.000
DQN Hyperparameter	
NR_NEURONS_PER_LAYER	**(128, 64)**, (128, 64, 32), (128, 64, 32, 16)
ACTIVATION_FUNCTION	ReLU, Tanh, Sigmoid, **SELU**
DROPOUT	**0**, 0.2, 0.5
Training Hyperparameter	
OPTIMIZER	SGD, **Adam**, RMSprop, Adagrad
LOSS_FUNCTION	MAE, **MSE**, RMSE
GAMMA	0.1, 0.2, 0.3, 0.4, **0.5**, 0.6, 0.7, 0.8, 0.9
LEARNING_RATE	1e-2, 1e-3, **1e-4**, 1e-5
L2_REGULARIZATION	0, 1e-1, **1e-2**, 1e-3, 1e-4
EPSILON_START	1.0
EPSILON_DEC	0.8/NR_EPISODES_PER_CLIENT
EPSILON_END	0.01
AGGREGATION_STRATEGY	FedAvg

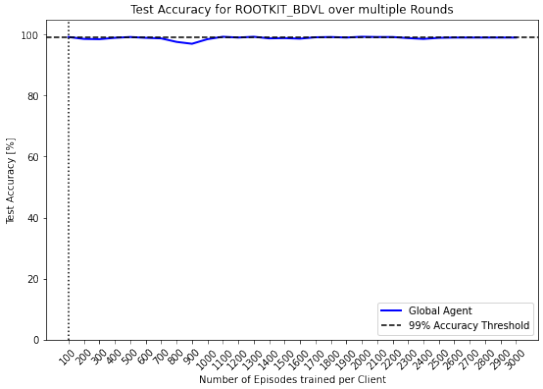
The curves in the following two Figures 5.6 & 5.7 show the test mitigation performance for each of the seven malware attacks. The last curve shows the micro accuracy of the entire system, which is calculated according to Equation 5.1 as the ratio of repelled attacks to the total number of seen attacks.

$$Micro\ Accuracy\ =\ \frac{1}{N}\sum_{i=1}^{N} I(y_{pred}(x_i) = y_{true}(x_i)) \tag{5.1}$$

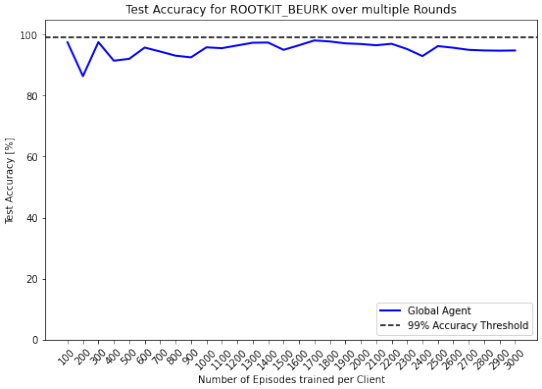
In this experiment, a globally and locally balanced sample distribution from scenario 01 is used (see Figure 4.2). The goal is to allow a baseline comparison between centralized and federated training.

Experiment 2.2.1 - Centralized Training Baseline (One Client)

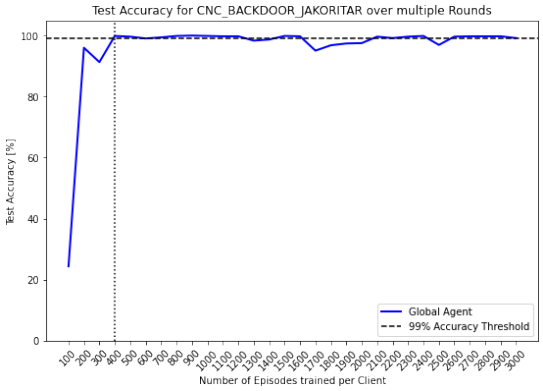
Figure 5.6: Prototype 01 Cenralized Training Baseline
(Attack Mitigation Performance over multiple Training Rounds)



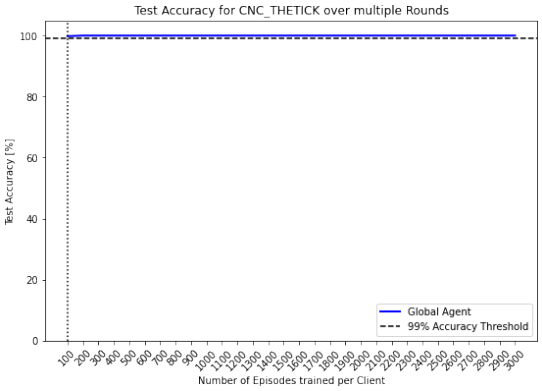
(a) ROOTKIT_BDVL



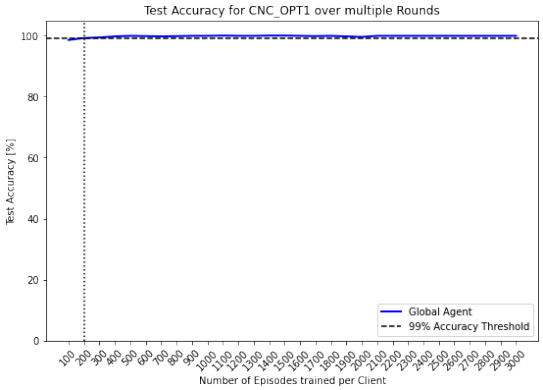
(b) ROOTKIT_BEURK



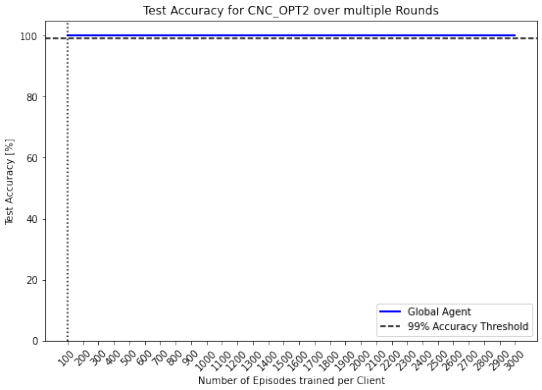
(c) CNC_BACKDOOR_JAKORITAR



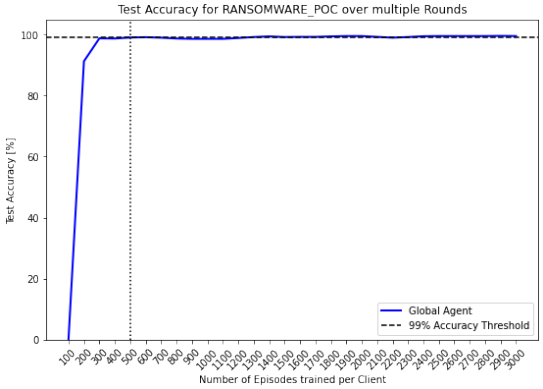
(d) CNC_THETICK



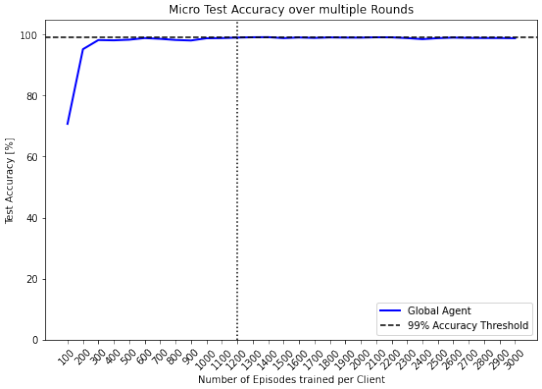
(e) CNC_OPT1



(f) CNC_OPT2



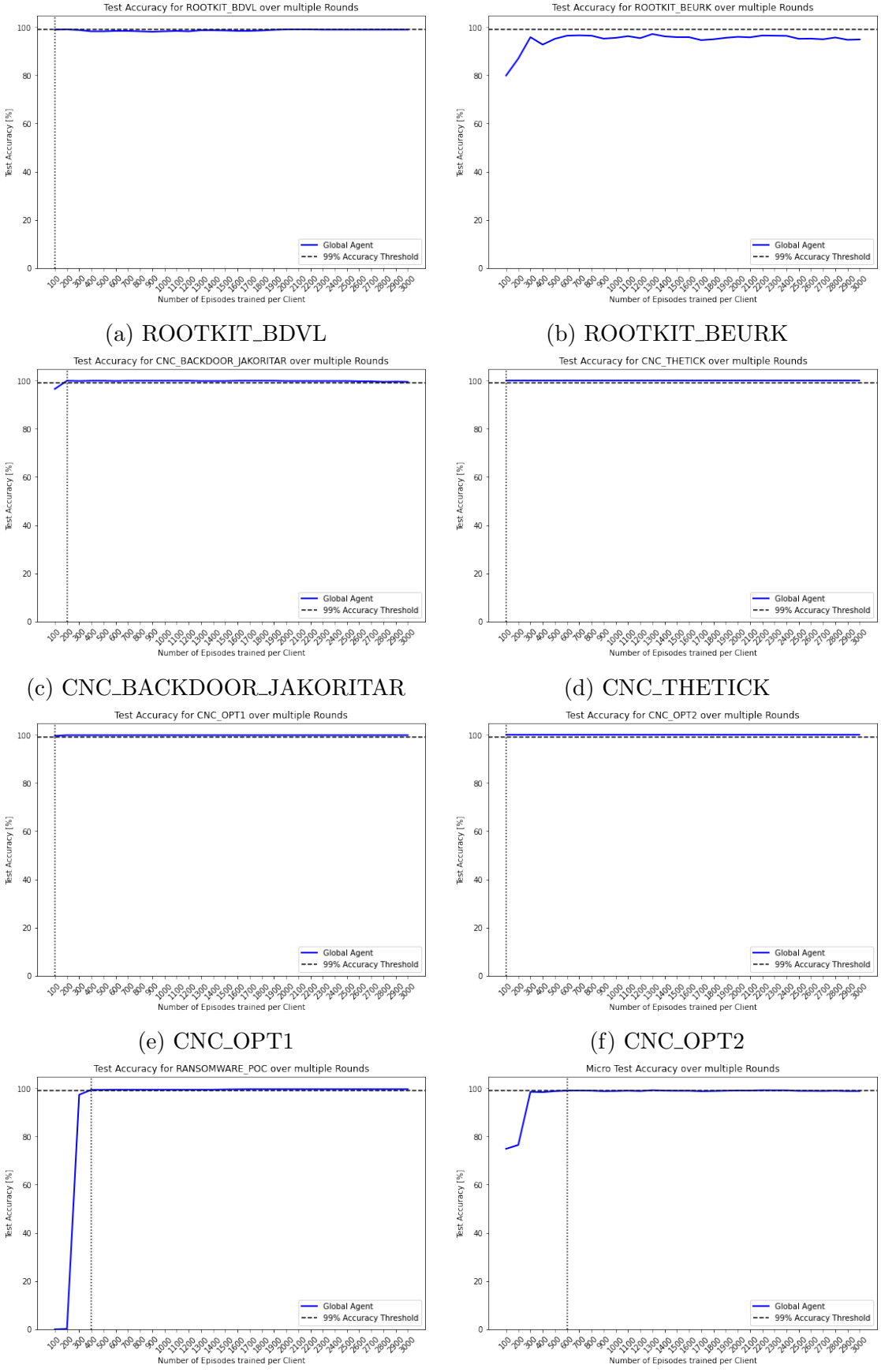
(g) RANSOMWARE_POC



(h) GLOBAL

Experiment 2.2.2 - Federated Training Baseline (10 Clients)

Figure 5.7: Prototype 01 Federated Training Baseline
(Attack Mitigation Performance over multiple Training Rounds)



(a) ROOTKIT_BDVL

(b) ROOTKIT_BEURK

(c) CNC_BACKDOOR_JAKORITAR

(d) CNC_THETICK

(e) CNC_OPT1

(f) CNC_OPT2

(g) RANSOMWARE_POC

(h) GLOBAL

When looking at the test curves of Experiment 1.2 in Figure 5.6 & 5.7, which reflect the test performance for each class over all training rounds, it is noticeable that after the first two training rounds (200 locally trained episodes) the mitigation test accuracy for ROOTKIT_BDVL, CNC_THETICK, CNC_OPT1, and CNC_OPT2 is already over 99%. A horizontal, dashed line in each diagram marks this self-defined 99% mitigation performance threshold. Should this threshold be exceeded at some point during training, then a second vertical dotted line will be displayed, indicating the training round during which this was achieved for the first time. However, for ROOTKIT_BEURK, this is neither the case for part 1 (Figure 5.6b) nor part 2 (Figure 5.7b) of the experiment, which is why no vertical line is displayed.

