University of Zurich[UZH]

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

Implementation and Detection of Spectrum Sensing Data Falsification Attacks Affecting Crowdsensing Platforms

Wassink Robin
Zürich, Schweiz
Student ID: 19-738-558

Supervisor: Dr. Alberto Huertas Celdran, Jan von der Assen
Date of Submission: July 24, 2022

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

ifi

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

Zusammenfassung

Die Nutzung von mobilen Daten hat drastisch zugenommen in den letzten Jahren und der Trend ist steigend. Teil dieses Trendes ist dem Internet-of-Things (IoT) zuzuschreiben, welches die digitale und physikalische Welten verbindet. IoT-Geräte sammeln und senden unzählige Mengen an Bits über Funkfrequenzen was zu Diskontinuität oder Überbelastung führt. IoT-Geräte werden aber auch vorteilhaft eingesetzt als Sensoren in Netzwerken die der Überwachung und Analyse der Funkfrequenzen gelten, mit dem Ziel, die Nutzung zu optimieren. Diese Geräte sind jedoch bekanntermassen ressourcenbeschränkt und folglich ein wachsendes Cybersecurity Risiko. In Sensornetzwerken sind sie sogenannten Spectrum Sensing Data Falsification (SSDF) Angriffen ausgesetzt, welche versuchen, Daten zu manipulieren. Jüngste Forschungsergebnisse haben maschinelles Lernen basierend auf 'behavioral fingerprinting' als vielversprechende Erkennungsmethode präsentiert.

Um der entsprechenden Forschung einen Beitrag zu leisten, wird in dieser Arbeit eine neue Implementation der SSDF Attacken präsentiert. Die Software, welche in Sensoren der crowdsourcing Platform ElectroSense verwendet wird, wurde modifiziert und die sieben SSDF Attacken wurden implementiert. Die Attacken wurden in verschiedenen Konfigurationen ausgeführt und das Verhalten des infizierten Gerätes überwacht anhand von Systemaufrufen. Eine Machine Learning (ML) Pipeline hat darauffolgend die Daten bereinigt, Merkmale extrahiert und mehrere unüberwachte maschinelle Lernalgorithmen trainiert mit normalem Verhalten. Die infizierten Daten wurden anschliessend von den Modellen klassifiziert, um die Anomalieerkennung zu bewerten in unterschiedlichen Situationen. Die Experimente haben erwiesen, das die neu implementierten Attacken, welche Variabeln verwenden, nicht zuverlässig entdeckt werden im Gegensatz zu den Attacken welche Dateien auf der Festplatte verwenden.

Abstract

The usage of mobile data has increased massively over the past few years and the trend is only rising. A part of this trend is due to the growth of the Internet-of-Things (IoT), which is merging the digital and physical worlds. IoT devices are collecting and transmitting countless of bits over the wireless spectrum and as a result, the radiofrequency (RF) spectrum is getting bursty and overcrowded. Yet IoT devices are also beneficial for the RF spectrum, as they are used as sensors in monitoring networks that analyze the spectrum usage to optimize the use of the wireless spectrum. However, these devices are well-known to be resource-constrained and therefore a growing cybersecurity concern. In a sensing network, they are vulnerable to Spectrum Sensing Data Falsification (SSDF) attacks trying to manipulate the data. Recent research has proposed behavioral fingerprinting and Machine/Deep Learning (ML/DL) to detect those attacks.

To improve the limitations of the recent literature, another implementation of the latest defined SSDF attacks is proposed in this thesis. The sensing software used in the crowdsensing monitoring platform ElectroSense has been modified to implement seven SSDF attacks. The attacks have been executed in several different configurations whilest the behavior of the infected device has been observed based on the system call trace. A Machine Learning (ML) framework thereafter has cleaned the gathered datasets, extracted features and trained multiple unsupervised ML algorithms with normal behavior data. The infected data has then been classified by the models to evaluate the anomaly detection performance in different settings. The experiments have demonstrated that the proposed implementation using variables is not reliably detectable compared to previous implementations using files stored in disk.

Acknowledgments

I would like to express my deepest gratitude to my supervisor Dr. Alberto Huertas Celdrán for his constant support and guidance during this bachelor thesis. The periodic and spontaneous meetings have been very helpful and have resulted into many great inputs.

Many thanks as well to Mr. Pedro Miguel Sánchez Sánchez for repeatedly meeting up and giving helpful insights.

Additionaly, I would like to thank Chao Feng for giving me insights into his master project which was the foundation of large parts of this thesis.

Many thanks should also go to my friends and family that have supported me throughout this journey and have helped proof-reading.

Finally, I'm very grateful to Prof. Dr. Burkhard Stiller and the Communication Systems Group of the University of Zürich for allowing me to carry out this challenging bachelor thesis at their research group.

Contents

Chapter 1

Introduction

Arguably, the Internet-of-Things (IoT) is becoming one of the most important technologies of our current century. Everyday items like the fridge, the car or the radiator are getting connected to the internet to enhance our daily routine. Sensors, microcontrollers, and other computing devices are creating, collecting, and sharing data to allow a seamless connection between the physical and digital world. Thanks to this evolving hyperconnected motto, huge amounts of data are flooding the radiofrequency (RF) band. On top of that, the mobile data usage overall is increasing tremendously. With all those trends, the need for a reliable method to analyze, optimize and efficiently allocate the RF spectrum is getting more and more urgent. Solutions to solve the problem have been proposed, but not with their fair share of challenges.

1.1 Motivation

One of the solutions proposed to optimize the usage of spectrum resources are cognitive radio networks (CRN). Devices of the network can sense the environment and adapt appropriately to use under-utilised bands or vacate overcrowded portions [1]. The cognitive cycle of the CRN usually consists of three steps, the first of them being the spectrum sensing, making it one of the most important components of a CRN [2]. ElectroSense is an open-source platform that is dedicated to such spectrum sensing. Relying on crowdsourced IoT sensors consisting of an antenna attached to a Raspberry PI, the platform collects, analyzes and presents spectrum data [3]. Low-cost sensing nodes are used as they keep the entry barrier for volunteers as low as possible, which also has its drawbacks. Resource constrained IoT devices generally have an increased vulnerability to various cyberattacks as a consequence of their decreased computational power, storage space and battery [4]. Furthermore, CRNs not only face the regular malicious attacks as traditional wireless networks, but are also threatened by attacks specifically to the cognitive setting [5]. These include the so called Spectrum Sensing Data Falsification attack (SSDF) which tries to disrupt the spectrum allocation by sending false sensing data. SSDF attacks occur in various network settings with the main goal being the modification of spectrum data with malicious intent. Thus, to protect the network from interference of hostile attackers,

it is necessary to implement a defense mechanism with the purpose of detecting SSDF attacks. An appropriate way to detect anomalies in IoT sensors are Machine Learning (ML) algorithms [6]. Combined with behavioral fingerprinting, it has been suggested to be one of the most promising approaches to detect cyberattacks [7]. However, the existing works have some key constraints. On one hand, the existing SSDF attacks have been implemented based on a specific approach (using files stored in disk) whilst other implementation techniques have not been considered. On the other hand, the existing machine learning and deep learning (ML/DL) based solutions using behavioral fingerprinting create their profile according to the usage of resources [8]. Other ways to create behavioral fingerprints such as monitoring and analyzing system calls are on the rise [9] but need further evaluation.

1.2 Description of Work

To improve the current studies, the main objective of this thesis is to implement SSDF attacks in a new fashion and evaluate their detection with behavioral fingerprinting and ML algorithms considering a different type of data source than resource usage. To reach that goal, the main contributions of this work include:

- Implementing the functionality and behavior of seven malicious attacks called repeat, mimic, confusion, noise, spoof, freeze and delay which are classified as Spectrum Sensing Data Falsification attacks. The implementation is based on another criterion than proposed in the previous literature, specifically using variables instead of files stored in disk.
- Designing and implementing a monitoring component to observe the normal and infected behavior of a sensory device connected to a crowdsourced spectrum monitoring platform. The results are different datasets in order to create a device fingerprint based on system calls.
- Setting up a ML framework that preprocesses, cleans and transforms the gathered datasets into numerical features to train unsupervised ML algorithms in order to detect anomalies.

The acquired results show the successful implementation of five out of seven SSDF attacks utilizing variables. System calls prove not to be a robust approach for behavioral fingerprinting considering the newly implemented attacks as the detection performance is generally poor. Furthermore, the affected bandwidth and corresponding impact on the software influences the detection performance significantly.

1.3 Thesis Outline

The remainder of this work is structured as follows. The second chapter introduces the necessary terms used in this work and provides the theoretical knowledge to understand

the work done. In Chapter 3, a review of different approaches to solving the problem of SSDF attacks is provided. After that, in Chapter 4 the underlying scenario and the consequently resulting system design are stated. Chapter 5 then explains the implementation of the first requirement, the novel implementations of the SSDF attacks. Following that, the implementation of the detection, which contains feature extraction and ML model training is described in Chapter 6. Chapter 7 reviews the gathered data and evaluates the performance of the trained algorithms. Finally, this thesis is concluded with a summary and a short outlook for future work.

Chapter 2

Background

This chapter describes the theoretical background used in this thesis. First, basic underlying technologies are presented, which lead to the introduction of the SSDF attack. This is followed up by the introduction into system monitoring including behavioral fingerprinting. Finally, different data preparation methods and ML/DL techniques to detect anomalies are presented.

2.1 Radiofrequency spectrum

In the past few years, the general public has get used to flawless and rapid wireless communication. It has become increasingly harder to ensure this high standard due to the accumulation of different developments. The demand for mobile data has grown tremendously, data rates are increasing and new technologies arise. This leads to the challenge of efficient usage of the backbone of modern wireless communication - the radiofrequency spectrum [3].

2.1.1 Electromagnetic waves

Modern communication highly relies on the electromagnetic spectrum, which contains the complete range of wavelengths of the electromagnetic radiation. Electromagnetic radiation travels in the form of waves and has electric and magnetic properties. These waves can either be naturally emitted, for example by the sun, which is commonly known as the visible light, or man-made as used in wireless communication, microwaves, radar or x-rays. The whole electromagnetic spectrum can be divided in so called frequency bands, depending on their wavelength, frequency or energy, which are all proportional to each other. The segments this work is interested in, are part of the radiofrequency spectrum, ranging from 3 kHz up to 300 GHz. This contains bands used for military communication, radio, TV, Wi-Fi and many more [10].

2.1.2 Spectrum monitoring

Next to legacy technologies using the radiofrequency spectrum, novel technologies like the Internet-of-Things are on the rise. Ordinary things in daily life are getting smarter and smarter, which leads to an enormous amount of data being sent over the Internet [11]. Combined with a huge variety of different protocols, this leads to a very diverse and fragmented use of the radiofrequency spectrum [3]. In the worst case, some frequency bands are overcrowded which causes lower transmission quality due to re-transmissions or decreased data rates [8].

The most promising approach to improve the spectrum utilization are CRNs [5]. The secondary user can access a vacant spectrum without interfering with the primary user, which is licensed to operate in that specific frequency band. With the help of this dynamic spectrum allocation, the resource scarcity can be reduced. This involves spectrum sensing, spectrum decision making and spectrum management [12]. On the first step is where crowdsensing platforms come in handy. One of the goals of those monitoring platforms is to provide knowledge about the actual radiofrequency utilization over time and space to allow useful spectrum allocation of under-occupied bands. The crowdsensing platform used in this work is called ElectroSense, "a collaborative spectrum data monitoring network [...] that monitors the spectrum at large scale with low-cost spectrum sensing nodes" [3].

2.1.3 Spectrum sensing data falsification attack

There are different ways to disturb the efficient spectrum management and as in any network, one of them is a cyberattack. There are a lot of different security threats on various layers but this work is concerned with malicious attacks at the root of cognitive radio networks, the spectrum sensing level. The integrity of spectrum sensing data is generally manipulated by active attacks that modify the information, while passive attacks do not interact with the network and solely intercept information [13]. Further classification of the attacks is reasonably broad but in general, the attacks relevant for this thesis are the primary user emulation attack (PUEAs) and SSDF attack. A malicious user that misdirects other users by sending modified spectrum data is called a SSDF attack. The PUEA is performed when a malicious user misleads other users by transmitting emulated signals from the primary user [14]. Making other devices believe that a specific frequency band is occupied by sending high sensing values, although that is not the case, can be considered a SSDF attack for instance. In this context, this work has studied the recent research of SSDF attacks. There are different categorizations, the one used in this work consists of three families: Transmission hiding, Transmission simulation and a hybrid variant containing both of the previous ones. The following table contains the seven proposed attacks and their corresponding description.

Table 2.1: Table of the seven SSDF attacks based on [8]

Family	Attack	Description
Hybrid	Repeat	Replicates the Power Spectrum Data (PSD) of Segment A (S_A) in a specific moment of time into the following segments over a longer period of time
	Mimic	Replicates the PSD values of (S_A) into Segment B (S_B) in every sensing cycle
	Confusion	Exchanges the values of S_A with S_B
Transmission simulation	Noise	Adds random noise to the PSD values of S_A
	Spoof	Mimics S_A into S_B and adds random noise to the PSD values
Transmission hiding	Freeze	The PSD values of S_A are replicated over a certain amount of time
	Delay	The PSD values of S_A are delayed with a sliding time window

2.2 Anomaly detection

2.2.1 Behavioral fingerprinting

One of the most promising approaches to detecting cyberattacks is behavioral fingerprinting, which is a special form of device fingerprinting [7]. The idea of device fingerprinting is creating a robust profile to recognize a particular device and its legitimacy. This fingerprint is based on specific information related to that distinct device that then functions as an immutable identifier. Behavioral fingerprinting specifically creates this ”digital biometric” based on the actions of the device such as resource usage or network traffic [15]. As there has been a previous study utilizing resource usage [8] and system calls have proven to be a suitable feature for anomaly detection [16], latter has been chosen in this work to improve the current state of research.

2.2.2 System call

A system call is the way a computer program interacts with the kernel of the operating system. An executing instance of a program is called a process. In general, processes are run in so called user space that has limited privileges to ensure stability and security. To execute tasks with higher privileges, the kernel space has to be accessed with a system call [17]. By that, processes can perform operations as opening and manipulating files, creating processes or much more by sending a task to the kernel.

2.2.3 System Call Feature extraction

System calls can easily be monitored on a computer. They consist of a command, timestamp and the corresponding parameters. As they can be seen as a series of instructions, they are comparable to a text sequence. Therefore, to transform the raw data into numerical features, which are needed to train a machine learning algorithm, several feature extraction methods from Natural Language Processing (NLP) can be applied [9].

The ones used in this paper are based on a bag-of-words, sequential encoding and a simple hashing encoding. The bag-of-words approach creates a list that keeps track of every occurrence of a system call. Thus, the original trace is converted into a bag of system calls where only the frequency but not the ordering information is preserved [18]. Every system call is then represented as a variable in the final feature vector, depending on the created dictionary that keeps track of the indices of the existing variables. In general, these bags can be filled in different ways: with the actual frequency, the so called Term Frequency-Inverse Document Frequency (TFIDF), which considers the significance of a specific term compared to the whole corpus [19], or a hashing encoding. The TFIDF can extract information from the whole dataset instead of just from one sample as with the term frequency approach.

System calls can be viewed individually but can also be considered in pairs or more. This is known as creating n-grams. An n-gram is a contiguous sequence of n items from a larger sequence [20]. That means, in this work an n-gram is a sequence of n consecutive system calls extracted from a monitoring cycle. The previously reviewed bag of words approach can by that be extended and assign values corresponding to their cumulative frequency or TFIDF to these newly created n-grams. The maximum n-gram used in this thesis is a 3-gram or trigram, which corresponds to a variable consisting of three consecutive system calls.

The other way to create a feature for the machine learning algorithms is based on sequential encoding, which represents the system call sequence with a numerical value. The numerical value can either be the dictionary index or the one-hot encoding [9]. The one-hot encoding transforms each system call into a list as long as the total amount of system calls of binary values that represents the actual system call with a 1 at the corresponding index. An example can be found in Figure 2.1 representing all feature extraction approaches used in this thesis.

System call trace	open, read, ioctl, poll, ioctl,
Dictionary	{0:open, 1:read, 2:ioctl, 3:poll}
Bag-of-words frequency	[1, 1, 2, 1]
Bag-of-words TFIDF	[0.2, 0.2, 0.4, 0.2]
Sequence dictionary index	[0,1,2,3,2]
Sequence dictionary onehot	[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1], [0,0,1,0]]
Dictionary 2-gram	{0:open read, 1:read ioctl, 2:ioctl poll, 3:poll ioctl}

Figure 2.1: Examples of the feature extraction techniques

2.2.4 ML algorithms

The previous section has explained how the captured data can be numerically represented through different features. Finding outliers inside that numerical data that diverge from the normal patterns is considered anomaly detection [21]. Widely used algorithms for outlier detection are Isolation Forest (IF), Local Outlier Factor (LOF), One-class Support Vector Machine (OC-SVM) and Robust Covariance [22, 23]. Each will be reviewed in the following subsections.

Isolation Forest

Random Forest is an attractive way when working with high-dimensional datasets [9] and has already been proposed as best machine learning algorithm for anomaly detection [24]. Hence, the IF is a promising algorithm for this thesis. The algorithm splits the data recursively based on a feature to build a random IF containing a lot of tree-like models. The basis to detect anomalies is the path length from the root to the leaf nodes [25]. It can suffer from long training time since it will generate many trees, is not known for dealing with high dimensionality data but is robust to outliers and can handle them well [24].

One Class Support Vector Machine

The Support Vector Machine (SVM) is a commonly used machine learning algorithm in various forms for different domains [20]. Essentially, the OC-SVM algorithm maps the

input data into a high dimensional feature space with the help of kernel functions and then attempts to find an appropriate hyperplane that separates the training data points from the origin [26].The algorithm considers outliers close to the origin and normal data far away. The classification of normal or anomalous can then be made based on the hyperplane [26]. OC-SVM is well suited for tasks with high data dimensionality, it can handle large data volumes and makes no distribution of the data [9].

Robust Covariance

Another commonly used anomaly detection algorithm is Robust Covariance. In contrast to OC-SVM, this algorithm assumes a Gaussian distribution of the normal data and finds the boundaries of inlier and outlier data as elliptical or ellipsoidal [27]. From there, the algorithm makes an estimate of the location and covariance of the normal data and tries to find the optimal elliptic boundary. The Robust Covariance algorithm has high robustness and effectiveness, but its data distribution assumptions and high computational consumption limits its usage scenarios if working with high dimensionality data [9].

Local Outlier Factor

Local Outlier Factor calculates the lof value, which is used to describe the ”degree of outlier” for each datapoint based on the relative density measure outlier factor relative to its surrounding points [28]. LOF calculates the local deviation of every data point and evaluates it as outlier or inlier based on the density between the data point and its neighbors. A lower density means that the point is more likely to be identified as an outlier [23]. Being independent of an underlying distribution assumption and other prerequisites makes the LOF an appropriate choicee in anomaly detection.

Chapter 3

Related Work

In this chapter, a review of different related works about SSDF attack detection is provided. First, different SSDF attack types are discussed. Based on that, the different detection techniques and the countermeasures are presented. To finalize this chapter, an overview of the different related literature coupled with the therefore arising research gap is provided.

3.1 SSDF detection

The concern of SSDF attacks has made an appearance with the rise of cognitive radio networks. Detecting SSDF attacks has since then been a topic of interest for over a decade now. Over this time span, people have amassed different approaches to define SSDF attacks and proposed various detection methods. In the following section an overview of the common approaches to tackle the SSDF attack problem is provided.

There are different definitions of the SSDF attack, but usually it is considered in a cognitive network setting, where secondary users are collaborating to sense the radio frequency spectrum and decide if a segment is vacant or occupied [29]. This can be done in a centralized way, where a fusion center reads in all information and makes a final decision or decentralized, where secondary users communicate with each other and decide on their own [30]. Over a long period of time, attackers and their behavior have only been sparsely defined and vaguely categorized into the three groups [31]:

- Malicious users that send manipulated data in order to confuse other nodes or the fusion center. The goal is to let other users make the wrong decision of using or avoiding a segment to disrupt efficient spectrum allocation or cause harmful interference with other devices.
- Greedy users that report all the time that a specific segment is used with the aim to keep the specific band for themselves.

- Unintentionally misbehaving users that send wrong observations because of a faulty behavior due to a virus or internal mistake [31].

Furthermore, a secondary user can report their incoming signal in two ways, either continuous (e.g. power estimation from an energy detector) or binary, where the signal is either absent or present [31]. There are also more ways to modify the attacks from the adversary's perspective. The question of when to attack can be changed: either with a certain probability or not. The extreme cases of always or never attacking lead to the so called always yes/no attacks. Probabilistic attacks are supposed to be stealthier but can also be easier identified by statistical analysis as the attacker is consistent over time slots [30].

Those underlying different definitions can lead to whole different proposed detection techniques with various results. In [32], the authors make use of the probabilistic always yes attack and propose a detection method based on the neighborhood distance approach. With the help of spatial information and the energy detection, outliers are detected. Based on simulated data, malicious users send false high energy values if the primary signal is absent, which is successfully detected for the case of a CRN spread over a wide area with significant difference in path loss components of the channels between the sensors.

Another work that makes use of spatial information is [33]. The research has been done with the sort of Denial of Service (DoS) attack, namely the PUEA, which is rather similar. Reviewed are selfish and malicious attacks, which are equal to the ones of the SSDF. The authors utilize both the location information of the primary user and the Received Signal Strength (RSS), which means a continuous reporting of the signal is used. With their transmitter verification scheme, called LocDef (localization based defense), multiple RSS of various sensory nodes are compared and evaluated to localize the transmitter. Test in different simulations have proven the localization scheme to be effective.

[34] makes use of continuous report as well, although the detection is based on another technique. The authors of this work propose a so called trust based technique. The trust based technique assigns an initial value to all users, which are updated for every spectrum sensing cycle. Malicious behavior is rewarded by a decreased weight to exclude the reported data at the fusion center [29]. [34] classifies three SSDF attacks: spectrum spoofing (sending data from another node), spectrum inversion (flipping sensed spectrum), and spectrum shifting (sending a shifted version of spectrum sensed). Malicious behavior is detected based on spectrum matching using Euclidian distance and ignores incoming results if the weighted trust does not pass a specific threshold [29].

Comparable to the trust based detection technique is the reputation based method, which basically boils down to the same principle. The work proposed in [35] consists of two steps: the reputation assignment step, where the reputation either increases if the sensing report matches with the final decision or the opposite, and the hypothesis step. The hypothesis step makes the decision based on Weighted Sequential Probability Ratio Test, which is an evolution of SPRT, first proposed in the previous century [36]. In the simulations the two probabilistic always-yes and always-free have been made use of. The hypothesis step has since then been improved even further [37].

Another popular addition to reputation based techniques are Bayesian-statistics as first used in [38], which proposed an onion-peeling defense scheme. It first computes the suspicion level of the nodes according to their sensing report, and then removes malicious nodes one by one based on their reputation until all remaining nodes are non-malicious. Simulations have shown a significant reduction in false alarm rates when applying this defense scheme.

A Bayesian-statistics approach has also been used in a recent paper by Fu et al. with an impressive detection method for probabilistic attacks in [39]. Multiple small sliding windows build a weighted trust scheme during the spectrum sensing process. Using a sigmoid log function, secondary users are evaluated to be discarded if a threshold is reached. This Bayesian-inference-based sliding window trust model has proven to be successful in simulations with various attackers and attacking probabilities.

Another approach without assigning trust or reputation values to users is by detecting abnormalities. In [40] the authors have proposed a multiple linear regression based on Maximum-Match-filter algorithm to combat the SSDF attacks in CRNs. The Hamming distance between the sensing data measures the similarity, which get plotted in a three dimensional space. A regression plane then decides what sensing values are reasonable. A threshold on the fitted plane selects the proper sensing reports for the collaborative spectrum sensing decision. With relatively small datasets, the algorithm can produce good results.

Clustering methods can also detect abnormalities as proposed in [41]. The k-medoids method used in this work has been called PAM2 and consists of two stages: checking if an attacker might exist and identifying attackers if any exist. In general, the intuition would be, that if two clusters are created, one contains honest users and one the malicious users. If the first stage therefore forms multiple clusters, a malicious user might be manipulating the data. From the simulations, it turns out that the approach gives good performance in terms of detection rate and false detection rate, without predefining a threshold or knowing the attackers strategy.

A completely different approach is proposed by [42], which makes use of a fingerprinting based method. PUEAs are detected through recognizing untampered data based on phase noise which is random but unique for each transmitter. Legitimate primary users can therefore be distinguished by emulated ones through the unique feature coupled with the Local Oscillator. As this sort of detection deviates from the previously mentioned approaches, the attack model is not really comparable, although it shows that transmitter identification is a feasible detection method.