The training of the entire system can be considered complete when this convergence threshold has been exceeded for both micro and macro accuracy. When comparing the global micro accuracies with each other, it can be seen that centralized learning requires twice as many training rounds (12 rounds equals 1200 local episodes) compared to collaborative learning (6 rounds equals 600 local episodes). Exactly the same relationship can also be observed for ROOTKIT_BDVL. Overall, based on experiment 1.2, it can be concluded that centralized and federated training both work well. However, the training time required to reach convergence is about 50% lower in the federated setting, making it faster than centralized setting training.

The learning curve of the individual client from experiment 2.1 in Figure 5.8 shows how average received return (blue) over the last 10 episodes evolves during training. A clearly positive correlation can be seen in how the average return increases with the total number of trained episodes. Furthermore, it additionally becomes evident that the average return also increases with decreasing epsilon (red), meaning that exploitation is slowly favored over exploration.

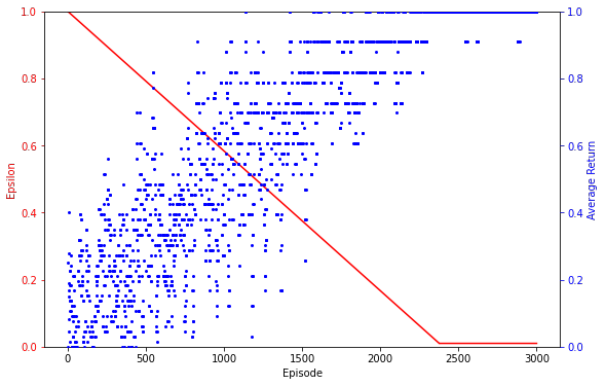


Figure 5.8: Learning Curve of Single Client from Experiment 2.1

Establishing Time Estimation Limits

In order to translate episodes trained into time estimates, an upper and lower limit for the time it takes to run a single episode has to be established. In the chosen application scenario, an episode consists of at least one but up to maximally four individual MTD

deployments. The script used to record the current status of the local host device runs for 5 seconds. In an episode with n MTD deployments, this monitoring must be carried out exactly n+1 times. The average time that the n MTD deployments within one episode require has to be added on top. Since individual MTD technologies (such as the rootkit sanitizer) only run for a very short time (a few ms) while others potentially run for a very long time, for example several seconds or minutes (ransomware trapping), this time is more difficult to quantify. Upper and lower time limits could be estimated, but for the sake of simplicity it is assumed that each individual MTD deployment is allowed to use 5 seconds for its moving phase of before the afterstate monitoring starts to run. Thus, the time required for running through an episode with n steps (Equation 5.4) can be calculated as the monitoring time (Equation 5.2) plus the total time required for MTD execution (Equation 5.3)

$$t_{monitoring}(n) = (n+1) * 5s \tag{5.2}$$

$$t_{defense}(n) = n * 5s \tag{5.3}$$

$$t_{episode}(n) = (2n+1) * 5s \tag{5.4}$$

When assuming that the average episode completed after at least one, but no more than two MTD deployments, the average episode length becomes approximately 1.5. This allows to estimate the average time required to run through an average episode according to Equation 5.4 as $(2 * 1.5 + 1) * 5s = 30$ seconds. For example, in experiment 2.2 a total of 600 training episodes are needed to exceed the desired attack mitigation performance of 99%. This would translate to an online training time of around 300 minutes or 5 hours. This episode-to-time conversion factor can also be applied to the results of other experiments.

5.1.3 Experiment 1.3

The experiments in this section are supposed to evaluate the effects that different data splits between clients might have on the training and the performance of the final agent. Scientific literature indicates that FedAvg has particular problems when dealing with non-identical and even disjoint sets of classes (i.e. client-exclusive classes) [81]. Various assumptions can be made for the training - for example that the data is either IID or non-IID. A mild case of non-IID data would be if every client would face each attack with a slightly different frequency. One of the most extreme non-IID cases would be if each individual client only saw a disjoint set of attacks during training as shown in Figure 4.8. It would be desirable and interesting do to a full sweep of $MID \times WCS \in [0,1] \times [0,1]$ and then display global model performance after the last training round in a 2d heat map. However, in a multi class classification scenario it is not feasible to construct such a full sweep due to the complexity of the calculation formulas for MID and WCS. However, it is possible to carry out two separate sweeps, one over MID and one over WCS.

Both micro and macro accuracy were used for the evaluation. The macro accuracy Equation 5.5 computes the accuracy class by class and takes the average, which is why it is also called mean class accuracy. The micro accuracy is simply computes according to Equation 5.1 as the ratio of correct predictions to total sample size.

$$Macro\ Accuracy\ =\ \frac{1}{|C|}\sum_{c=1}^{C}\frac{1}{|x_i : y_{pred}(x_i) = c|}\sum_{x_i:y_{pred}(x_i)=c} I(y_{true}(x_i) = c) \tag{5.5}$$

For the MID and the WCS sweep, 11 different sample distributions were created with different MID or different WCS values. Only one metrics was varied at a time while the other one was kept constant. For each of the 11 different MID and WCS distributions, the entire system comprising 10 clients, was trained for a combined total of 30.000 episodes in order to record the final accuracy. The results of the two sweeps consisting of final micro and macro test accuracy at the end of the training can be found in Figure 5.9 & 5.10.

Experiment 1.3.1 - MID Sweep

Based on sample distribution scenario 2, 11 globally unbalanced data splits of different strengths were created with MID values ranging approximately from 0 to 1. Since each client received the same class sample distribution, the data split is locally balanced with a WCS value of 1. As can be seen in Figure 5.9, a good final test performance is still achieved for MID values of less than 0.7. The final performance is significantly worse, for data distributions with a global imbalance of MID > 0.7. Therefore these should be avoided, although such sample distributions are rather unrealistic in the chosen application scenario.

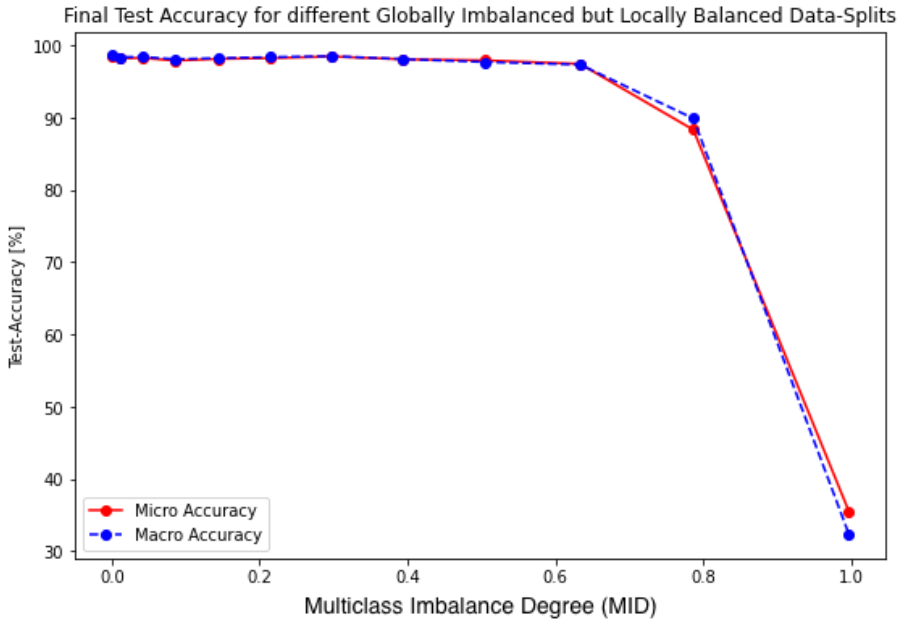


Figure 5.9: Sweep over multiple globally imbalanced data distributions

Experiment 1.3.2 - WCS Sweep

Based on sample distribution scenario 2, 11 locally unbalanced data splits of different strengths were created with WCS values ranging approximately from 0.7 to 1. Since it was ensured that each attack is seen with equal frequency on a global scale, the data split is globally balanced with a MID value of 0. As can be seen in Figure 5.10, a good final test performance is still achieved for WCS values greater than or equal to 0.8. The final performance is significantly worse, for data distributions with a local imbalance of WCS $<$ 0.8. Although, such sample distributions could occur in the chosen application scenario, it is still safe to assume that prototype 01 has a certain robustness against locally skewed sample distributions.

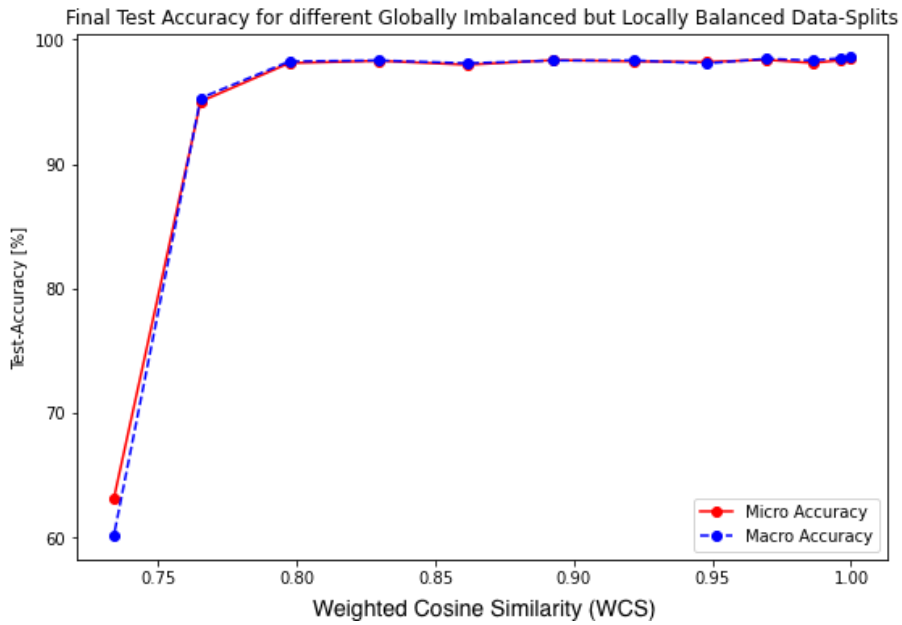
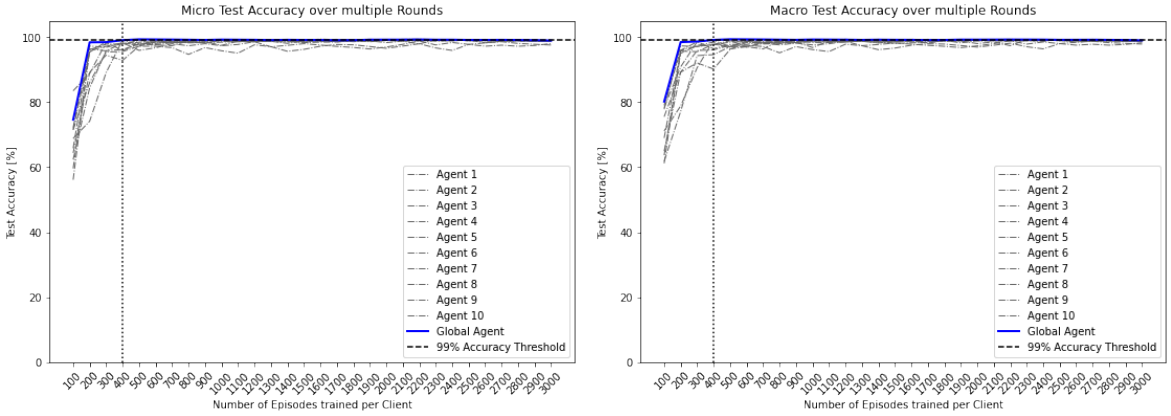


Figure 5.10: Sweep over multiple locally imbalanced data distributions

5.1.4 Experiment 1.4

The three different client exclusive sampling probabilities from scenario 5 were used for this experiment and goal was to check how well prototype 01 can handle such sampling probabilities. The individual graphs show the test performance of the global agent as well as the individual local agents after each training round. The total 30.000 state samples were distributed among 10 local agents and used to train them for 30 rounds of 100 episodes each. As shown in Figure 5.11, the training with weak client exclusive sampling still converges approximately as fast as training with uniform distribution.