The last work proposing an improved fingerprinting detection technique is [8]. Instead of the previous non-behavioral fingerprinting, the authors make use of behavioral fingerprinting. Internal events are monitored inside the spectrum sensing device and evaluated with machine learning and deep learning (ML/DL) algorithms. Furthermore, the authors propose the behavior of seven new SSDF attacks. Experiments with novel implementations have shown almost perfect detection for five of the seven attacks.

3.2 Limitations and comparison

The previous research in terms of SSDF attack detection has amassed various different approaches to solving the problem. Conspicuous is the fact, that most of the authors implement their proposal in simulations. Only fingerprinting based approaches have made use of experiments. Furthermore noticeable is that some of the approaches are more popular than others, for instance the reputation or trust based approaches seem to have a predominance, although they are far from perfect. To be fair, all detection techniques have their flaws, to name a few:

- Location based approach: The number one constraint of the location based approach depends on other nodes to validate a signal, which is problematic as their integrity can not be assured either. It relies on a high number of trusted collaborators [29] which is hard to achieve outside of a simulation. There are furthermore environmental factors that might influence the RSS and influence the classification. Location based approaches that communicate their position are also problematic because of the so called location privacy leakage, as an attacker can overhear the transmission and duplicate it itself [14].

- Statistical data based approach: The primary limitation of statistical data based approaches is the confidence level. A lower confidence level increases the chance of detecting an outlier and ensures the robustness of the model. However, this also increases the possibility of misdetection. A high sensibility therefore can only be guaranteed with an increased amount of misidentified data, which leads to an inconvenient trade-off problem [14].

- Reputation based approach: The main restriction of reputation based mechanisms is the dependency on historical data and other nodes. It takes time to be able to get a valid reputation level to be sure a node is not infected. It is possible, the sensing time is not long enough for finishing the process of assigning a valuable trust factor. The second major practical research issue is relying on valid other sensory nodes, with which the data from the original device can be compared with [14]. The honesty of those other sensory nodes can not be guaranteed in every situation. The detection quality also decreases for multiple attackers and might even break down once the attackers prevail.

- Behavioral fingerprinting: Behavioral fingerprinting does not rely on other devices but on a reliable fingerprint that guarantees the truthfulness of the data. This either needs a robust unique identifier, otherwise the attacker might be able to reproduce the fingerprint and deceive the defense mechanism. Furthermore, if the anomaly detection of the fingerprint is based on ML/DL algorithms, the complexity is significantly higher compared to other approaches. It also relies on reliable training data that can not always be guaranteed.

The main limitations therefore are the dependency on given securities, for instance a wide area coverage, other trusted devices or a robust fingerprint. The sensibility to detect every attack can not always be guaranteed either. As multiple trusted collaborators can not

be guaranteed in a real world application, the statistical and fingerprinting techniques seem to be most applicable. The issue of the trade-off problem leaves therefore the behavioral fingerprinting, which combined with ML/DL has been proposed to be a promising detection method [7]. Once a reliable training data set of a robust fingerprint can be guaranteed, small anomalies can be detected without the need of other devices. In recent research, the authors have proposed CyberSpec [8], but the research is for a start rather biased. As Celdrán et al. have stated, further investigations with behavioral fingerprinting can lead to an improvement of the detection accuracy, which has been the main reason for this thesis.

Table 3.1 provides an overview of all the previously mentioned works categorizing them with their corresponding detection technique. Their main characteristics and the way it has been implemented is documented as well. The attack probability (AP), either probabilistic (P) or non-probabilistic (NP) and way of sensory reporting (SR), which can be either continuous (C) or binary(B) is in the last two columns.

Table 3.1: Comparison of different SSDF detection approaches

Work	**Detection technique**	**Characteristics**	**Implementation**	**AP**	**SR**
[32]	Neighborhoodlocalization	• Compare received signals to detect outliers • No previous knowledge required	Simulation	P	C
[33]	Neighborhoodlocalization	• Analazying PUEA • Compare received signals to localize transmitter • Seperate sensor network for attack detection	Simulation	NP	C
[34]	Trust based	• Using three attacks: spectrum spoofing, spectrum inversion and spectrum shifting • Assigning trust based on behavior • Behavior analyzed with spectrum matching	Simulation	NP	C
[35]	Reputation based	• Assigning reputation based on behavior • Behavior analyzed with WSPRT	Simulation	P	B
[38]	Reputation based	• Assigning suspicion level based on onion-peeling defense scheme	Simulation	P	B
[39]	Trust based	• Weighted trust scheme based on multiple small sliding windows • Discarding data if a threshold is erached by using sigmoid function	Simulation	P	B
[40]	Abnormality detection	• Similarity measurement using Hamming distance between data • Multiple linear regression to evaluate outliers	Simulation	NP	B
[41]	Abnormality detection	• Outlier detection with clustering technique k-medoids • Two steps: checking if an attacker might exist and identifying attackers if any exist	Simulation	P	B
[42]	Fingerprinting	• Analyzing PUEA • Recognizing data based on phase noise	Experiment	-	-
[8]	Behavioral fingerprinting	• Using seven novel attacks • Behavioral fingerprinting with ML/DL algorithms	Experiment	NP	C

Chapter 4

Scenario & System Design

This chapter describes the underlying scenario and presents the system design that will shape the rest of the thesis. First, the environment of the sensor is explained. In the second part, the architecture with an introduction to the individual components is presented.

4.1 Scenario

Before describing the system design and its particular parts, the fundamental setup is introduced. ElectroSense has been the chosen RF spectrum sensing platform making it the central software to work with. In the next section, an overview of the ElectroSense environment is given.

4.1.1 ElectroSense Setup

The main goal of ElectroSense is to provide a platform for spectrum analysis. To serve the spectrum sensing data as intended, ElectroSense has setup a flexible and scalable architecture. This archictecture consists of three major parts: sensors, a centralized controller infrastructure and the backend. Although the other components are impressive in their provided value, this work focuses only on the sensors [3].

In order to setup a large-scale sensing network which guarantees the best possible coverage, there need to be as many sensors as possible. To achieve this goal, ElectroSense has chosen the crowdsourcing approach. With that in mind, the deployment of a sensor needs to be user friendly and cheap, aiming to keep the entry barrier as low as possible. This leads to sensing nodes consisting mainly of two components: an antenna and a single circuit board computer. The device used in this work is a Raspberry Pi 3 Model B with an armv7l Linux kernel, 1GB of RAM and 1.2GHz CPU Frequency. The device is connected to the Internet via Ethernet and sends the sensory data to the backend, where it is represented as the sensor UZH_CyberSpec_2. The Raspberry Pi is running headless and therefore has to be modified through a ssh connection. This has been massively simplified With the help

of Visual studio code remote connection [43] and port sharing to enable easy development from anywhere. ElectroSense furthermore provides a web front-end [44] which makes visualization of the spectrum possible [3]. Figure 4.1 visualizes the explained setup.

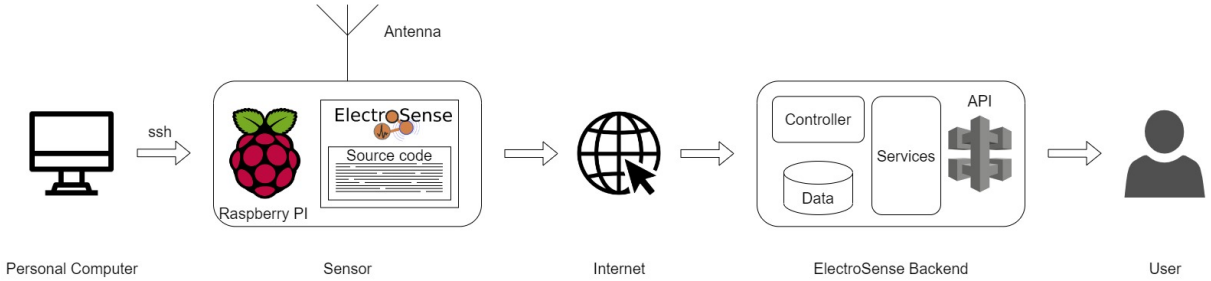


Figure 4.1: Setup of this thesis inspired by [3]

4.1.2 ElectroSense Source Code

The software running on the sensors is maintained by the internal developers of ElectroSense and publicly available [45]. The fact that the software is open source makes the sensing program easy to analyze and modify. The software is based on multiple source and header files written in the programming language C++, which get compiled with the help of the CMake compiling software. The executable which gets created this way works rather straight forward in a cyclic way through the RF spectrum and is visualized in the flow diagram in Figure 4.2.

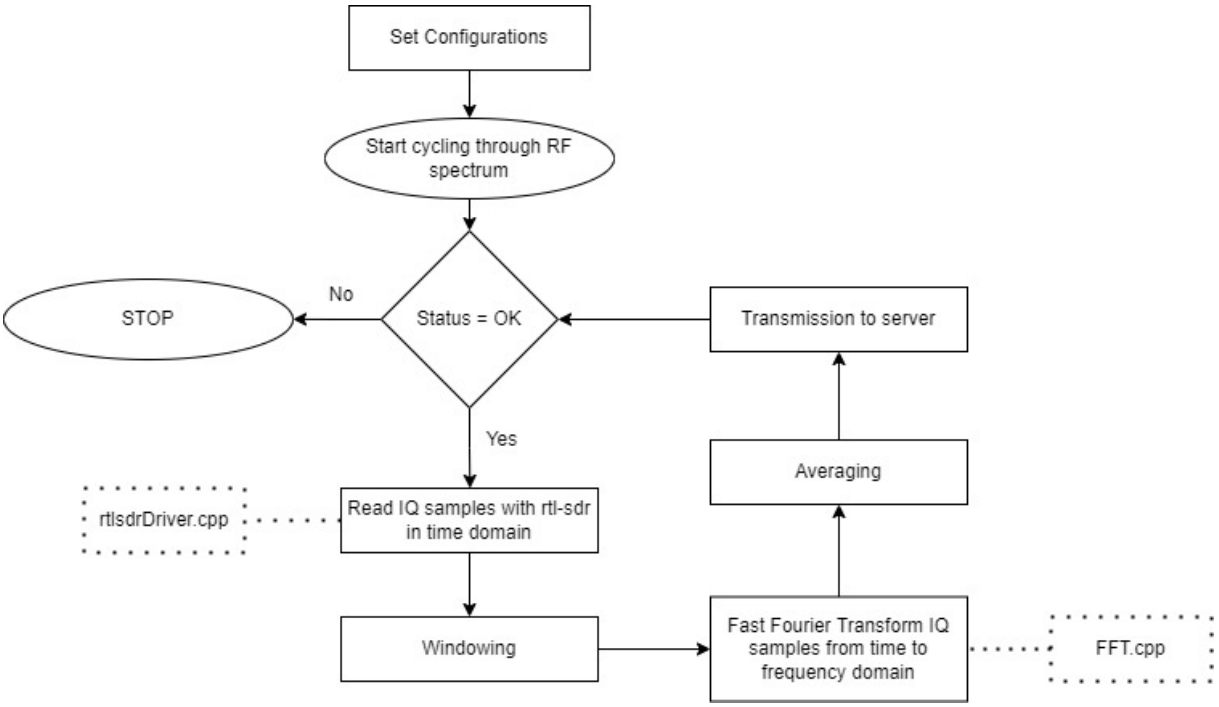


Figure 4.2: Flow Diagram of the ElectroSense program es_sensor

The process starts with setting the configurations and starting its components. Once the setup is done, it starts cycling through small segments of the RF spectrum. For every segment, the IQ samples are sensed with the help of the RTL-SDR dongle, where SDR stands for software defined radio. The IQ samples are quadrature signals detected by the antenna, currently in time domain, that need to be transformed. This is done after the windowing with the help of the Fast Fourier Transformation. The resulting samples in the frequency domain are averaged and transmitted to the server. Then the whole process starts again with another segment as long as the status of the sensory device is OK.

As the code has been modified before, the foundation of the source code used in this work is the already altered code used in previous research [8, 9]. There, the previously mentioned seven SSDF attacks have been implemented into two components that have been marked in Figure 4.2:

- rtlsdrDriver.cpp: The module responsible for retrieving the sensory data in the time domain.
- FFT.cpp: The module that performs the Fast Fourier Transformation, which converts the data from the time domain into the frequency domain.

This time, the attacks were not performed with the help of external files but with variables inside the code. The main code containing configurations has also been modified to simplify future usage.