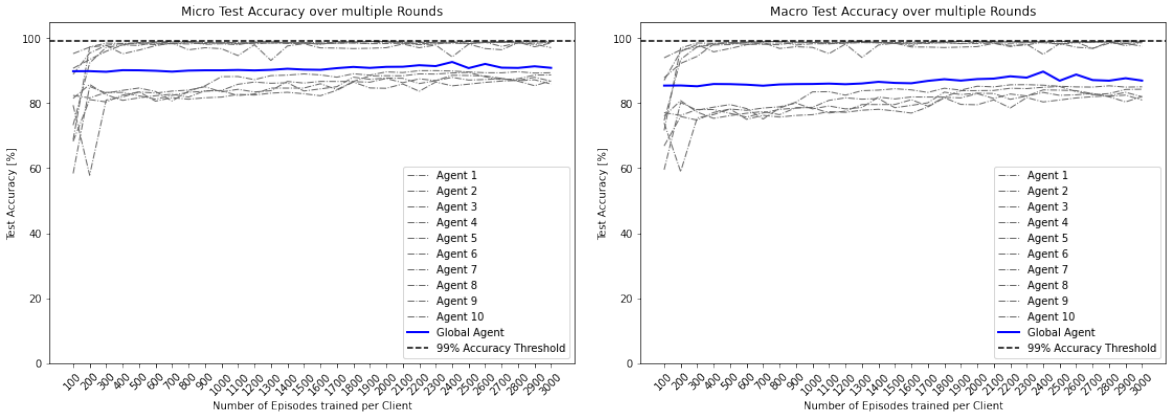
Figure 5.11: Experiment 1.4.1 Results
(Weak Client Exclusive Sampling Probabilities)



(a) Micro Test Accuracy (b) Macro Test Accuracy

The training with two client exclusive attacks as shown in Figure 5.12 does not reach the accuracy threshold of 99%, but stays around 90% attack mitigation performance. Interestingly, the test performances of the individual agents in the diagram form a corridor around those of the global agent. Individual local agents achieve better attack mitigation performances than the global agent, while others achieve worse ones.

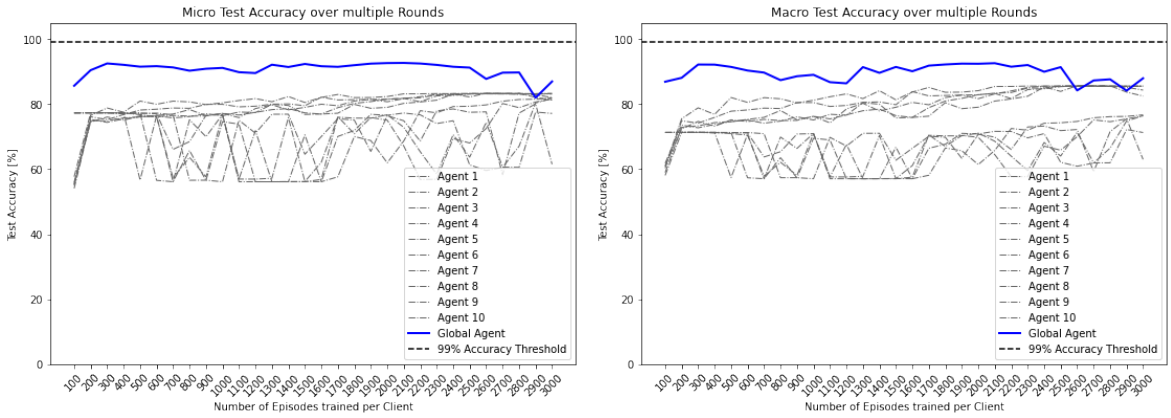
Figure 5.12: Experiment 1.4.2 Results
(Medium Client Exclusive Sampling Probabilities)



(a) Micro Test Accuracy (b) Macro Test Accuracy

In contrast to the two previous sub-experiments, when trained on strong client exclusive sampling probabilities (i.e. 3 client exclusive classes), the system does not even achieve a constant attack mitigation performance of 90%. As shown in Figure 5.13, there is a much larger fluctuation in test accuracy at both the local and global agent level.

Figure 5.13: Experiment 1.4.3 Results
(Strong Client Exclusive Sampling Probabilities)



(a) Micro Test Accuracy

(b) Macro Test Accuracy

Nevertheless, it can be said that the proposed system can handle client exclusive classes and attacks to a certain degree. In case of client distinct classes, as in scenario 5.3 (see Figure 4.8), the performance drops significantly and the training process does not converge. A setting with such a class distribution should therefore be avoided.

5.2 Prototype 02

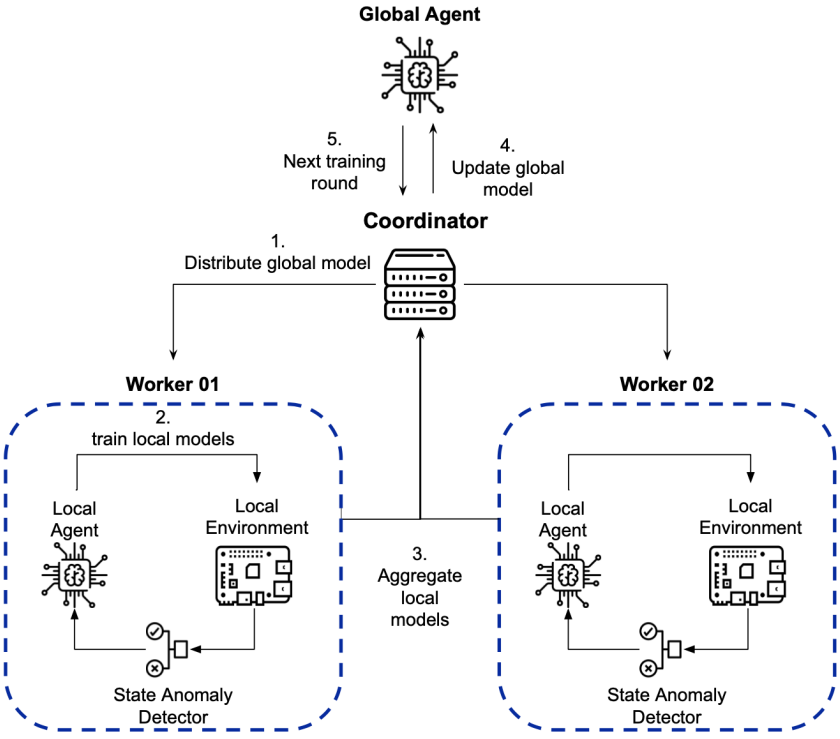


Figure 5.14: Prototype 02 / Federated Architecture

In prototype 01, where each state action reward sequence was generated synthetically, the reward singal was accurate with perfect certainty. In order to build a more realistic prototype, behavioural fingerprinting was used in order to generate the reward signal should based on the state S_{t+1} resulting after MTD execution. For this purpose a state anomaly detector, capable of classifying if a seen device state should be considered normal or abnormal, was added.

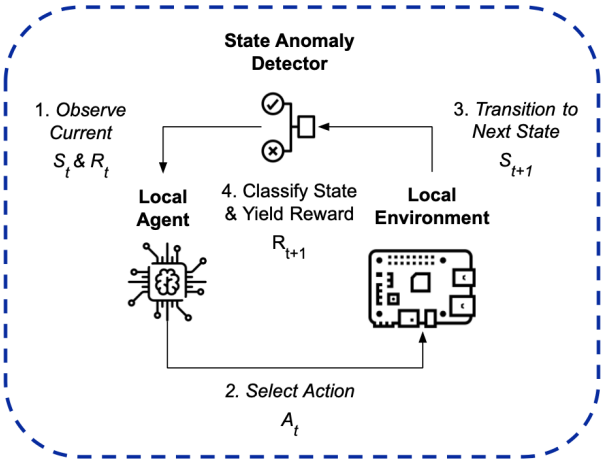


Figure 5.15: Prototype 02 / Individual Client with Anomaly Detector

As shown in Figure 5.15, the takes the following state S_{t+1} as input and has to classify if it can be considered normal. Based on the outcome of this binary classification, either a positive or negative reward is then yielded to the local agent. The MTD selecting agent is only invoked if the state anomaly detector classifies the current state of the device as abnormal/under attack. Therefore, it is sufficient to train the state anomaly detector only on normal behavioral samples, such that it learns to flag significantly differing test samples. Only if this is the case, the agent gets notified and receives the respective state sample as input in order to select an appropriate MTD in response. If the selected MTD/ action A_t was correct for state S_t, the behavior of the host devices should go back to normal (afterstate) and the episode should terminate. For this, the afterstate still has to be correctly classified as normal by the state anomaly detector (True Negative (TN)). However, it can also happen that the afterstate is incorrectly classified as non normal, which means that no reward is yielded and the episode does not terminate (False Positive (FP)). Alternatively, it may also be the case that the wrong MTD was selected for the observed state and the afterstate is incorrectly classified as normal (False Negative (FN)). FNs are considered worse than FPs, because for a FN classification the attack can continue unnoticed and the agent even stops deploying further MTDs.

For each episode the agent remembers which MTDs has already been deployed, which ensures that no MTD will be deployed twice per episode. In case all MTDs have been selected once by the agent within one episode, the episode terminates and is marked as a FP. Since it is ensured that for every attack in this controlled environment a mitigating MTD exists, it is safe to say that at some point during this episode the state anomaly detector gave a false positive classification. This means that the attack was definitely repelled, but the afterstate was incorrectly classified as not normal.

5.2.1 Evaluation of Autoencoder Models for State Anomaly Detection

In order to determine the reward R_{t+1} for the agent, the state anomaly detection must assess whether state S_{t+1} after MTD execution can be considered normal. If this is the case, then the agent receives a positive reward for his action selection and a negative one otherwise. Such a task is also called novelty detection, since it must be decided if a yet unseen sample deviates significantly from the known "normal" state. In other words, the model has to classify whether a test sample deviates significantly from the seen training samples.

This works uses a generative unsupervised model called AutoEncoder. The model architecture, as shown in Figure 5.16, is based on a neural network with an equal number of input and output neurons and a bottleneck in the middle. The objective is to learn a compact latent representation of the input space. The encoder compresses the high-dimensional input in order to fit through the central bottleneck and the decoder tries to reconstruct the original input from the compressed representation [35].