4.2 System design

With the final goal being a new implementation of SSDF attacks including a model to evaluate the systems integrity, this work consists of two main parts. First the actual creation of the SSDF attacks inside the sensor and second the construction of a detection framework. This leads to the system design as shown in 4.3. The two main layers will be evaluated in detail in the following subsections.

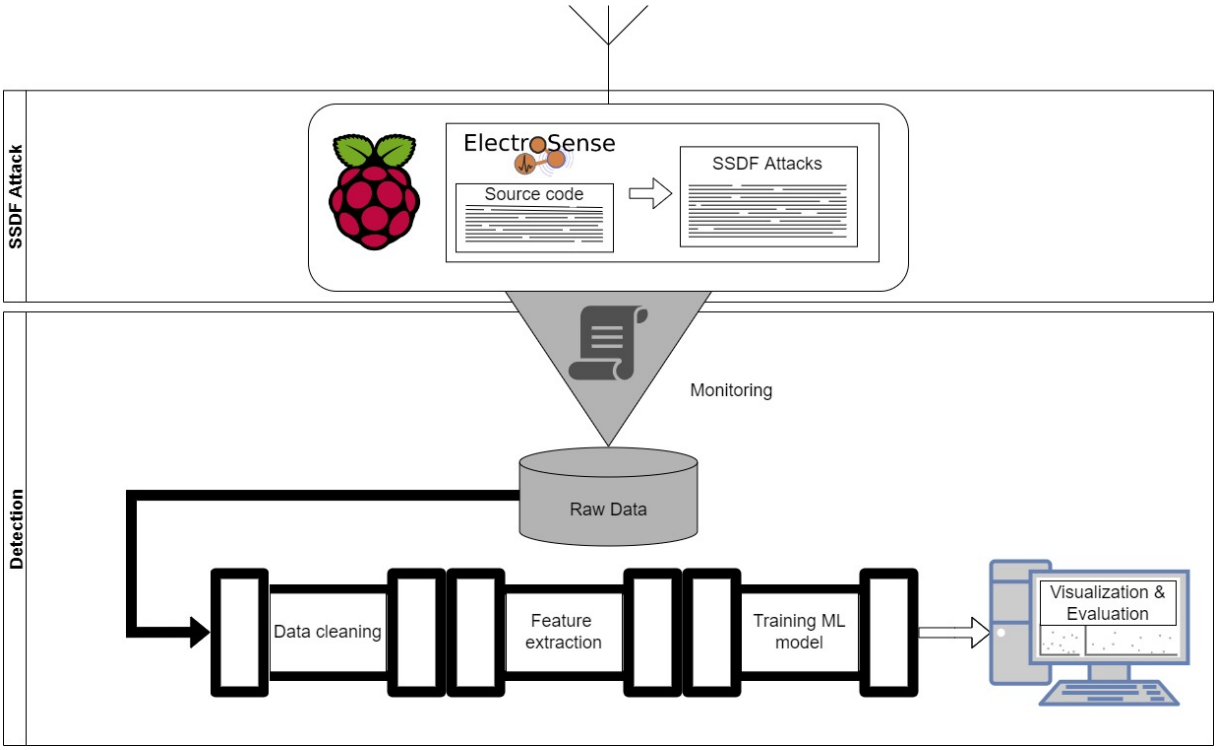


Figure 4.3: System design

4.2.1 SSDF Attacks

The first layer considers the aspects of the attack implementation. It is initiated with a reliable setup of an ElectroSense sensor which has been explained in the previous section. As a follow up, the source code needs to be altered to perform the correct behavior for each of the seven SSDF attacks. The attacks can be executed in different ways, only one of them being implemented currently, namely by writing sensing data to a file and accessing that later on once again [8]. This creates some overhead that can be detected through either monitoring the system calls or the resource usage. The robustness of this system can be tested through implementing the SSDF attacks in another way and analyze the performance again.

Another way to perform the attacks is by saving the temporary sensing data to a variable inside the executable. Accessing and manipulating a variable inside the program does not generate a system call as manipulating a file would, which implies that the new

implementation does not perform deviating instructions detectable by monitoring the system call trace. However, it might have a completely different impact on the device which still creates detectable anomalies like increased or decreased frequencies of specific instructions.

4.2.2 Detection

The second part of the proposed system consists of the dataset creation, data preprocessing & cleaning, feature extraction and finally training and evaluating the attack detection model. The following sections introduce the monitoring and ML framework responsible for those tasks. The procedure, features and algorithms are inspired by Chao Feng's work [9]. Furthermore, the scripts used in the detection part of this work are heavily based on the github repository [46] to create a pricavy-preserving SSDF cyberattack intelligent detection system, but adjusted where necessary.

Monitoring

Different monitoring techniques have been proposed to monitor the spectrum sensor. This work makes use of the system call tracking to create a valuable fingerprint. Therefore, the script needs to reliably supervise all instructions that are sent by the sensing process and all its sub-processes to the kernel. This is done with the Linux system monitoring tool `perf trace` [47]. All system calls with its associated parameters like the arguments and return code are written to a .txt file. Only the instruction is of interest in a future stage, which is the reason why a preprocessing script will be cleaning the data in a first step. Only the actual system call and its corresponding timestamps will remain in the subsequent .csv file, which is significantly less heavy.

The device has been monitored for two and a half hours for every behavior including the normal one. This created 300 samples for 30 seconds of monitoring, inspired by the configuration of [9], where 360x60sec was used. As the attacking bandwidth might have an influence of the device behavior, different attacking ranges have been observed: 20kHz, 200kHz, 20MHz, 200MHz and 800MHz.The first two bandwidths trigger the attack in the FFT.cpp section, the latter three in the rltsdrDriver.cpp module, which concludes all the raw data gathered. This resulted in 2400 samples of 30 seconds of monitoring for each bandwidth which meant an approximate total monitoring time of six days including the preprocessing time.

ML framework

The second part of the detection layer considers all work done outside the sensor device. The raw data has been transferred to a personal computer to create a ML pipeline. The detection framework has been implemented with the programming language python that supports numerous scientific libraries. The most important ones are scikit-learn for

preprocessing and machine learning, pickle for object serialization, numpy & pandas for dataframe handling and matplotlib & seaborn for visualization.

The first step of the framework involves further data cleaning to remove any erroneous or useless data. To prepare the data for training various algorithms, the textual data needs to be transformed into numerical data. This is done with feature extraction to convert the sequence of instructions into feature vectors. The frequency approaches have been combined with n-grams up to $n = 3$. Result of the feature extraction step are therefore nine datasets, two of them are sequence related, the other seven are different configurations of bag-of-words encoded features which are listed in Table 4.1.

Table 4.1: Feature extraction approaches inspired by [9]

Feature Extraction Approach	**Feature Encoding Type**
Bag-of-words	Frequency 1-Gram
	Frequency 2-Gram
	Frequency 3-Gram
	TFIDF 1-Gram
	TFIDF 2-Gram
	TFIDF 3-Gram
	Hashing
Sequence	One-hot encoding
	Dictionary index encoding

Some ML algorithms might yield a better performance with scaled data due to their sensitivity for the range of values. Therefore, the frequency and TFIDF features also have been standardized to compare the performance. Standardization is another scaling method where the values are centered around mean with a unit standard deviation.

Finally, the models needed to be trained and the performance evaluated. The datasets gathered for the normal behavior serve as training data for the different algorithms. Then, the data under infected circumstances had to be classified in order to evaluate the anomaly detection performance. The final used ML algorithms are Robust covariance, One-Class Support Vector Machine, Isolation Forest, Stochastic Gradient Descent One-Class Support Vector Machine and the Local Outlier Factor. The Robust Covariance had to be omitted with sequence features due to unreasonably long training time (8 hours+) as the dimensions of those features are significantly larger.

Furthermore, in this work, the performance of the ML models has been evaluated based on two different scores, the True Positive Rate (TPR) and the True Negative Rate (TNR). A True Positive (TP) stands for a correctly classified anomaly while a True Negative (TN) represents a correctly detected normal behavior. The opposite stands for False Positive (FP) and False Negative (FN). Thus, the TPR describes the ratio of actual positives (anomalies) correctly predicted. It is calculated by $TPR = \frac{TP}{TP+FN}$ Finally, the ratio of actual negatives correctly predicted (normal behavior) is the TNR, calculated by $TNR = \frac{TN}{TN+FP}$.

Chapter 5

SSDF Attack Implementation

The previous chapter has explained what underlying scenario this thesis is working with and what approach therefore resulted. Under those circumstances, the new SSDF attacks have been implemented. Previous implementation made use of files to keep data stored over time. The detection of those attacks lead to a promising performance but might not show the full picture as other versions interfere. The following implementations intend to improve this field of research. The exact source code can be found on Github [48].

5.1 SSDF Attacks

Latest research has proposed seven different SSDF attacks, which have to be implemented into the ElectroSense source code. They need to be implemented into two different classes which are responsible for retrieving the sensing values and applying the Fast Fourier Transformation. Depending on the affected bandwidth of the attack, the data manipulation occures in either one of the files. For 2MHz and below, the attacks take place in the FFT.cpp and above takes place in the rtlsdrDriver.cpp. Both of the implementations are comparable, which leads to the following subsections only explaining the realization in the rtlsdrDriver file. The attacks all build-upon the normal behavior. In a normal setting, the code cycles trough the whole spectrum and senses the IQ samples for a small segment around a given frequency center. Once the attacked segment is reached, the sensed data is either saved and or overwritten according to the attack behavior. All attacks are therefore an addition to the normal behavior. Table 5.1 lists the general variables and functions used in the pseudo codes.

Table 5.1: Variables and Functions used in the SSDF attack implementation pseudo code

Variable/Function	**Description**
Att_freq	The frequency where the attack starts
Att_freq2	The second attacked frequency if needed (only in mimic, confusion and spoof)
$cent\_freq$	The center frequency of the currently sensed segment
Att_bw	The bandwith the attack is affecting
Att_impact	The attack impact in the FFT.cpp segment, depending on Att_bw
$sdr.readSamples()$	Sample IQ signals with SDR
$fft\_execute()$	Perform the Fast Fourier Transform

Normal

To show where the attacks are implemented in the cycle, the following pseudo code shows a very simplified of the regular sensing progress in general with a focus on the two important modules:

Algorithm 1 Normal behavior

1: $setConfigurations()$
2: $InitQueue(Q)$
3: **while** $Running = True$ **do**
4: $iq\_vector = sdr.readSamples()$ ▷ rtsldrDriver.cpp
5: **if** behavior is malicious & $Att\_bw > 2MHz$ **then**
6: manipulate iq_vector ▷ perform attack here
7: **end if**
8: $segment = newSpectrumSegment(iq\_vector)$
9: $enqueue(Q, segment)$
10: ...
11: $segment = dequeue(Q)$ ▷ FFT.cpp
12: $signal\_freq = fft\_execute(segment.iq\_vector)$
13: **if** behavior is malicious & $Att\_bw \leq 2MHz$ **then**
14: manipulate $signal\_freq[Att\_impact]$ ▷ perform attack here
15: **end if**
16: assign $signal\_freq$ to $segment$
17: **end while**

First, in the initialisation of the program, the configurations are set. Following this, the main function initializes a queue that connects the different components like the rtlsdr-driver, windowing, etc. and allows the data to flow between them. Once the components are setup, the iterative sensing starts in the rtlsdr component by reading the samples and filling the iq_vector. SSDF attacks above 2MHz manipulate this variable. A SpectrumSegment class gets created with the variable iq_vector as a parameter. This goes through the other components until it gets dequeued in the FFT module. Here, the transformation is applied and the signal in frequency domain extracted. If there was a malicious attack in this bandwidth, it would manipulate the obtained values. The extracted signal is assigned back to the segment and the data flow can go on to different steps like the transmission to the backend. Now that the positions of the SSDF attacks are defined, each individual attack in the rtlsdr component is briefly introduced and visualized with pseudo code.