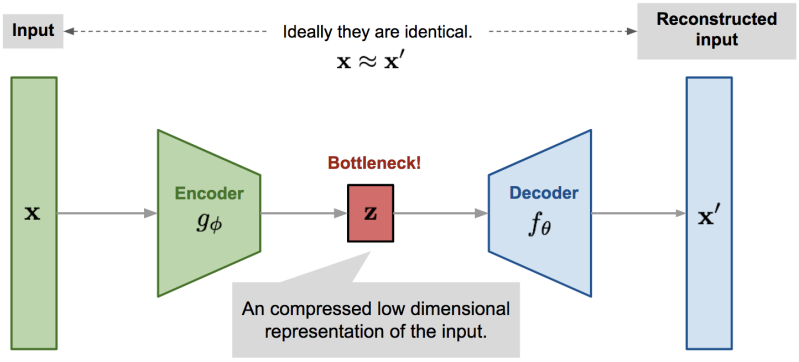


Figure 5.16: Autoencoder Architecture [74]

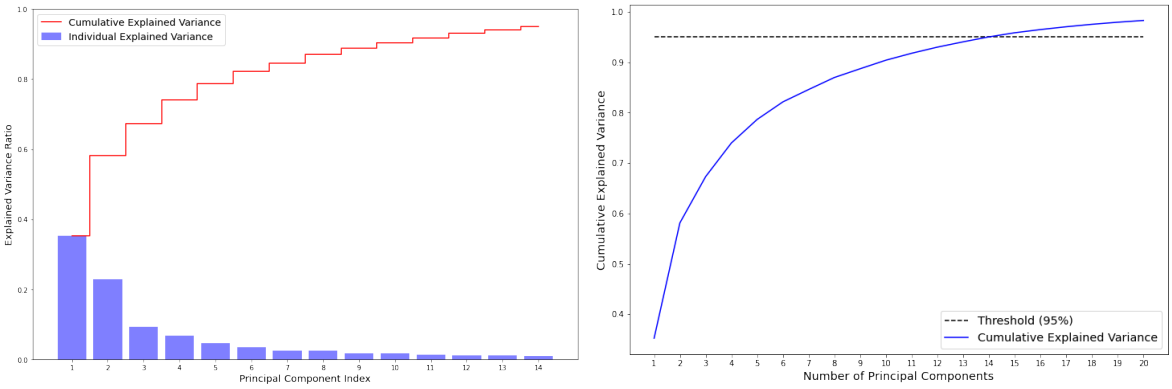
The AutoEncoders training objective is trying to minimize the reconstruction error. According to one-class classification, the model is only shown normal state samples and the encoder learns to compress them and the decoder to reconstruct them from the latent representation. However, an unseen abnormal state sample will be different from a normal transaction. The idea behind the anomaly detection is that the AutoEncoder has learned to process normal state samples better than unseen abnormal ones [35].

Anomalies are detected by comparing the magnitude of the reconstruction loss with a baseline reconstruction loss, gathered on a holdout validation set. The model was not explicitly trained to compress & reconstruct abnormal samples, and hence the reconstruction error is expected to be higher. If a samples reconstruction loss deviates from the baseline for more than n standard deviations, it is considered an anomaly [35]. Since incorrect classifications of the state after MTD execution by the state anomaly detection have a negative impact on the overall performance, the anomaly detection is very important.

Approximating the bottleneck size

In this case the bottleneck size might be the most important training hyperparameter. If it is chosen too small, not enough variance is retained to fully reconstruct the compressed information. If chosen too large, the training objective might be too easy to reach for the AutoEncoder and the information is not compressed enough. In order to get an initial approximation for a suitable bottleneck size, a Principal Component Analysis (PCA) was performed on data set 1 (Figure 5.17). The Figure 5.17a on the left shows the individually explained variance of each principal component, while Figure 5.17b displays the cumulative explained variance of the first n principal components. Most interestingly, it is shown that already 14 of the 85 features can explain 95% of the variance in the data set.

Figure 5.17: Principal Component Analysis (PCA) Result



(a) Individual Explained Variance

(b) Cumulative Explained Variance

For the anomaly detection any sample reflecting abnormal behavior is considered positive (labeled with 1) and any sample of normal behavior is considered negative (labeled with 0). When performing a class wise evaluation as shown in Table 5.2 and for example only passing positive samples, then no false positives can exist by definition, since no sample with negative label has been passed during this part of the evaluation process. Therefore, precision, recall and f-1 score were only computed for the whole test data set. Although True Positive (TP) and False Negative (FN) classifications both occur, the recall per class is not explicitly given, since it equals the class wise accuracy. The 60.000 samples from data set 01 were split equally into two subsets, one for training the anomaly detection and a second one for training the RL agents.

Trying an Initial Model for the AutoEncoder based State Anomaly Detection

An initial autoencoder model, which is supposed to be used for anomaly detection, was trained using the hyperparameters mentioned by Timo Schenk [68]. For optimization, SGD with learning rate 1e-4 and momentum 0.9 was used. The model was trained for a maximum of 100 epochs with a batch size of 64. 2.5 standard deviations were used as the reconstruction error threshold. Considering the fact that the state anomaly detection is

crucial for the performance of the entire system, the detection accuracy of 80.61% as shown in table Table 5.2 is not yet sufficient. Therefore, in the following, several hyperparameter combinations were evaluated and compared.

Table 5.2: Initial Anomaly Detection Rate

Behavior	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**#Samples**
NORMAL	99.66%	x	x	x	7382
ROOTKIT_BDVL	21.98%	x	x	x	2844
ROOTKIT_BEURK	100.00%	x	x	x	3730
CNC_BACKDOOR_JAKORITAR	100.00%	x	x	x	2149
CNC_THETICK	100.00%	x	x	x	3834
CNC_OPT1	12.75%	x	x	x	2893
CNC_OPT2	53.52%	x	x	x	2046
RANSOMWARE_POC	100.00%	x	x	x	4624
GLOBAL	80.61%	99.85%	74.26%	85.17%	29502

Finding the best Hyperparameter Combination for the State Anomaly Detection

Since the anomaly detection performance of the initial model (Table 5.2) was not yet sufficient, a hyperparameter search was carried out. The full set of hyperparameter combinations, was inspired by Keshtkaran and Pandarinath work [34] and can be found in Table 5.3. Sklearns GridSearchCV with five fold cross validation was used in order to try all different hyperparameter combinations. Early stopping with a patience of five was used in order to prevent overfitting. Each trained model was scored using the mean validation accuracy. The result of the hyperparameter search is shown in Table 5.4.

Table 5.3: Hyperparameter Combinations for AutoEncoder State Anomaly Detection

Model Hyperparameter	
NR_NEURONS_PER_LAYER	(64, 32), **(64, 16)**, (64, 8)
ACTIVATION_FUNCTION	Sigmoid, Tanh, ReLU, ELU, GELU
BATCH_NORMALIZATION	False, **True**
Optimization Hyperparameter	
LOSS_FUNCTION	MAE, MSE, RMSE
OPTIMIZER	SGD, Adam, **RMSprop**
LR	1e-3, **1e-4**, 1e-5
MOMENTUM	0.9
L2_REGULARIZATION	1e-1, **1e-2**, 1e-3, 1e-4
CV	5
MAX_EPOCHS	100
EARLY_STOPPING	False, **True (patience=5)**
BATCH_SIZE	**32**, 64
Prediction Hyperparameter	
N_STD	1, **2**, 3

Table 5.4: Highest Anomaly Detection Rate

rank	**mean_validation_accuracy**	**n_hidden_1**	**n_hidden_2**	**n_stds**	**optimizer**	**loss_function**
1	99.67%	64	16	2	RMSprop	RMSE
2	99.66%	64	32	2	Adam	RMSE
3	99.64%	64	8	3	RMSprop	MSE
4	99.63%	64	32	2	RMSprop	RMSE
5	99.43%	64	16	2	Adam	RMSE
6	99.35%	64	8	2	RMSprop	RMSE
7	99.34%	64	16	3	RMSprop	RMSE
8	99.33%	64	8	2	Adam	MSE
9	99.29%	64	16	2	RMSprop	MSE
10	99.13%	64	32	1	Adam	RMSE

Since the best hyperparameter combination shown in Table 5.4 only achieved 92% test accuracy, the second best hyperparameter combination (with rank 2) was also evaluated. As shown in Table 5.5,this model achieved a 99.88% evaluation accuracy and was therefore used during the following experiments.

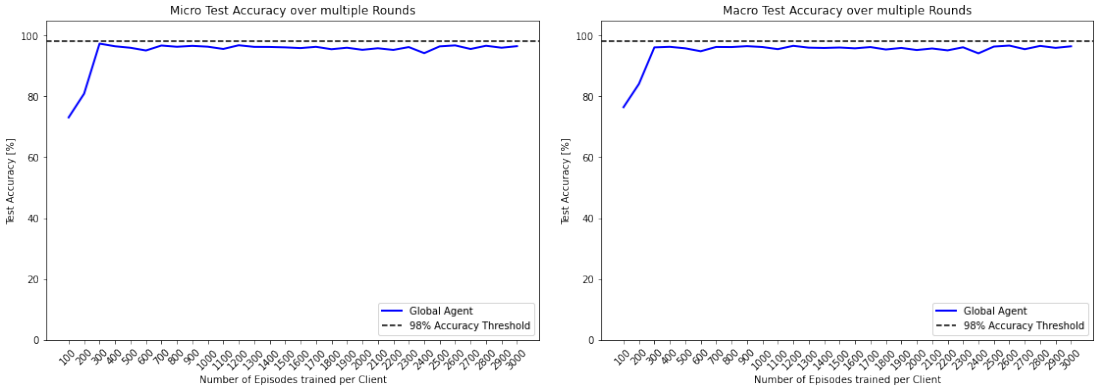
Table 5.5: #2 Hyperparameter Evaluation

Behavior	Accuracy	Precision	Recall	F1-Score	#Samples
NORMAL	99.54%	x	x	x	7382
ROOTKIT_BDVL	100.00%	x	x	x	2844
ROOTKIT_BEURK	100.00%	x	x	x	3730
CNC_BACKDOOR_JAKORITAR	100.00%	x	x	x	2149
CNC_THETICK	100.00%	x	x	x	3834
CNC_OPT1	100.00%	x	x	x	2893
CNC_OPT2	100.00%	x	x	x	2046
RANSOMWARE_POC	100.00%	x	x	x	4624
GLOBAL	99.88%	99.85%	100.00%	99.92%	29502

5.2.2 Experiment 2.1

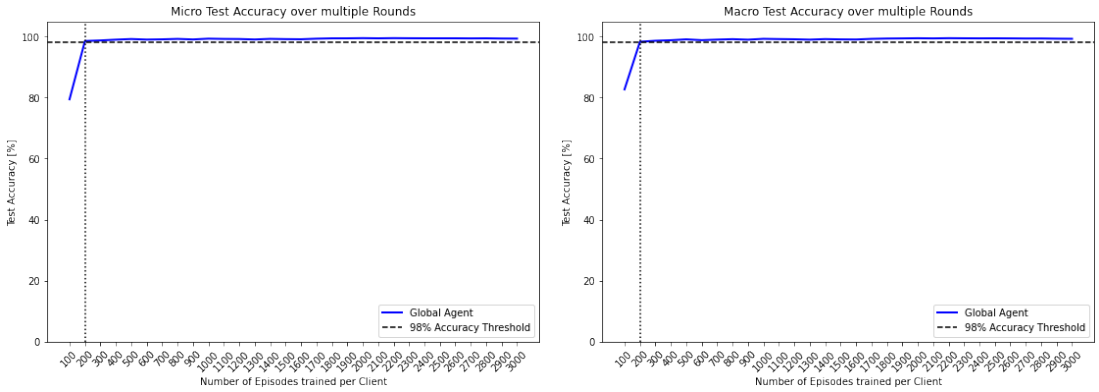
This experiment is about reproducing the results from experiment 1.2 for the second prototype for analysis and comparison. Therefore, this experiment was carried out in a completely analogous way with the using the same hyperparameters and unit sample class distribution. Since the results turned out to be very similar, a separate curve is not given for each individual attack, but only the global micro and macro test accuracy are plotted in Figure 5.18 and Figure 5.19.