Repeat

The repeat attack essentially senses the first part of a larger segment and replicates this data across the specified range. Once the source segment is created, the data will be copied over the whole range until the attack stops. This leads to the following pseudo code:

Algorithm 2 SSDF Attack: Repeat

```
1: Repeat_source_segment ← empty vector
2: while Running = True do
3:     iq_vector = sdr.readSamples()
4:     if Att_freq < cent_freq < Att_freq + Att_bw then
5:         if Repeat_source_segment = empty then  ▷ Only the case for first segment & first iteration
6:             Repeat_source_segment = iq_vector
7:         else
8:             iq_vector = Repeat_source_segment
9:         end if
10:    end if
11: end while
```

Mimic

The mimic attack is basically a more sophisticated repeat attack, in the sense that a specific second segment is stated and the PSD values are copied every cycle instead of just being replaced by the PSD values at $t = 0$. This leads to the following pseudo code:

Algorithm 3 SSDF Attack: Mimic

```
1: Mimic_source_segment ← empty vector
2: SegmentCount ← 0
3: while Running = True do
4:     iq_vector = sdr.readSamples()
5:     if Att_freq < cent_freq < Att_freq + Att_bw then
6:         Mimic_source_segment push back iq_vector
7:     end if
8:     if Att_freq2 < cent_freq < Att_freq2 + Att_bw then
9:         iq_vector = Mimic_source_segment[SegmentCount]
10:        SegmentCount++
11:    end if
12:    if cent_freq > Att_freq2 + Att_bw then
13:        clear Mimic_source_segment
14:        Reset SegmentCount ← 0
15:    end if
16: end while
```

Confusion

Disorder and exchange are other designations this SSDF variation can be named as, which might be more descriptive to this behavior. The first segment is exchanged with the second attacked segment. Algorithm 4 visualizes this through pseudo code.

Algorithm 4 SSDF Attack: Confusion

```
1: Confusion_source_segment1 ← empty vector
2: Confusion_source_segment2 ← empty vector
3: SegmentCount ← 0
4: while Running = True do
5:     iq_vector = sdr.readSamples()
6:     if Att_freq < cent_freq < Att_freq + Att_bw then
7:         Confusion_source_segment1 push back iq_vector
8:         if Confusion_source_segment2 is not empty then ▷ Do not exchange on the
   first iteration as the second segment has not been sensed yet
9:             iq_vector = Confusion_source_segment2[SegmentCount]
10:            SegmentCount++
11:        end if
12:    end if
13:    if Att_freq + Att_bw < cent_freq < Att_freq2 then
14:        clear Confusion_source_segment2
15:        Reset SegmentCount ← 0
16:    end if
17:    if Att_freq < cent_freq < Att_freq + Att_bw then
18:        Confusion_source_segment2 push back iq_vector
19:        iq_vector = Confusion_source_segment1[SegmentCount]
20:        SegmentCount++
21:    end if
22:    if cent_freq > Att_freq2 + Att_bw then
23:        clear Confusion_source_segment1
24:        Reset SegmentCount ← 0
25:    end if
26: end while
```

Noise

The noise attack adds random noise to the attacked segment. The implementation of the noise attack has a flaw as the generation of the number relies on a random_device object. This object makes use of a system-specific source of randomness, the filesystem path "/dev/urandom" in this case.

Algorithm 5 SSDF Attack: Noise

```
1: while Running = True do
2:     iq_vector = sdr.readSamples()
3:     if Att_freq < cent_freq < Att_freq + Att_bw then
4:         iq_vector = iq_vector + randomValue
5:     end if
6: end while
```

Spoof

Spoofing is like the mimic attack but adding noise to the copied values. Since the noise once again relies on the "/dev/urandom" filesystem path, the implementation does not fulfil the requirement of independance of files.

Algorithm 6 SSDF Attack: Spoof

```
1: Spoof_source_segment ← empty vector
2: SegmentCount ← 0
3: while Running = True do
4:     iq_vector = sdr.readSamples()
5:     if Att_freq < cent_freq < Att_freq + Att_bw then
6:         Spoof_source_segment push back iq_vector
7:     end if
8:     if Att_freq2 < cent_freq < Att_freq2 + Att_bw then
9:         iq_vector = Spoof_source_segment[SegmentCount] + randomValue
10:        SegmentCount++
11:    end if
12:    if cent_freq > Att_freq2 + Att_bw then
13:        clear Spoof_source_segment
14:        Reset SegmentCount ← 0
15:    end if
16: end while
```

Freeze

The behavior of the freeze attack is defined as repeating the same values of a segment over time. This can be achieved with the following code represented in pseudo code:

Algorithm 7 SSDF Attack: Freeze

```
1: Freeze_source_segment ← empty vector
2: Freeze ← False
3: SegmentCount ← 0
4: while Running = True do
5:     iq_vector = sdr.readSamples()
6:     if Att_freq < cent_freq < Att_freq + Att_bw then
7:         if Freeze == False then
8:             Freeze_source_segment push back iq_vector
9:         else
10:            iq_vector = Freeze_source_segment[SegmentCount]
11:            SegmentCount++
12:        end if
13:    end if
14: end while
```

Delay

The delay is the most sophisticated attack from the implementation perspective. The signal needs to be stored and being repeated after the delay has been reached. The delay has been implemented as a count of segment cycles. It relies on a 3d array consisting of nested vectors containing the iq_vectors for multiple segments over time. Basically there exist three modes: init, filling and full. Init is the first iteration, thereafter comes filling and once the *Delay_source_segment* is full, the mode full is activated. The different modes are explained in Algorithm 8 and kept rather simple to demonstrate the idea. The full code can be found in the github repository.

Algorithm 8 SSDF Attack: Delay

```
1: Delay_source_segment ← empty 3d array
2: mode ← init, filling or full
3: delay ← x
4: SegmentCount ← 0
5: CurrentIteration ← 0
6: while Running = True do
7:     iq_vector = sdr.readSamples()
8:     if Att_freq < cent_freq < Att_freq + Att_bw then
9:         if mode == init then
10:            Delay_source_segment push back vector of size delay with iq_vector at the
   first position
11:        end if
12:        if mode == filling then
13:            Delay_source_segment[SegmentCount][CurrentIteration]      push back
   iq_vector
14:            SegmentCount++
15:        end if
16:        if mode == full then
17:            Exchange iq_vector with Delay_source_segment[SegmentCount][CurrentIteration]
18:            SegmentCount++
19:        end if
20:    end if
21:    if cent_freq > Att_freq + Att_bw then
22:        Reset SegmentCount ← 0
23:        when CurrentIteration == delay then Reset CurrentIteration else
   CurrentIteration++
24:    end if
25: end while
```

5.2 ElectroSense source code modifications

Instead of making an individual executable with all the different attacks replacing the normal behavior, the whole program has been modified to read all needed arguments and

run the desired behavior. This modification reduces the used space on the device as there does not need to be an executable for every configuration and it furthermore increases the convenience for future usage as the behavior or affected bandwidth can be modified by just changing an argument to the program. This has been exploited in the monitoring script to create an independent and automatic data collector.

The modifications have been made in different configuration segments of the code. The function parse_args() which is responsible for parsing the arguments to the program in the main.cpp file has been altered to read in the additional arguments:

- mode: Telling the program if it is behaving normally or under one of the seven attacks
- bandwidth: The affected bandwidth that is under attack (if it is attacked)
- First attacked segment: The attacked radiofrequency
- Second attacked segment: The second radiofrequency that is attacked (depending on the attack)

The command to start the executable therefore changed from (... representing different arguments):

```
es_sensor ... min_freq max_freq
```

to:

```
es_sensor ... -v {mode} -j {bandwidth} min_freq max_freq \
{First attacked segment} {second attacked segment}
```

The previous arguments and their corresponding setters and getters are added to the configuration in the ElectrosenseContext.cpp and the corresponding header file. The arguments are saved and can be accessed later. Depending on the bandwidth, the attacks are executed in a different manner. The program therefore has the evaluate the affected frequency range and act accordingly. If the bandwidth is above 2MHz, the attack is performed in the rtlsdr driver component, otherwise in the FFT component. The impact in the FFT module is proportional to the affected range, which is calculated by the FFT size (default is 2^8) divided by the ratio of the attack bandwidth to the maximum range of 2MHz. The FFT size describes the amount of bins used for dividing the segment, thus the frequency resolution. The attack impact is therefore $impact = \frac{256}{2'000'000/\text{attack bandwidth}}$. Finally, the configuration gets printed at startup to check for any errors. After setting up, the actual sensing module will perform the behavior as configured.

As an alternative to the waterplots of the ElectroSense web front-end, different intermediate print statements have been implemented to supervise the faultless functioning of the program. As the attacks in the rtlsdrDriver component have proven to do what they are supposed to do in the frontend, the only thing left to check when running the

attacks was the flawless flow of the program. This is monitored by printing a statement to the output after every successful cycle through the whole spectrum, executed by the SequentialHopping.cpp file. It has not been possible to verify the implementation of the attacks into the FFT.cpp visually with the waterplots, since the web front-end has not been reliably working since their creation. The alternative ways of logging the desired behavior have been positive.

Chapter 6

Detection Implementation

The previous chapter has explained the way the new SSDF attacks have been implemented. Subsequently, the system has to be observed while it is running under the new circumstances, which is reviewed in the first part. The second part presents the following ML framework that consists of data cleaning, feature extraction and ML training to detect anomalies in the newly generated data. The presentation of the code does not imply the creation of the files, which are mainly done by Chao Feng in the master thesis [9] created at the Communication Systems Group at the University of Zürich (UZH). Several adjustments have been made, mainly in the monitoring due to different configurations and in the feature extraction because of different procedures, but the main creator of large parts of the code is Chao Feng. Nevertheless, the code used in this work for monitoring ([48]) and the ML framework ([49]) can be found in on github.

6.1 System monitoring

There exist different tools to keep track of the system calls, one of them being the bash command perf trace, which is used in this configuration. The main goal of the system monitoring is to create valuable datasets that can evaluate the implementation of the SSDF attacks based on behavioral fingerprinting with system calls.

6.1.1 Implementation

The scripts which have been used in [9] were adapted to make it suitable for this configuration and setup. Instead of restarting the service, the executable has to be rerun to configure the desired format. The monitoring consists of three files:

- start_monitor.sh: The main script to start the data acquisition.
- monitor.sh: The script that does the actual monitoring.
- preprocessing.py: The python script that turns the heavy txt file into a more suitable csv file.

start_monitor.sh

This bash script is used to start the data acquisition. It is recommended to be run in a window inside the shell (with the bash command `screen` [50]) so it does not abort when the ssh connection to the Raspberry PI is lost. The output when run with screen can also be redirected to a file, which ensures appropriate logging. The script essentially performs a massive automated data collection with two nested for loops, which run through the desired time windows and bandwidths. For every configuration, it creates an appropriate folder and runs the sensing program with all behaviors. Once the service is running, the monitor.sh script is executed to monitor the actual system calls.

Algorithm 9 start_monitor.sh

```
for sample durations do
    for bandwidths do
        Create folder
        for every behavior do
            Rerun executable with behavior
            execute monitor.sh
        end for
    end for
end for
```

monitor.sh

After the sensing executable is started appropriately, the monitor.sh script is run. Inside the script, there is a loop that runs for a predetermined amount of times to get an appropriate amount of samples for every behavior. Once the process ID is found, the system is monitored with the most important line in the monitoring part:

```
timeout -s 1 ${time_window} perf trace -o /data/$2/raw/${path}.txt \
-e !nanosleep -T -p ${pid}
```

The timeout command regulates the time until the next command is run, in this occasion the monitoring command perf trace. All system calls except nanosleep, as it occurs too often and is therefore not valuable later on, made by the process and its sub-processes are written to a text file. A line in the text file contains the timestamp, system call, the arguments and de return value. Once the observing command finishes, the text file is converted to a csv file with the following script.

Algorithm 10 monitor.sh($time$)

```
while count < total samples do
    PID = process ID of es_sensor process
    perf trace the process PID for time seconds except nanosleep
    execute preprocessing.py
    count++
end while
```

preprocessing.py

This python program reads all the lines from the text files and filters them. It loops over all the data which looks like:

```
913236911.638 ( 0.013 ms): es_sensor/28227 timerfd_settime \
(arg0: 8, arg1: 1, arg2: 1978215436, arg3: 0, arg4: 1959791344, \
arg5: 8598792) = 0
```

and filters only the process id, timestamp, system call and time cost out, which leads to:

```
es_sensor/28227,913236911.638,timerfd_settime,0.013
```

These get written to a new file which is saved as a csv.

6.2 Detection

Every dataset has a folder named with the timestamp, the manipulated component, the bandwidth and the monitoring time, with the data which has been gathered on the Raspberry PI and is stored in the subfolder called ”raw”. The folder is moved with secure copy to another computer for the next steps. Here the actual steps to train a ML algorithm in an effort to detect the different behaviors are performed. This starts of with further data cleaning and feature extraction. After that, the algorithms are trained and the performance evaluated.