Figure 5.18: Prototype 02 Cenralized Training Baseline
(Attack Mitigation Performance over multiple Training Rounds)



(a) E2.1.1 Micro Test Accuracy (b) E2.1.2 Macro Test Accuracy

Figure 5.19: Prototype 02 Federated Training Baseline
(Attack Mitigation Performance over multiple Training Rounds)



(a) E2.1.2 Micro Test Accuracy (b) E2.1.2 Macro Test Accuracy

The fact that the graphs of the micro and macro accuracy are almost identical makes the analysis easier, since only the graph of the micro test accuracy needs to be considered here. as can be seen in Figure 5.18a, the test accuracy reaches a maximum 97.34% and never exceeds the 99% threshold. As shown in Figure 5.19a, the federated version of this experiment achieves a maximum test accuracy of 99.49%. Both sub-experiments achieve good accuracy after 300 episodes, but they differ by more than 2 percent. This means that the collaboratively learned policy is superior.

5.2.3 Experiment 2.2

This experiment follows the experimental setup (used hyperparameters and sample distributions) of experiment 1.3. Therefore, for a more detailed understanding, please refer to experiment 1.3, because only the results will be explained here.

Experiment 2.2.1 - MID Sweep

As shown in Figure 5.20, the final test performance drops significantly for data sets with a global imbalance of MID > 0.6. Thus, it be claimed that prototype 02 is reasonably robust against global class imbalance.

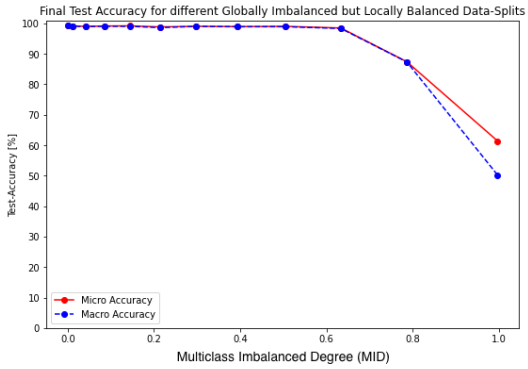


Figure 5.20: Sweep over multiple globally imbalanced data distributions

Experiment 2.2.2 - WCS Sweep

As shown in Figure 5.21, the final test performance drops significantly for data sets with a local imbalance of WCS < 0.7. Thus, it be claimed that prototype 02 is also reasonably robust against local class imbalance.

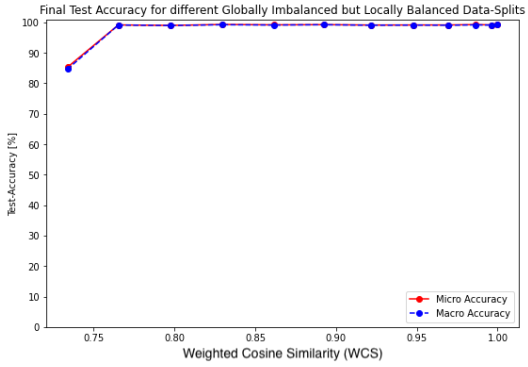


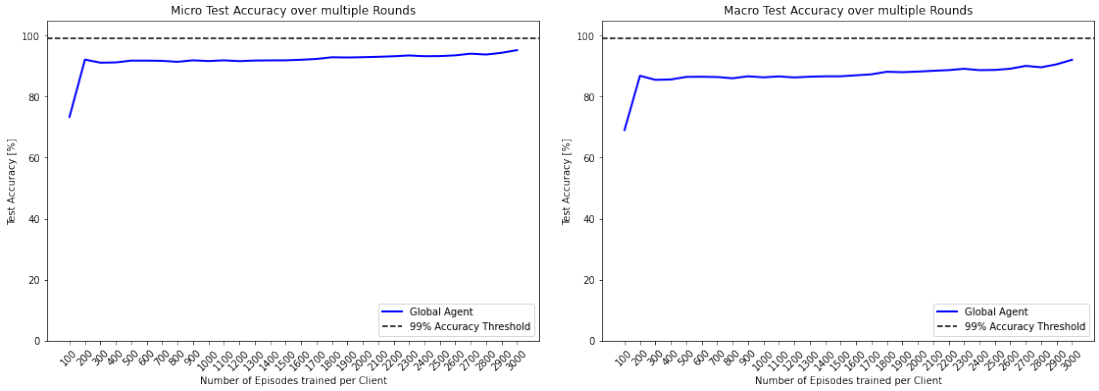
Figure 5.21: Sweep over multiple locally imbalanced data distributions

Due to the results from experiment 2.2, it is safe to claim that prototype 02 also has a certain robustness against locally and globally imbalanced data distributions.

5.2.4 Experiment 2.3

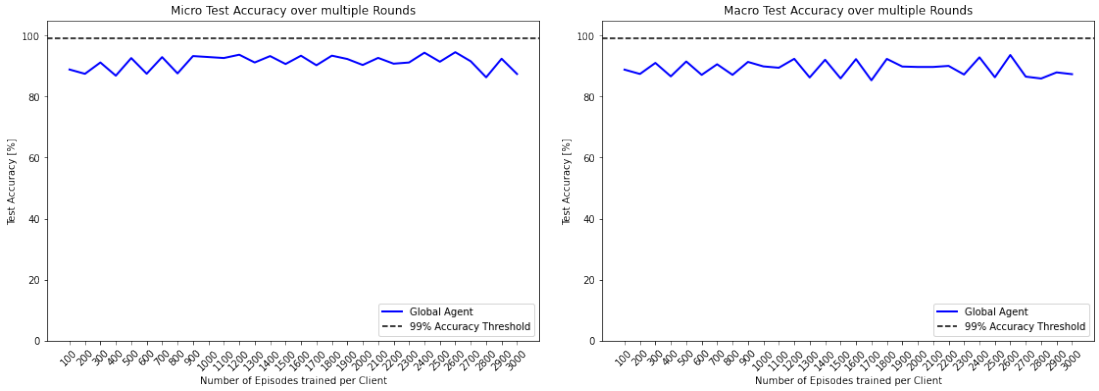
The setup of this experiment is very similar to Experiment 1.4. Therefore it is recommended to read this section before proceeding. The objective of this experiment is also to investigate how well prototype 02 (including the state anomaly detection) can handle different client exclusive sampling probabilities. In contrast to Experiment 1.4, the graphs do not show the individual test performances of the each local agent. The graph for the training with the weak client exclusive sampling probabilities was omitted because it looks like Figure 5.11 and therefore does not provide any new information. As shown in Figure 5.22, the overall system is still able to handle medium client exclusive sampling probabilities (2 client exclusive classes each). The test accuracy is continuously increasing slightly and one does not know if the maximum has been reached after 3000 episodes. On the other hand, Figure 5.23 shows that prototype 02 also has its problems with stronlgy client exclusive sampling probabilities. The test accuracies alternate strongly and the training does not seem to converge.

Figure 5.22: Experiment 2.3.2 Results
(Medium Client Exclusive Sampling Probabilities)



(a) Micro Test Accuracy (b) Macro Test Accuracy

Figure 5.23: Experiment 2.3.3 Results
(Strong Client Exclusive Sampling Probabilities)



(a) Micro Test Accuracy (b) Macro Test Accuracy

5.3 (Hardware) Prototype 03

In contrast to the software based prototypes 01 and 02, this the only hardware based prototype that has been built for this project. As shown in Figure 5.24, the setup comprises four ElectroSense sensors, each consisting of a RPi paired with a suitable RF antenna. The idea behind this prototype is to get one step closer to a full online version of the collaborative FedRL system for MTD selection. This prototype can be used for several different purposes in the future, for example to collect an even more realistic training data set from a homogeneous set of devices instead of collecting it from a single isolated device. Alternatively, the offline trained policy can be transferred to this context in order to examine how well this policy transfer works and how much fine tuning would still be required.

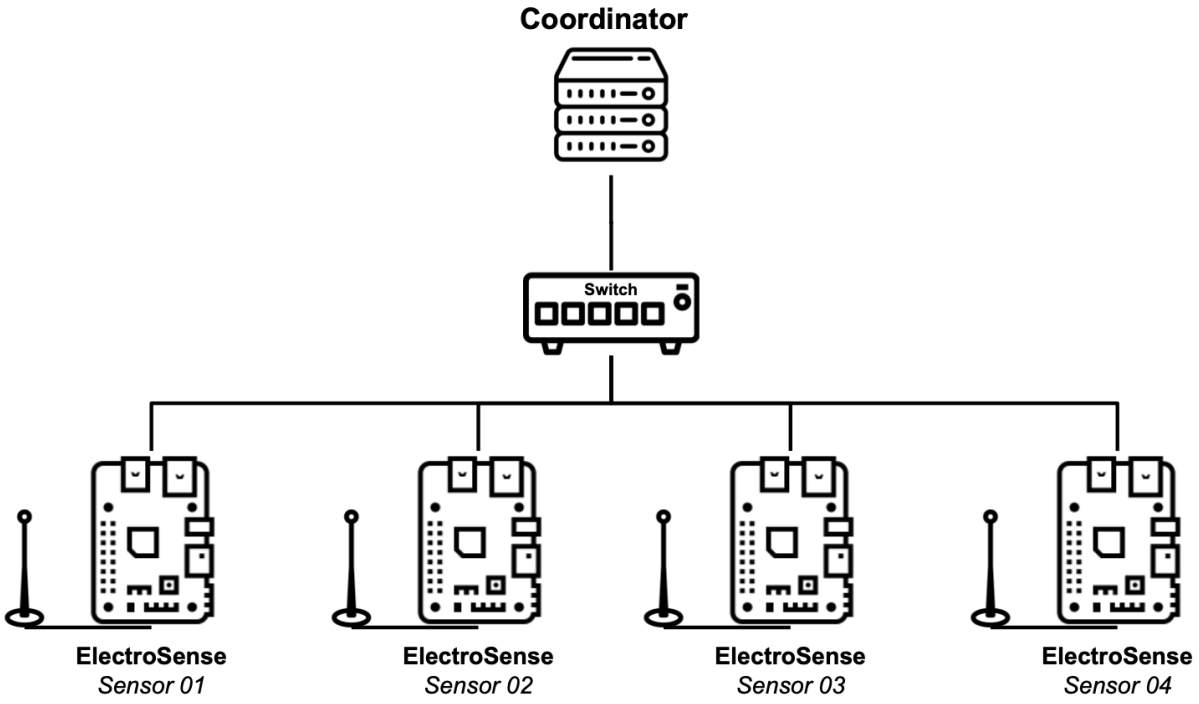


Figure 5.24: Prototype 03 (Schematic Representation)

5.3.1 Setup

As shown in Figure 5.24 & Figure 5.25, four Raspberry Pis (RPis) together with an Radio Frequency (RF) antenna combined were set up in the ElectroSense network. They are linked via Ethernet to a 5-port switch that is connected to the Internet. A fifth, older generation RPi 2 takes on the role of the coordinating server. The current status of the sensor according to the state anomaly detector can be signaled by a red and a green Light Emitting Diode (LED). For this purpose, the LEDs need to be connected to the RPi's GPIO pins, so that they can draw power and can be controlled. In order to protect the LEDs from excessive current, a pre-resistor of 220Ω for green and 330Ω for the red LEDs need to be used. The resistor needs to be placed in sequence with the LEDs.

Figure 5.25: Prototype 03 (Implemented Configuration)

5.3.2 Comparing different FL-frameworks

The first prototype simulated each FL worker locally as individual threats each running on a physically separate CPU-core. This can be considered realistic for centralized pre-training of the agent, but less so when aiming to implement a fully online prototype. An important design decision that has to be taken is how to optimally implement and orchestrate the federated learning process. There are several approaches to this. A custom version of the federation learning could be implemented, or existing FL frameworks could be used.

The goal is to train a global deep neural network model (based on PyTorch) in a federated fashion on 5 RaspberryPis. In the search for the best federated learning framework to support this endeavor, the following seven FL have been identified as viable options. It remains unclear, however, which one would be best suited for federated training of PyTorch models. (still needs a finishing touch)

Framework	**Explicit support for RL/ DQN? (✓ . . . Yes, o . . . Unkown, x . . . No)**	**Developing Organization**
1. TensorFlow Federated	✓	Google
2. IBM federated-learning	✓	IBM
3. OpenFL	✓	Intel
4. PySyft	o	OpenMined
5. FATE	o	Webank
6. Flower	o	Adap
7. Substra	x	

5.3.3 Constraints

Since the given ElectroSense OS image does not grant easy write access, a new image would have to be created. Due to a lack of time, this prototype could not be extended further than described until here.

Chapter 6

Discussion

6.1 Summary & Conclusions

This work showcases that it is possible to collaboratively learn an MTD selection policy in a real world IoT scenario like ElectroSense. So far, it is one of the few works that use a combination of FL and RL in a real world scenario in order to boost IT security.

- Experiment 1.2 showed that collaboratively learning a selection policy can achieve a good calculatory speedup of about 50% over centralized learning.
- Experiment 2.1 proved that collaboratively learning an MTD selection policy can potentially outperform a centralized approach. The test attack mitigation performance of the multi-agent system consistently exceeded the one of the single-agent system by more than 2%.
- Experiment 1.3 & 2.2 revealed that the proposed FedRL system has a certain robustness against local and global class imbalance. The performance of the entire system only begins to degrade if a strongly locally or globally imbalanced distribution is used.
- Experiment 1.4 & 2.3 demonstrated that learning a common MTD policy still works for some distributions with client exclusive classes.

The main goals formulated at the beginning of this thesis have been largely achieved. However, more work and research is needed as the good anomaly detection and attack mitigation performance on data set 01 has not yet been achieved on the more realistic and fine granular data set 02. This would be an important next step on the way to a collaborative online system for MTD selection. A hyperparameter search was performed for four different anomaly detection models (OneClassSVM, IsolationForest, LocalOutlierFactor, and Autoencoder) to identity the most suitable one, together with the best combination of hyperparameter. Unfortunately, none of the four models could achieve an anomaly detection accuracy of over 60% considering all tested hyperparameter combinations. Further limitations that have been noticed in the course of this thesis and their implications for future work are discussed in the next section.

6.2 Limitations & Future Work

In the following, the limitations of this work, that either represent a deviation from the originally defined goals or have arisen in the course of this work, are described. These open problems or newly found tangents can motivate future research. It must be noted that data set 01 consists of only 8 behaviors that can be separated particularly well from one another and therefore work well for anomaly detection and RL agent. As the evaluation of the anomaly detection on data set 2 has shown, training using a more realistic data set is much more difficult.

Although it was shown that FedAvg in this scenario is reasonably robust with respect to locally and globally imbalanced class distributions, there are still many other aggregation functions for which it would also be very interesting to try them out.

The MTDs are static and do not yet adapt to the attacks, which poses a problem if a new attack arises or an attacker is smart enough to understand and evade the MTD. For some mtds, such as ransomware trapping, it is crucial that they are started at the right moment. Timing aspects, such as when an mtd is best to be deployed, have not been taken into account so far.Another specific problem is the fact that the cnc ip shuffle MTD can mitigate 4 out of 7 possible attacks and, therefore, is actually too powerful in this scenario. A trivial but stupid agent could always deploy this MTD first since it is a good first guess.

It also has to be admitted that so far every experiment was running in a virtually simulated environment and has not yet been trained online on a real set of host devices. Trying to do this would also result in a number of other technical problems and hurdles, such as selecting the right FL framework. All in all, it can be said that it is extremely difficult to build a working online FL framework based on DQNs for the collaborative selection of moving target defense mechanisms as part of a master's thesis. The number of hyperparameters for such a system is very high (>20) and there is a multitude of small technical details that need to be considered, making the system challenging to build and optimize. Since no functioning online prototype exists yet, it was not possible to examine how well the transfer of the offline pre-trained policy would work.

Bibliography

[1] Muhammad Shoaib Akhtar and Tao Feng. "Malware Analysis and Detection Using Machine Learning Algorithms". In: *Symmetry* 14.11 (2022). ISSN: 2073-8994. DOI: 10.3390/sym14112304. URL: https://www.mdpi.com/2073-8994/14/11/2304.

[2] Massimiliano Albanese, Sushil Jajodia, and Sridhar Venkatesan. "Defending from Stealthy Botnets Using Moving Target Defenses". In: *IEEE Security & Privacy* 16 (Feb. 2018), pp. 92–97. DOI: 10.1109/MSP.2018.1331034.

[3] Hafiz Asif, Mohamed Karim, and Firdous Kausar. "Federated Learning and its Applications for Security and Communication". In: *International Journal of Advanced Computer Science and Applications* 13 (Jan. 2022). DOI: 10.14569/IJACSA.2022.0130838.

[4] Jan von der Assen et al. *A Lightweight Moving Target Defense Framework for Multi-purpose Malware Affecting IoT Devices*. 2022. arXiv: 2210.07719 [cs.CR].

[5] David Beazley. *Understanding the Python Global Interpreter Lock (GIL)*. 2010.

[6] Subrato Bharati et al. "Federated learning: Applications, challenges and future directions". In: *International Journal of Hybrid Intelligent Systems* 18.1-2 (May 2022), pp. 19–35. DOI: 10.3233/his-220006. URL: https://doi.org/10.3233%2Fhis-220006.

[7] Gui-lin Cai et al. "Moving target defense: state of the art and characteristics". In: *Frontiers of Information Technology and Electronic Engineering* 17 (Nov. 2016), pp. 1122–1153. DOI: 10.1631/FITEE.1601321.

[8] Xinzhong Chai et al. "DQ-MOTAG: Deep Reinforcement Learning-based Moving Target Defense Against DDoS Attacks". In: July 2020, pp. 375–379. DOI: 10.1109/DSC50466.2020.00065.

[9] Jin-Hee Cho et al. "Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense". In: *CoRR* abs/1909.08092 (2019). arXiv: 1909.08092. URL: http://arxiv.org/abs/1909.08092.

[10] Ankur Chowdhary et al. "SDN-based Moving Target Defense using Multi-agent Reinforcement Learning". In: Mar. 2021.

[11] Computerworld. *IBM und Visa wollen Bezahlen revolutionieren*. 2017. URL: https://www.computerworld.ch/business/digitalisierung/ibm-visa-bezahlen-revolutionieren-1347904.html.

[12] Yue Deng et al. "Deep Direct Reinforcement Learning for Financial Signal Representation and Trading". In: *IEEE Transactions on Neural Networks and Learning Systems* 28 (Feb. 2016), pp. 1–12. DOI: 10.1109/TNNLS.2016.2522401.

[13] Oxford Reference Dictionary. *Malware*. 2023. URL: https://www.oxfordreference.com/display/10.1093/oi/authority.20110803100129408.

[14] Taha Eghtesad, Yevgeniy Vorobeychik, and Aron Laszka. "Deep Reinforcement Learning based Adaptive Moving Target Defense". In: (Nov. 2019).

[15] ElectroSense. *Collaborative Spectrum Monitoring*. 2022. URL: https://electrosense.org.

[16] Minghong Fang et al. *Local Model Poisoning Attacks to Byzantine-Robust Federated Learning*. 2021. arXiv: 1911.11815 [cs.CR].

[17] William Fedus et al. *Revisiting Fundamentals of Experience Replay*. 2020. arXiv: 2007.06700 [cs.LG].

[18] Mohamed Amine Ferrag et al. "Federated Deep Learning for Cyber Security in the Internet of Things: Concepts, Applications, and Experimental Analysis". In: *IEEE Access* 9 (2021), pp. 138509–138542. DOI: 10.1109/ACCESS.2021.3118642.

[19] Chungang Gao and Yongjie Wang. "Reinforcement learning based self-adaptive moving target defense against DDoS attacks". In: *Journal of Physics: Conference Series* 1812 (Feb. 2021), p. 012039. DOI: 10.1088/1742-6596/1812/1/012039.

[20] Anup Ghosh. *National Cyber Leap Year Summit 2009 Co-Chairs Report*. 2009.

[21] Daniel Goldberg, Ofri Ziv, and Mor Matalon. *Operation Prowli: Monetizing 40,000 Victim Machines*. 2018. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[22] Roger Hallman et al. "IoDDoS — The Internet of Distributed Denial of Service Attacks: A Case Study of the Mirai Malware and IoT-Based Botnets". In: Apr. 2017. DOI: 10.5220/0006246600470058.

[23] Jin Hong and Dan Kim. "Assessing the Effectiveness of Moving Target Defenses Using Security Models". In: *IEEE Transactions on Dependable and Secure Computing* 13 (June 2015), pp. 1–1. DOI: 10.1109/TDSC.2015.2443790.

[24] Transforma Insights. *Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030*. 2022. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[25] Transforma Insights. *Number of Internet of Things connected devices worldwide*. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[26] Federal Bureau of Investigation. *Internet Crime Report*. 2021.

[27] Łukasz Jalowski, Marek Zmuda, and Mariusz Rawski. "A Survey on Moving Target Defense for Networks: A Practical View". In: *Electronics* 11.18 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11182886. URL: https://www.mdpi.com/2079-9292/11/18/2886.

[28] Jan Kreischer. *Federated Reinforcement Learning for Private and Collaborative Selection of Moving Target Defense Mechanisms for IoT Device Security*. 2023. URL: https://github.com/jan-kreischer/FedRL-for-IT-Sec.

[29] Quan Jia, Kun Sun, and Angelos Stavrou. "MOTAG: Moving Target Defense against Internet Denial of Service Attacks". In: *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*. 2013, pp. 1–9. DOI: 10.1109/ICCCN.2013.6614155.

[30] Jordan Cedeño. "Mitigating Cyberattacks Affecting Resource-constrained Devices Through Moving Target Defense Mechanisms". In: (2022).

[31] Peter Kálnai and Jaromír Horejší. "IoT Malware: When embedded devices flood". In: (Dec. 2015). DOI: 10.13140/RG.2.2.23928.80644.

[32] Stamatis Karnouskos. "Stuxnet Worm Impact on Industrial Cyber-Physical System Security". In: *IECON Proceedings (Industrial Electronics Conference)* (Nov. 2011). DOI: 10.1109/IECON.2011.6120048.

[33] Kernel.org. *perf: Linux profiling with performance counters*. 2023. URL: https://perf.wiki.kernel.org/index.php/Main_Page.

[34] Mohammad Reza Keshtkaran and Chethan Pandarinath. *Enabling hyperparameter optimization in sequential autoencoders for spiking neural data*. 2019. arXiv: 1908.07896 [cs.LG].

[35] Renu Khandelwal. *Anomaly Detection using Autoencoders*. 2021. URL: https://towardsdatascience.com/anomaly-detection-using-autoencoders-5b032178a1ea.

[36] Sunghwan Kim et al. "DIVERGENCE: Deep Reinforcement Learning-based Adaptive Traffic Inspection and Moving Target Defense Countermeasure Framework". In: *IEEE Transactions on Network and Service Management* PP (Jan. 2021), pp. 1–1. DOI: 10.1109/TNSM.2021.3139928.