6.2.1 Feature extraction

The feature extraction is the first step after the data has been transferred and is performed by the get_features.py file. The raw data for the given folder is read in and is saved in a pickle file for future reuse. While reading the raw data, the system calls are set side to side with the official system calls (obtained by the bash command ‘man syscalls‘) to filter out any erroneous data. With the help of the sklearn library, vectorizers are created and saved into their corresponding folder. The dictionary encoding creates a copy of every system call trace for each sample and replaces the instruction with the corresponding unique index. The one-hot encoding function does the same but replaces the string with a list containing only 0’s except a 1 at the corresponding index. The sequence features only consider the first 2500 instructions which roughly correlates to a second of monitoring due to the otherwise massive feature dimensions. The vectorizers then create the bag-of-words features and append all to a dataframe containing all samples with the designated features. Finally, the features are written to an individual csv file for later reuse. Figure 6.1 visualizes the procedure of the get_features.py file.

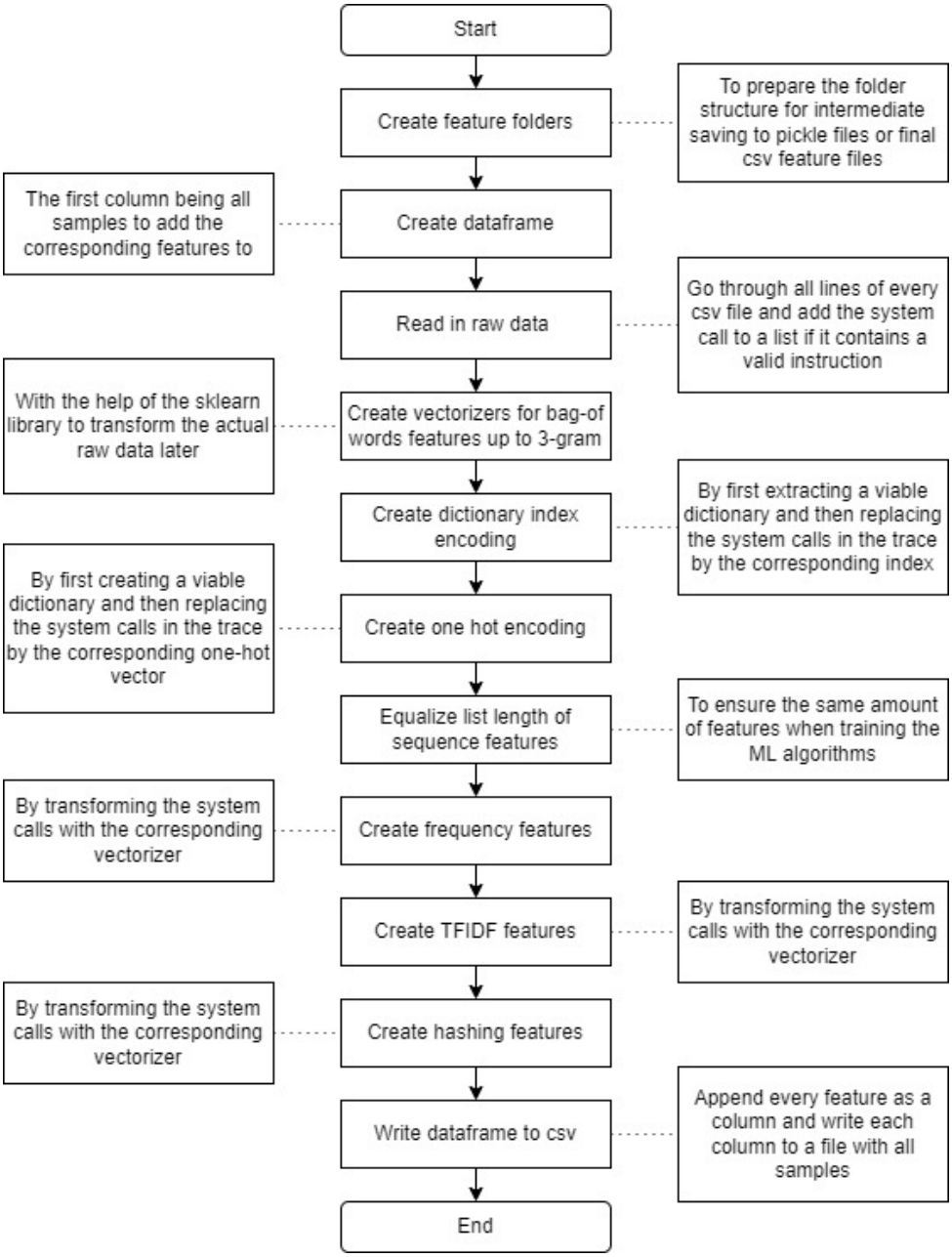


Figure 6.1: Feature extraction program flow

Scaling

Worth mentioning as well is the scaling.py script that reads the applicable bag-of-words approaches, which are the frequency and TFIDF n-grams, and scales them. This is done by reading in the regular features, creating a StandardScaler with the sklearn library and scaling the data according to the normal behavior to test its performance later on.

6.2.2 ML algorithms

Creating the final ML models is done in the ml.py file. Running the script with the folder name as an argument will read in the available features for the corresponding bandwidth. The script runs through all the features, reshapes them if needed and creates the training set based on the 300 normal behavior samples monitored. The script iterates through the ML algorithms mentioned in Chapter 4.2.2 and splits the training set into a 70% training data and 30% validation data. The TNR results from predicting the validation set. Once trained, the different ML algorithms predict the malicious data and get the corresponding TPR. Finally for each algorithm, a total TPR will be assigned. The pseudo code in Algorithm 11 visualizes the described procedure.

Algorithm 11 ML training

```
for every feature do
    Create train set containing all 300 normal samples
    Create list of test sets containing 7x300 malicious samples
    for every ML algorithm do
        Create train/test split from train data
        Get TNR by predicting test set from train data
        for every malicious behavior do
            Get individual TPR by predicting all 300 samples
        end for
        Get total TPR by predicting 2100 sets of malicious behaviors
    end for
end for
```

6.2.3 Visualization scripts

The last files created in the detection part are the python files used for visualization. Analyzing the raw data has been done by creating a dataframe filled with the frequency of each system call in every cycle for every behavior. This dataframe has been used to create the heatmap and system call evaluation in the next chapter. The performance evaluation has been done by creating two 4d arrays with the dimensions bandwidths x models x features x behaviors and have been filled with the corresponding TPR and TNR. Iterating through the array and averaging resulted into the different evaluations that will follow.

Chapter 7

Evaluation

The previous chapters have shown the process of implementing the proposed system. The final step of the framework is the evaluation and visualization of the data. To follow the previous separation of attack implementation and detection, the following chapter is split into the two sections again.

7.1 SSDF Attacks

Evaluating the newly implemented attacks can be done by reviewing the plots generated from the sensory data sent to the backend. As the backend has not been running stable since the final creation of the attacks, only provisional screenshots of six of the attacks can be presented. The delay attack and the creation of the malicious behavior in the FFT.cpp file have been verified by reviewing the data inside the program. Nevertheless, the following six figures present the repeat(Figure 7.1), mimic (Figure 7.2), confusion (Figure 7.3), noise (Figure 7.4), spoof (Figure 7.5) and freeze (Figure 7.6) attack. The attacks were sequentially executed for ten minutes.