[37] Vincent Francois Lavet et al. "An Introduction to Deep Reinforcement Learning". In: (2018).

[38] Sergey Levine et al. *End-to-End Training of Deep Visuomotor Policies*. 2016. arXiv: 1504.00702 [cs.LG].

[39] Henger Li and Zizhan Zheng. "Robust Moving Target Defense Against Unknown Attacks: A Meta-Reinforcement Learning Approach". In: *Decision and Game Theory for Security: 13th International Conference, GameSec 2022, Pittsburgh, PA, USA, October 26–28, 2022, Proceedings*. PIttsburgh, PA, USA: Springer-Verlag, 2023, pp. 107–126. ISBN: 978-3-031-26368-2. DOI: 10.1007/978-3-031-26369-9_6. URL: https://doi.org/10.1007/978-3-031-26369-9_6.

[40] Hyun-Kyo Lim et al. "Federated Reinforcement Learning for Training Control Policies on Multiple IoT Devices". In: *Sensors* 20 (Mar. 2020), p. 1359. DOI: 10.3390/s20051359.

[41] Longxin Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine Learning* 8 (1992), pp. 293–321.

[42] Ruo-Ze Liu et al. *An Introduction of mini-AlphaStar*. Apr. 2021.

[43] Heiko Ludwig et al. *IBM Federated Learning: an Enterprise Framework White Paper V0.1*. July 2020.

[44] man7.org. *top(1) — Linux manual page*. 2023. URL: https://man7.org/linux/man-pages/man1/top.1.html.

[45] Brendan McMahan. "Federated Learning from Research to Practice". In: (2019).

[46] H. Brendan McMahan et al. *Communication-Efficient Learning of Deep Networks from Decentralized Data*. 2023. arXiv: 1602.05629 [cs.LG].

[47] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (Feb. 2015), pp. 529–33. DOI: 10.1038/nature14236.

[48] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: (Dec. 2013).

[49] Viraaji Mothukuri et al. "Federated-Learning-Based Anomaly Detection for IoT Security Attacks". In: *IEEE Internet of Things Journal* 9 (May 2021), pp. 2327–4662. DOI: 10.1109/JIOT.2021.3077803.

[50] Anitta Patience Namanya et al. "The World of Malware: An Overview". In: Sept. 2018. DOI: 10.1109/FiCloud.2018.00067.

[51] Renzo Navas et al. "MTD, Where Art Thou? A Systematic Review of Moving Target Defense Techniques for IoT". In: *IEEE Internet of Things Journal* PP (Nov. 2020), pp. 1–1. DOI: 10.1109/JIOT.2020.3040358.

[52] Thanh Nguyen et al. *A New Tensioning Method using Deep Reinforcement Learning for Surgical Pattern Cutting*. Jan. 2019.

[53] Thanh Thi Nguyen and Vijay Janapa Reddi. "Deep Reinforcement Learning for Cyber Security". In: *CoRR* abs/1906.05799 (2019). arXiv: 1906.05799. URL: http://arxiv.org/abs/1906.05799.

[54] Roseline Ogundokun et al. "A Review on Federated Learning and Machine Learning Approaches: Categorization, Application Areas, and Blockchain Technology". In: *Information* 13 (May 2022), p. 263. DOI: 10.3390/info13050263.

[55] Xinlei Pan et al. "Virtual to Real Reinforcement Learning for Autonomous Driving". In: Jan. 2017. DOI: 10.5244/C.31.11.

[56] Dr. Yusuf Perwej et al. "The Internet-of-Things (IoT) Security : A Technological Perspective and Review". In: Volume 5 (Feb. 2019), Page 462–482. DOI: 10.32628/CSEIT195193.

[57] Le Phong et al. "Privacy-Preserving Deep Learning via Additively Homomorphic Encryption". In: *IEEE Transactions on Information Forensics and Security* PP (Dec. 2017), pp. 1–1. DOI: 10.1109/TIFS.2017.2787987.

[58] Prajoy Podder et al. "Review on the Security Threats of Internet of Things". In: *International Journal of Computer Applications* 176.41 (July 2020), pp. 37–45. DOI: 10.5120/ijca2020920548. URL: https://doi.org/10.5120%5C%2Fijca2020920548.

[59] Sreeraj Rajendran et al. "Electrosense: Open and Big Spectrum Data". In: *IEEE Communications Magazine* PP (Mar. 2017). DOI: 10.1109/MCOM.2017.1700200.

[60] Andrew Barto Richard Sutton. *Reinforcement Learning: An Introduction*. 2015.

[61] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2010.

[62] Paul Schaik. "Risk perceptions of cyber-security and precautionary behaviour". In: *Computers in Human Behavior* (May 2017).

[63] Kaspersky Security. *How to detect & prevent rootkits?* 2023. URL: https://www.kaspersky.com/resource-center/definitions/what-is-rootkit.

[64] Kaspersky Security. *What is Cryptojacking and how does it work?* 2023. URL: https://www.kaspersky.com/resource-center/definitions/what-is-cryptojacking.

[65] Sailik Sengupta and Subbarao Kambhampati. "Multi-agent Reinforcement Learning in Bayesian Stackelberg Markov Games for Adaptive Moving Target Defense". In: (July 2020).

[66] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961.

[67] Csaba Szepesvari. *Algorithms for Reinforcement Learning*. 2009.

[68] Timo Schenk. *Optimizig MTD Deployment on IoT Devices using Reinforcement Learning*. 2022.

[69] Vicenç Torra. "A Systematic Construction of Non-i.i.d. Data Sets from a Single Data Set: Non-Identically Distributed Data". In: *Knowl. Inf. Syst.* 65.3 (Nov. 2022), pp. 991–1003. ISSN: 0219-1377. DOI: 10.1007/s10115-022-01785-3. URL: https://doi.org/10.1007/s10115-022-01785-3.

[70] Lawrence Trautman and Peter Ormerod. "Wannacry, Ransomware, and the Emerging Threat to Corporations". In: *SSRN Electronic Journal* (Jan. 2018). DOI: 10.2139/ssrn.3238293.

[71] Aashma Uprety and Danda B Rawat. "Reinforcement Learning for IoT Security: A Comprehensive Survey". In: *IEEE Internet of Things Journal* PP (Nov. 2020), pp. 1–1. DOI: 10.1109/JIOT.2020.3040957.

[72] Vishnu Vijayan. *Deep Reinforcement Learning: Value Functions, DQN, Actor-Critic method, Back-propagation through stochastic functions*. 2020. URL: https://medium.com/@vishnuvijayanpv/deep-reinforcement-learning-value-functions-dqn-actor-critic-method-backpropagation-through-83a277d8c38d.

[73] Lixu Wang et al. *Addressing Class Imbalance in Federated Learning*. 2020. arXiv: 2008.06217 [cs.LG].

[74] Lilian Weng. *From Autoencoder to Beta-VAE*. 2018. URL: https://lilianweng.github.io/posts/2018-08-12-vae/.

[75] Chenguang Xiao and Shuo Wang. *An Experimental Study of Class Imbalance in Federated Learning*. Sept. 2022.

[76] Xiaoyu Xu et al. "Moving target defense of routing randomization with deep reinforcement learning against eavesdropping attack". In: *Digital Communications and Networks* 8 (Jan. 2022). DOI: 10.1016/j.dcan.2022.01.003.

[77] Qiang Yang et al. "Federated Machine Learning: Concept and Applications". In: *ACM Transactions on Intelligent Systems and Technology* 10 (Jan. 2019), pp. 1–19. DOI: 10.1145/3298981.

[78] Zhuoran Yang, Yuchen Xie, and Zhaoran Wang. "A Theoretical Analysis of Deep Q-Learning". In: *CoRR* abs/1901.00137 (2019). arXiv: 1901.00137. URL: http://arxiv.org/abs/1901.00137.

[79] Seunghyun Yoon et al. "DESOLATER: Deep Reinforcement Learning-based Resource Allocation and Moving Target Defense Deployment Framework". In: *IEEE Access* PP (Apr. 2021), pp. 1–1. DOI: 10.1109/ACCESS.2021.3076599.

[80] Igal Zeifman. *Breaking down Mirai: An IoT DDoS botnet analysis*. 2016. URL: https://www.imperva.com/blog/malware-analysis-mirai-ddos-botnet/.

[81] Jiayun Zhang et al. *Federated Learning with Client-Exclusive Classes*. 2023. arXiv: 2301.00489 [cs.LG].

[82] Tao Zhang et al. "How to Mitigate DDoS Intelligently in SD-IoV: A Moving Target Defense Approach". In: *IEEE Transactions on Industrial Informatics* PP (Jan. 2022), pp. 1–10. DOI: 10.1109/TII.2022.3190556.

[83] Rui Zhu et al. "LRID: A new metric of multi-class imbalance degree based on likelihood-ratio test". In: *Pattern Recognition Letters* 116 (Sept. 2018). DOI: 10.1016/j.patrec.2018.09.012.

Acronyms

API Application Programming Interface.

C&C Command and Control.

CPS Cyber Physical System.

CPU Central Processing Unit.

CSG Communication Systems Research Group.

DDoS Distributed Denial of Service.

DDPG Deep Deterministic Policy Gradient.

DiD Defense-in-Depth.

DL Deep Learning.

DNN Deep Neural Network.

DoS Denial of Service.

DP Dynamic Programming.

DQN Deep Q-Network.

DRL Deep Reinforcement Learning.

ED Edge Device.

FL Federated Learning.

FN False Negative.

FP False Positive.

FRL Federated Reinforcement Learning.

GIL Global Interpreter Lock.

GPIO General-Purpose Input/Output.

IDS Intrusion Detection System.

IID Independent and Identically Distributed.

IoT Internet of Things.

IoV Internet of Vehicles.

IP Internet Protocol.

IT Information Technology.

LED Light Emitting Diode.

MAE Mean Absolute Error.

MID Multiclass Imbalance Degree.

MitM Man-in-the-middle.

ML Machine Learning.

MP Moving Parameter.

MSE Mean Squared Error.

MTD Moving Target Defense.

NN Neural Network.

PCA Principal Component Analysis.

QOS Quality of Service.

RF Radio Frequency.

RL Reinforcement Learning.

RMSE Root Mean Squared Error.

RPi Raspberry Pi.

SARSA State–action–reward–state–action.

SDN Software-Defined Networking.

SGD Stochastic Gradient Descent.

SQLI SQL Injection.

TD Temporal Difference.

TN True Negative.

TP True Positive.

UPnP Universal Plug and Play.

UZH University of Zurich.

WCS Weighted Cosine Similarity.

XSS Cross Site Scripting.

Glossary

Artificial Intelligence : is a discipline concerned with building computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.

Machine Learning : refers to the use and development of systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data.

Deep Learning : is a subfield of machine learning concerned with algorithms and models inspired by the function of the human brain called artificial neural networks. These are built up in layers and when the number of layers becomes large, this is called Deep Learning.

Malware : is software that is specifically designed to disrupt, damage, or gain unauthorized access to a computer system.

Ransomware : is a type of malware that blocks the access to information and demands the payment of ransom for potentially regaining access.

Defense-in-Depth : Defense in Depth (DiD) refers to an information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within.

Software Defined Networking : is an approach to networking that uses software-based controllers or application programming interfaces (APIs) to communicate with underlying hardware infrastructure and direct traffic on a network.

Differential Privacy : is a method for being able to publicly share information about a data set by describing patterns of groups (e.g median, mean) while withholding information about individual observations.

Independent and Identically Distributed (IID) : a set of random variables is denoted as IID, if each random variable has the same probability distribution as the others and all are mutually independent from each other.