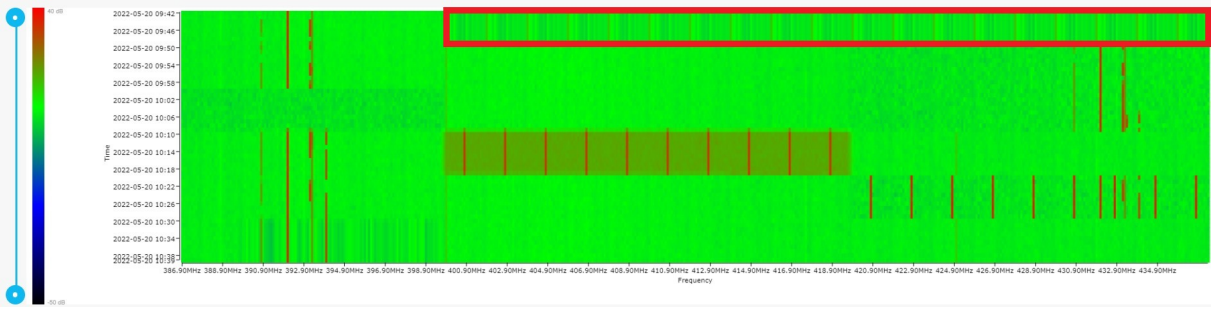


Figure 7.1: Waterplot of ElectroSense frontend: Repeat attack highlighted

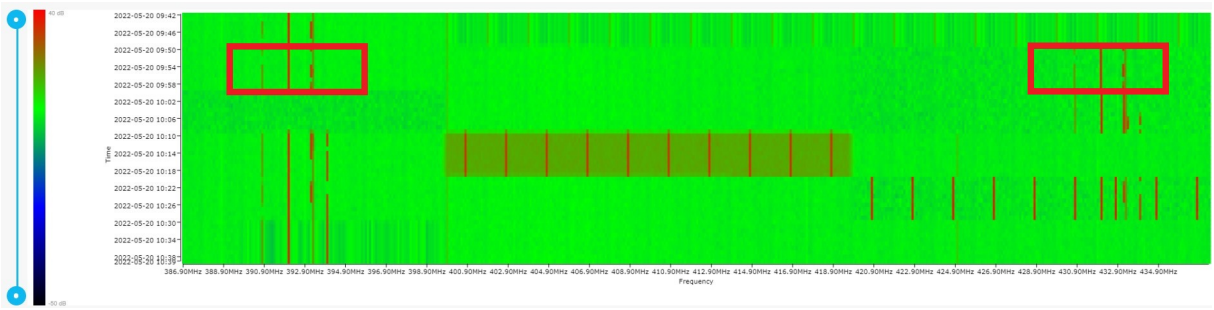


Figure 7.2: Waterplot of ElectroSense frontend: Mimic attack highlighted

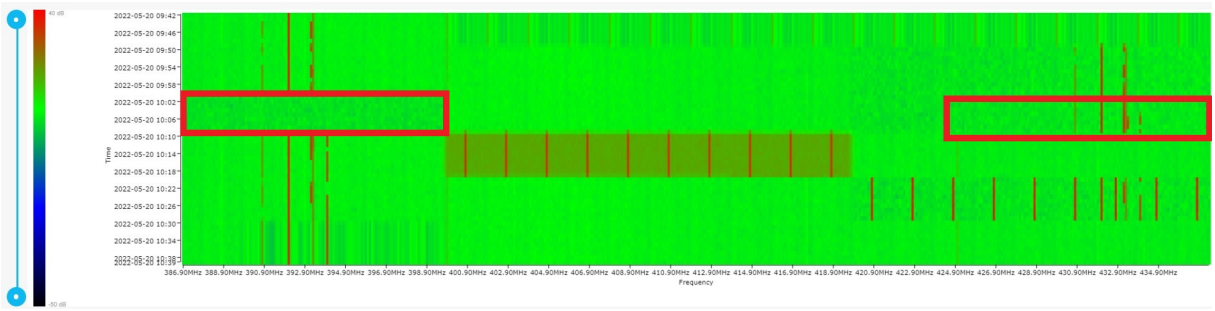


Figure 7.3: Waterplot of ElectroSense frontend: Confusion attack highlighted

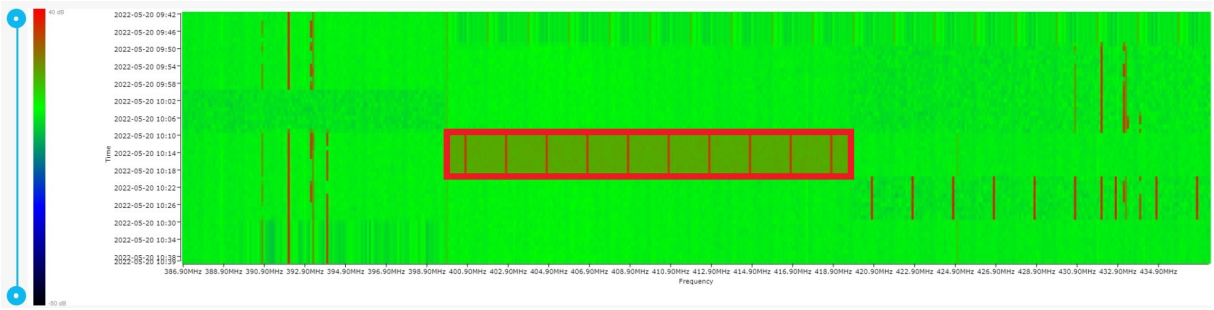


Figure 7.4: Waterplot of ElectroSense frontend: Noise attack highlighted

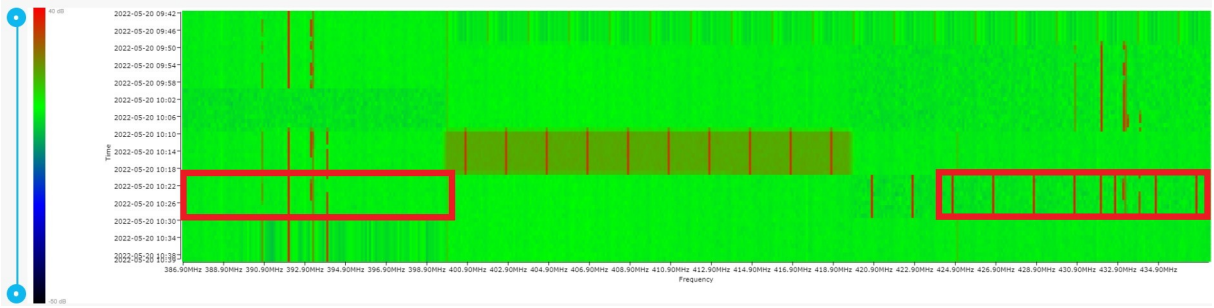


Figure 7.5: Waterplot of ElectroSense frontend: Spoof attack highlighted

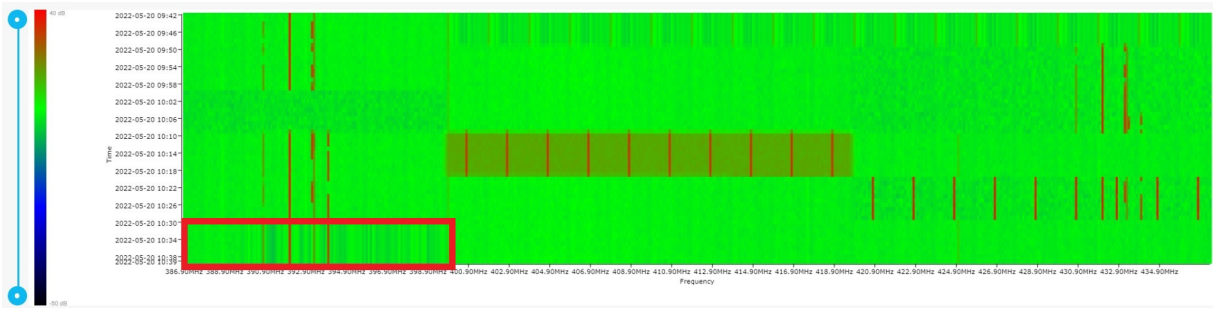


Figure 7.6: Waterplot of ElectroSense frontend: Freeze attack highlighted

7.2 Detection

The section evaluating the detection results presents multiple different aspects to review, as every step of the framework generated data which was assessed.

7.2.1 Data exploration

First is the raw data resulting from the monitoring step. Large sets of files for various configurations were generated. The files contain the preprocessed system calls row by row which can be analyzed and compared. Figure 7.7 shows a heatmap for an attacked bandwidth of 200kHz which has been monitored for 60 seconds. Having a look at the heatmap for the different behaviors, the normal and infected data can not be distinguished just by eye. In this case though, reviewing the actual numerical data in Table 7.1 instead of the visualization provides value as there seems to be some differences which do not show in the graphic. System calls which are called less than 30 times are omitted.

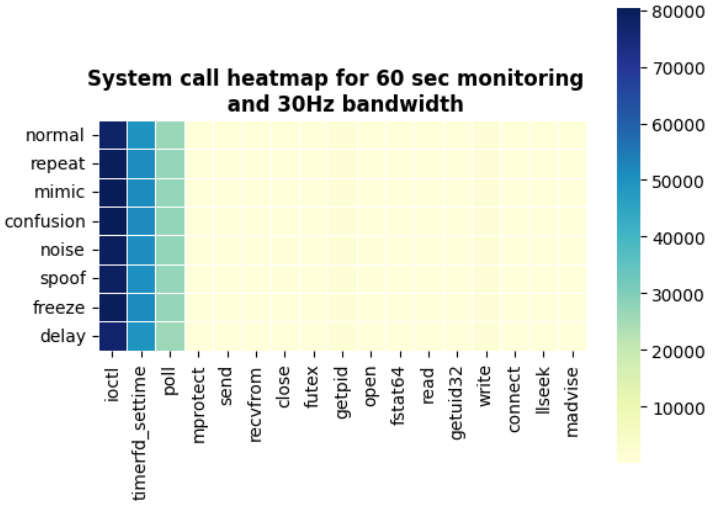
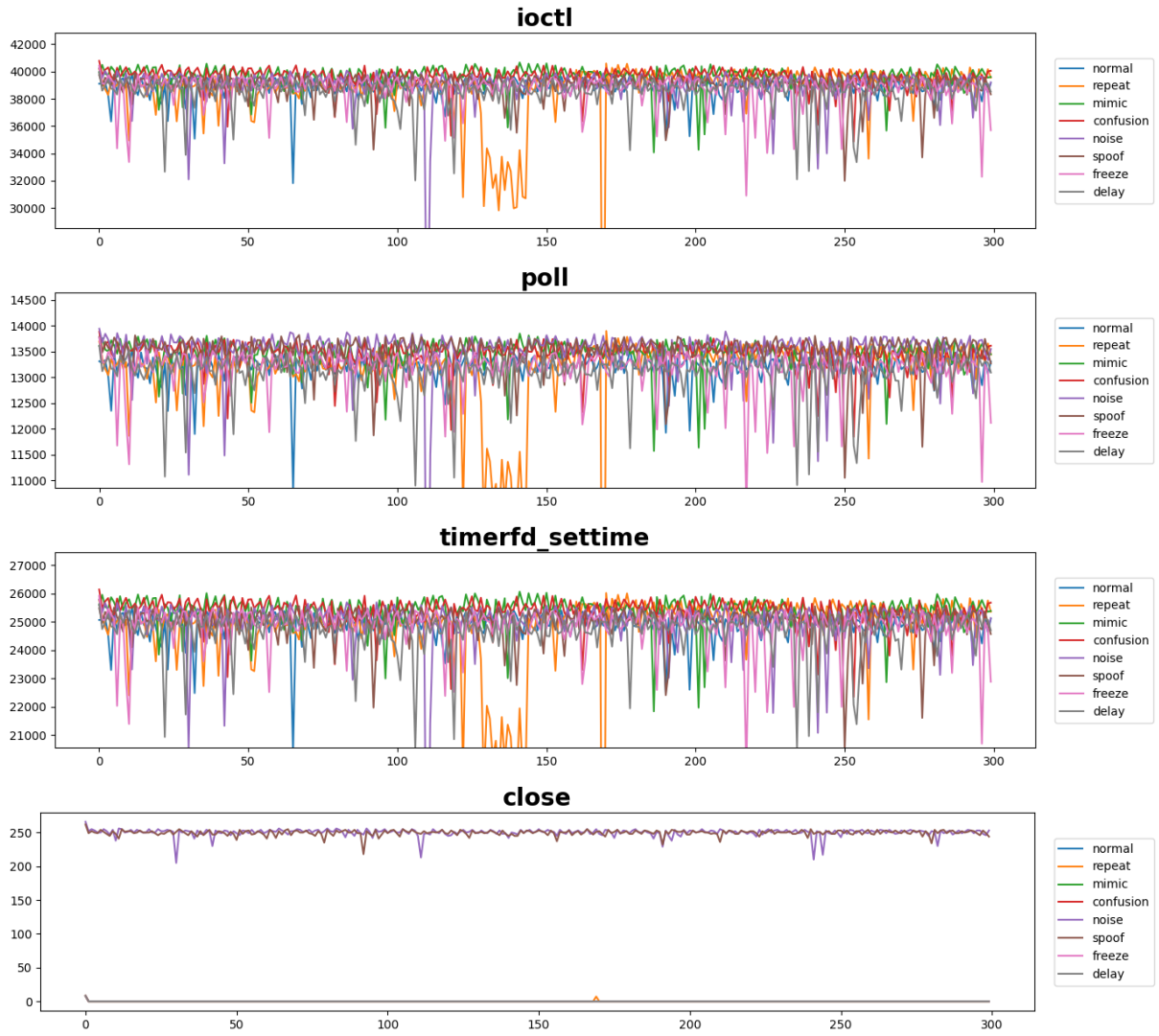


Figure 7.7: System call frequency heatmap for SSDF attacks implemented with variables

Table 7.1: Table of system calls for 60 seconds of monitoring

mode	ioctl	poll	timerfd_settime	close	futex	getpid	open	read	write
normal	77862	26482	49910	8	44	1086	6	30	1088
repeat	79649	27159	51053	8	51	1108	7	29	1185
mimic	80268	27316	51453	8	61	1118	6	29	1138
confusion	80440	27359	51566	8	49	1118	7	29	1122
noise	79345	27484	50863	516	25	1104	513	27	1115
spoof	78857	27315	50546	511	37	1094	510	29	1107
freeze	79992	27243	51277	8	67	1113	6	29	1124
delay	77073	26210	49403	8	41	1102	6	29	1116

Analyzing the data in Table 7.1, shows a few attackts seem to have almost identical behavior as the normal mode, while noise and spoof clearly differ from the conventional behavior as the instructions `open` and `close` occur more often. As this only shows the first 60 seconds, it is useful to examine the system call evaluation over time for the most important system calls `ioctl`, `poll`, `timerf\_settime`, `close`, `open` and `write`.



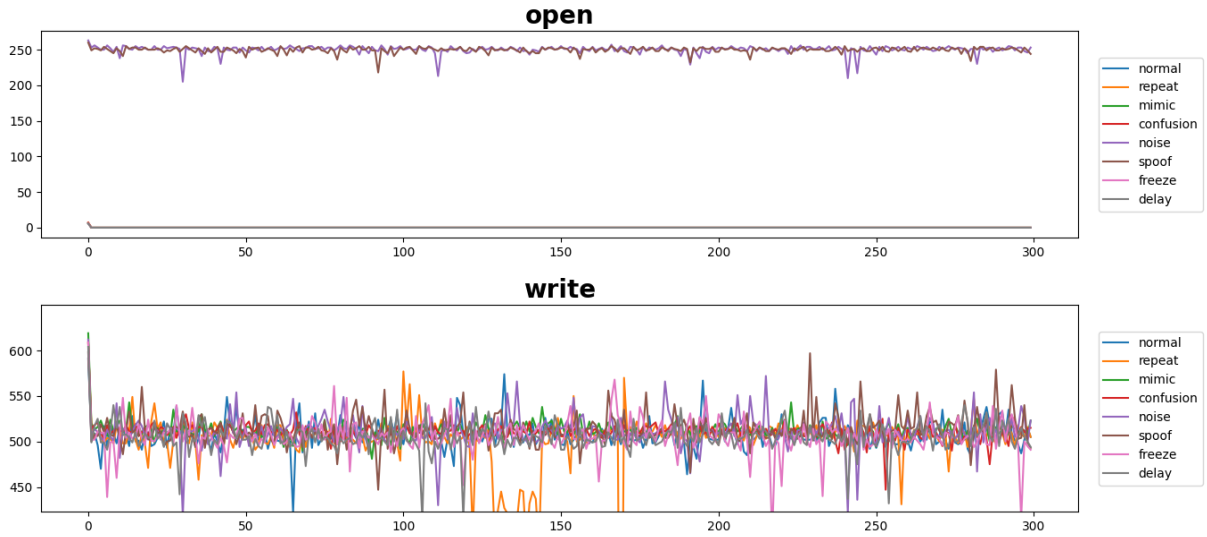


Figure 7.8: Evaluation of the system calls ioctl, poll, timerfd_settime, close, open and write

The subplots in Figure 7.8 display the 300 sequential samples for six different system calls of each behavior. The previously observed anomalies in the `open` and `close` system calls are independent of time and stay somewhat constant. This is an indication that continuously a file is opened and closed which had not been evaluated as an issue at the time. The noise and spoof attack rely on files, although it is not by saving temporary sensing data but to access a random number generator. Despite the fact of this not fulfilling the implementation requirements, in hindsight, it even turned out to be beneficiary due to the ability to compare the behaviors. The system calls `ioctl`, `poll`, `timerf\_settime` and `write` are constant over time as well with a noticeable amount of outliers.