List of Figures

List of Tables

Appendix A

Behavior Data Boxplots

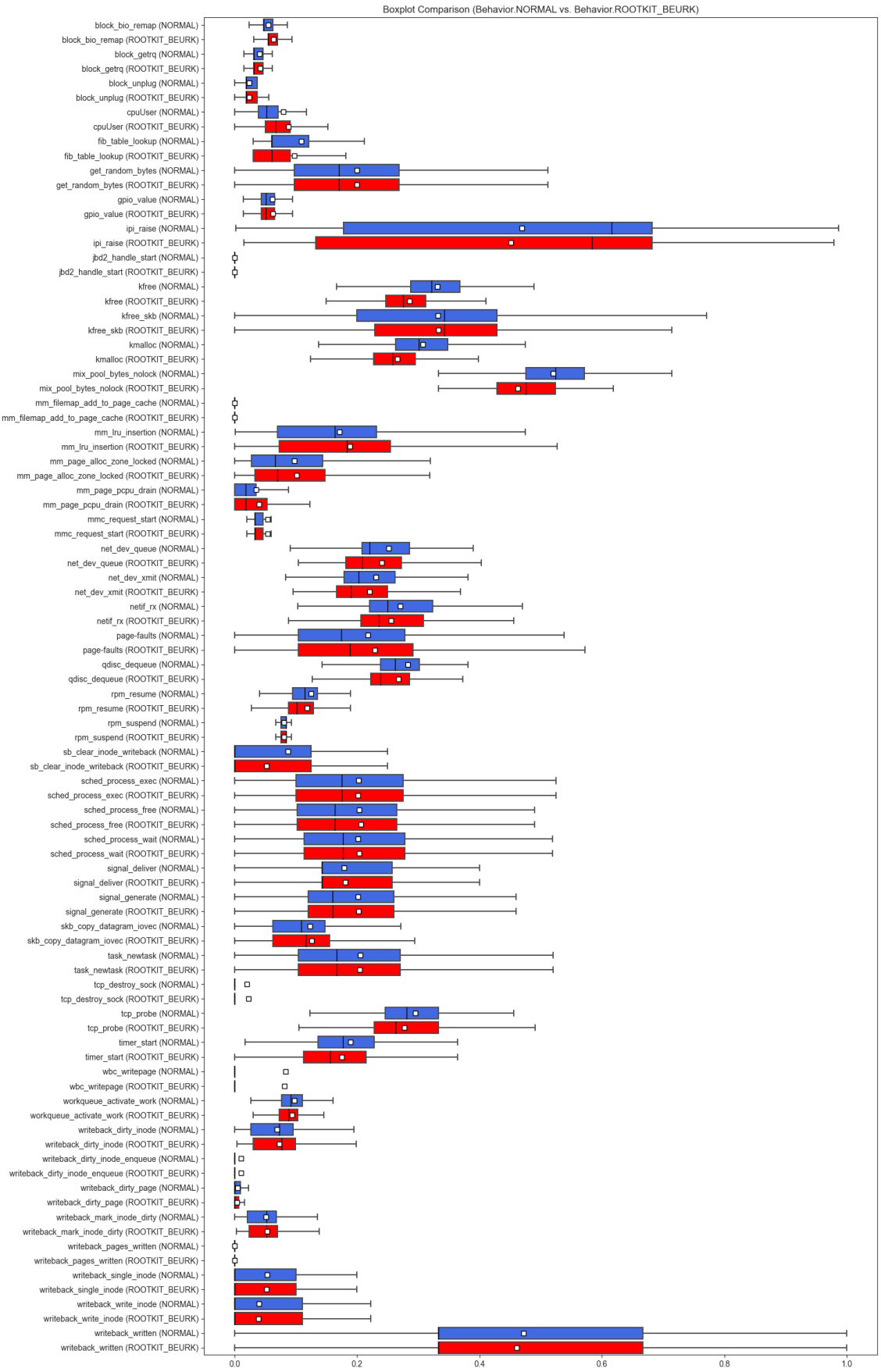


Figure A.1: Boxplot Comparison (Behavior.NORMAL vs. Behavior.BEURK)

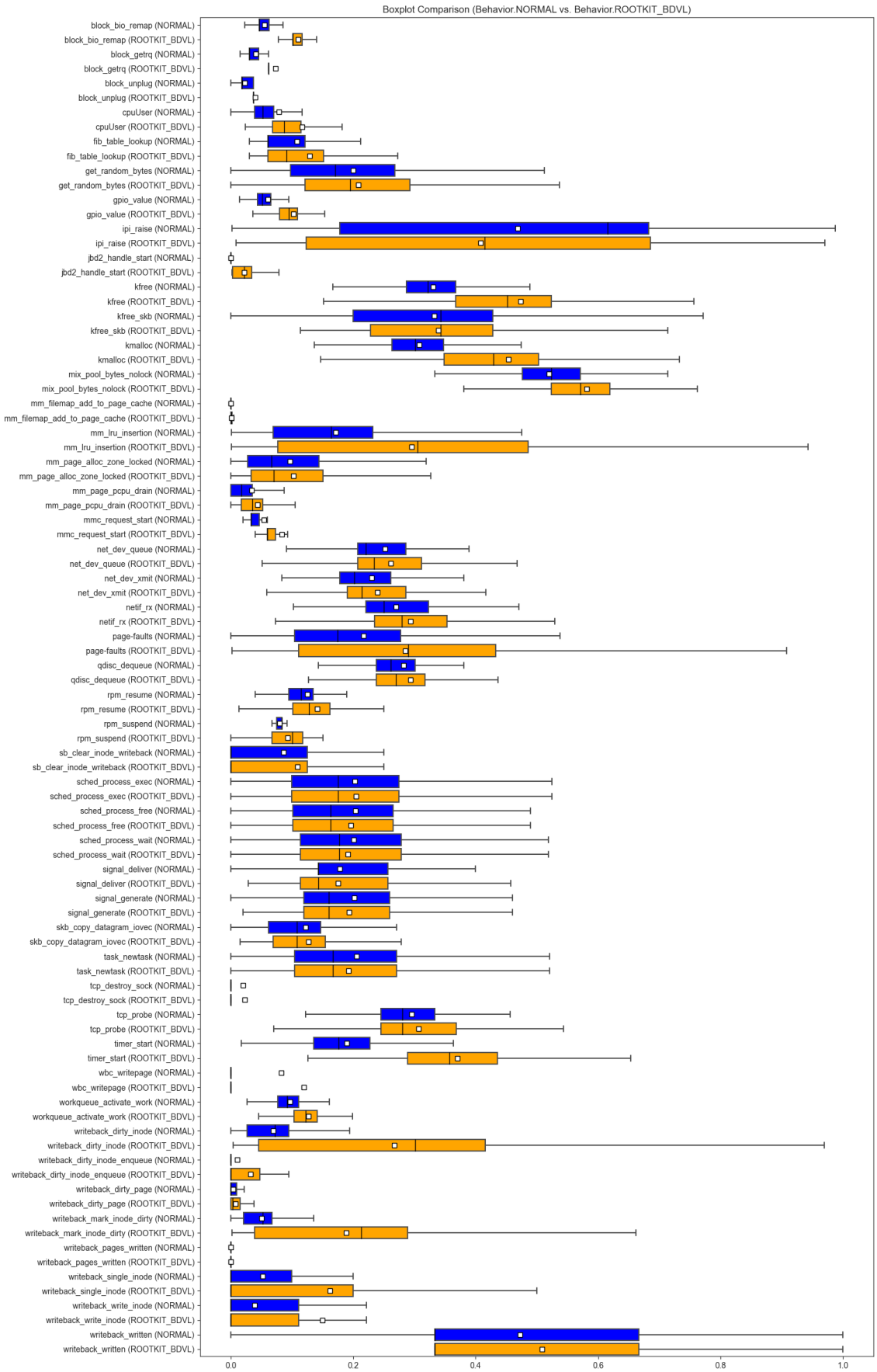


Figure A.2: Boxplot Comparison (Behavior.NORMAL vs. Behavior.ROOTKIT_BDVL)

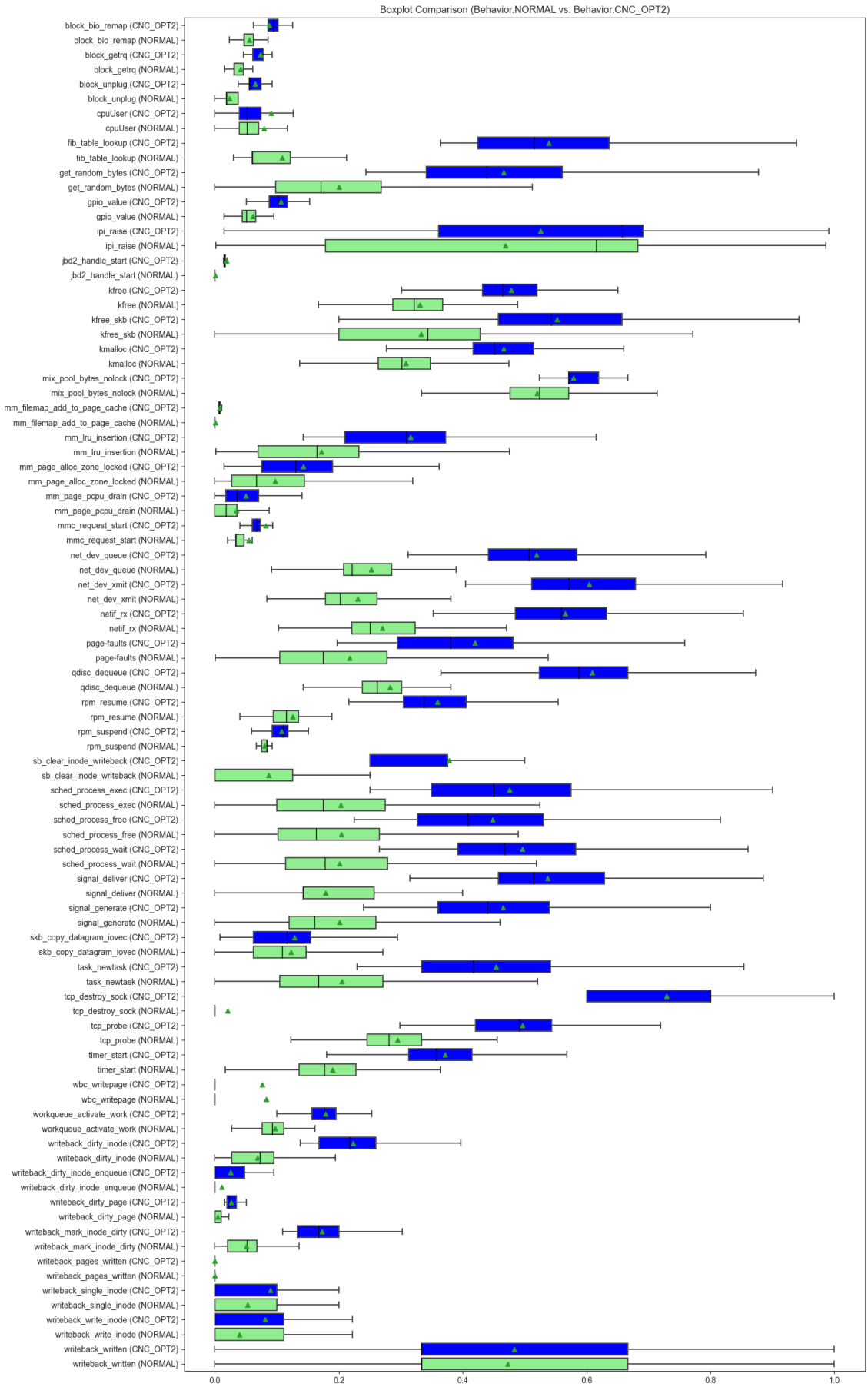


Figure A.3: Boxplot Comparison (Behavior.NORMAL vs. Behavior.CNC_OPT2)

Appendix B

Project Gantt Chart