7.2.2 Feature comparison

Creating different features and training the algorithms with the data explored above leads to various results. As mentioned in Chapter 4.2.2, the performance is evaluated with the TNR and TPR. In general, only models are taken into consideration if they reach a TNR of 85% and above, unless it is stated otherwise. The TNR signifies that the model has been able to learn and classify the normal data correctly. Given this condition, the TPR represents the accuracy of the anomaly detection. The first variable to assess and find the best detection variation is the best performing feature(s). Bag-of-words features can be scaled to decrease the range of the feature values. Figure 7.9 shows the difference between the average TPR for all models considered with an attack range of 200kHz with scaled (blue bars) and not scaled (orange bars) bag-of-words features.

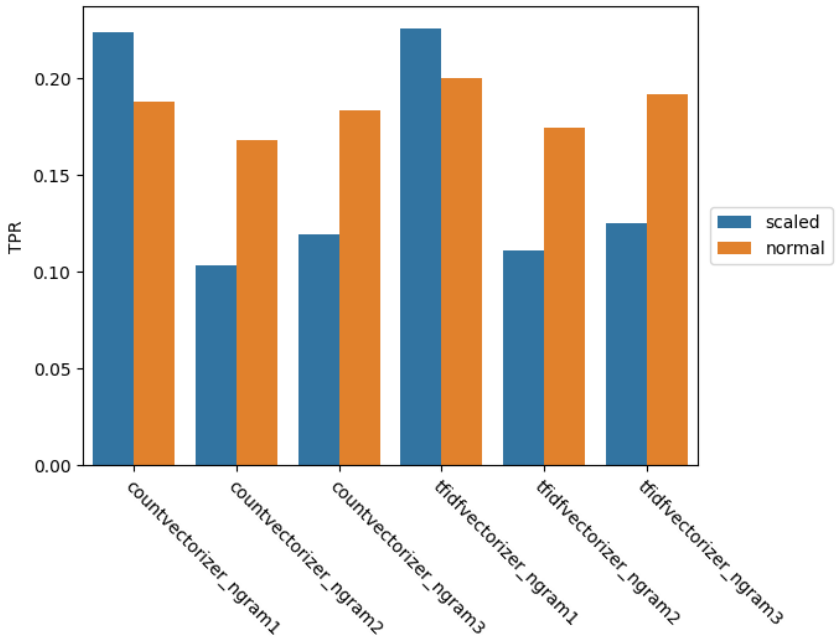


Figure 7.9: Feature comparison: Scaled and normal

Only the unigram variations perform better with scaling, possibly as it has the biggest influence on those feature values. When using those features, scaling might therefore have a positive influence on the detection, although in general, not scaling will be considered more accurate from here on forward.

Figure 7.10 shows the comparison between the features for all bandwidths across all ML algorithms. Noticeable are the weak performances by the sequence features with dictionary index-encoding and one-hot-encoding. The best TPR are reached by the 3-gram frequency feature and the TFIDF unigram. The trend for the frequency feature seems to be increasing which might imply an even better performance for higher n-grams.

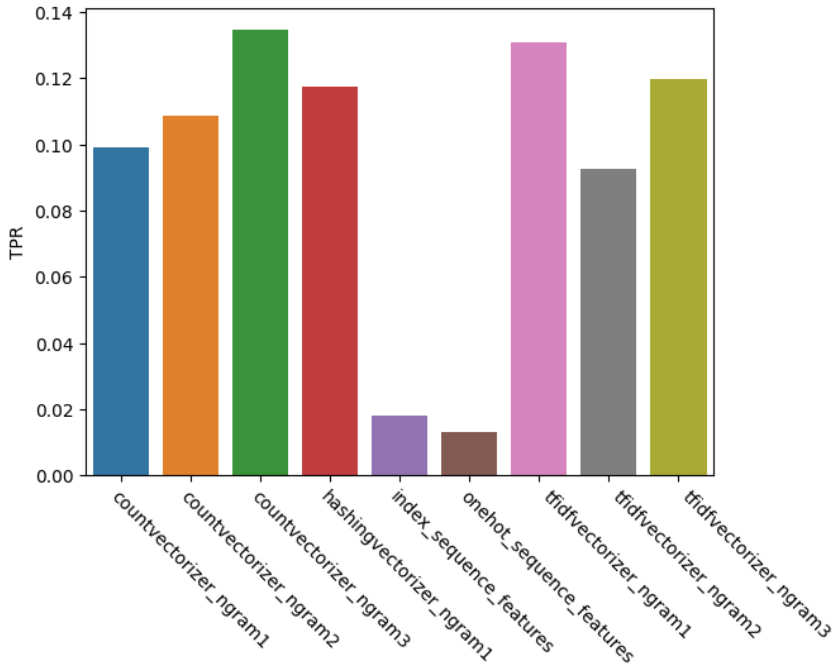


Figure 7.10: Feature comparison: Different features

The reason of the sequence features performing so bad is most likely due to the poor dimensionality reduction. As the computing power has not been enough to train models in a reasonable amount of time with the sequence features, the monitoring segments have been shortened to approximately a second. It is quite likely the attack was not performed during this second depending on the attacking bandwidth.

7.2.3 Model comparison

Once the best performing features were evaluated, the TPR and TNR of the different models were compared. TNR can be seen as how good the model was able to train and the TNR judges the anomaly detection performance. The comparison of the average of training with all features over all bandwidths for the five different models is visualized in Figure 7.11.

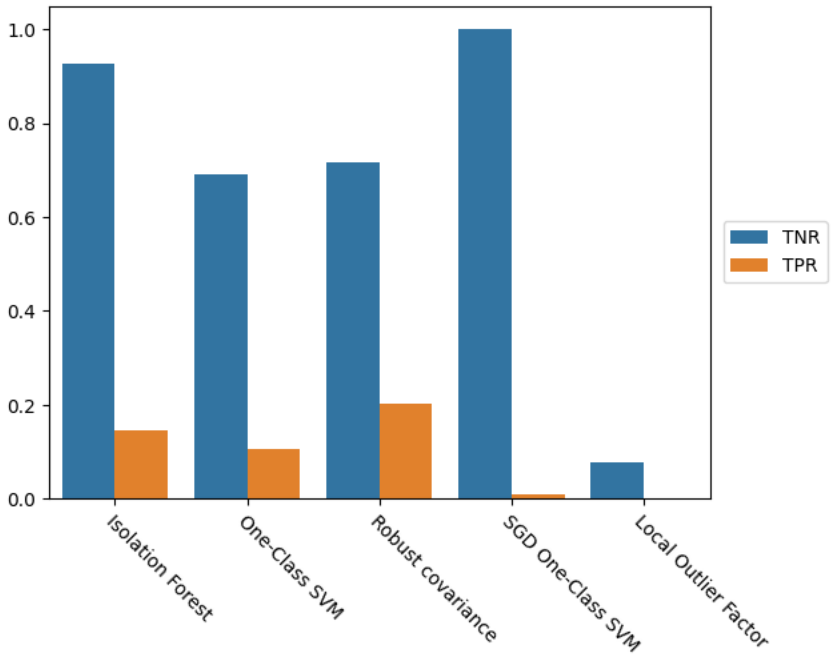


Figure 7.11: Model TPR & TNR comparison

Figure 7.11 shows some distinct takeaways. Firstly, the Isolation Forest model can classify the normal data correctly above 90% of the time. This makes it a valid detection method, although the actual anomalies can only be detected less then 20%. The One-Class SVM and Robust covariance TNR can be assessed as rather mediocre, while the RC shows the best anomaly detection rate. Admittedly, the TPR is still very underwhelming and is not representative due to the low TNR. The SGD One-Class SVM does train extraordinaly good but possibly because it classifies everything as normal behavior by evaluating the TPR. The Local Outlier Factor performed very odd, as it classified exactly the opposite and could not be improved by tuning the hyperparameters. In general, the TPR could be enhanced by a larger amount of data samples to train the algorithms better and classify normal data more accurate.

Reviewing the average TPR per bandwidth for each model trained with all features in Figure 7.12 shows the best detection possibilities in the 200kHz area. Furthermore, a slight trend from 20kHz to 200kHz and from 20MHz upwards can be detected. The Local Outlier Factor proves to be not suited for the case as it does not show up on the plot.

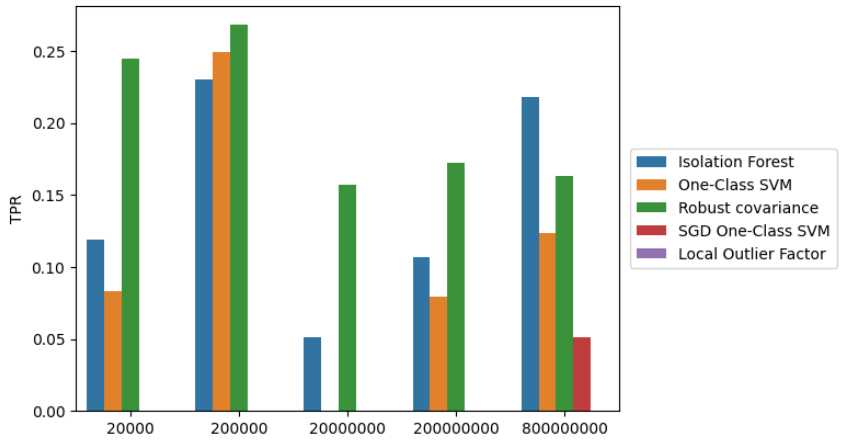


Figure 7.12: Model TPR comparison

7.2.4 Attack comparison

The frequency 3-gram feature has shown to be the best detecting feature, which is why the total detection performance for the different models will be evaluated with that specific approach. Assessing the TPR for all the malicious behaviors combined in Figure 7.13, the detection is bad with a maximal TPR of almost 50% for the 800MHz attack bandwidth. It can therefore be concluded, that the implementation with variables was successful and is not being reliably detected. Another recurring tendency is the incline after 20kHz and 20MHz. This can be explained by the impact the SSDF attack has in the corresponding file its executed in. The amount of data points that need to be modified increases proportionally with the bandwidth. A small bandwidth like 20kHz performs only a few value changes (1% of the values in FFT.cpp), while an attack ranging 200kHz has a much larger impact (10%). This might lead to an increased detection rate. Once the the point of inflection at 2MHz is reached, the attack is executed in the rtlsdrDriver file. Only a small amount of values are manipulated again, which leads to a drop of detection performance at the next bandwidth, but starts to increase again. The bandwidth and the amount of data manipulated associated with it therefore seems to have a connection with the detection performance.

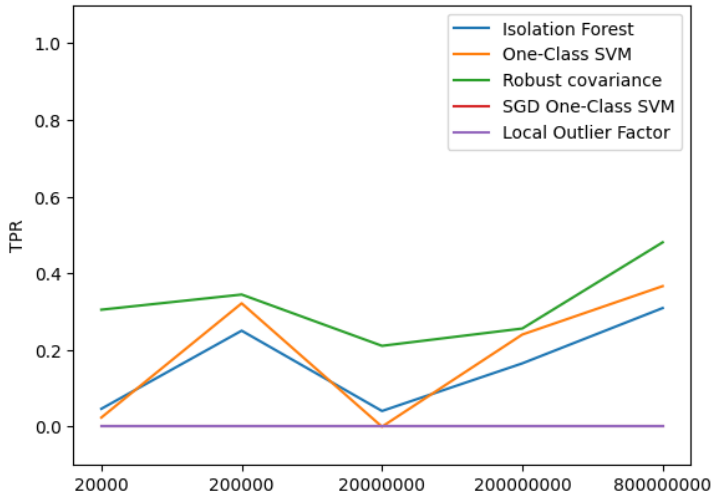


Figure 7.13: Total TPR of malicious behaviors with frequency 3-gram

Finally, the different behaviors can be compared with eachother using the frequency 3-gram feature encoding. The following group of graphics in Figure 7.14 show the TPR for the malicious behaviors and TNR of the normal functioning for the different models across the observed attacking bandwidths. The TNR all suggest a viable training of the models except for the LOF. Assessing the differences between the behaviors, spoof and noise stand out. As the data exploration has yielded, the system calls `open` and `close`, those attacks generate deviating system calls due to their dependence on a file to generate random noise. The RC algorithm detects those anomalies in the lower bandwidths perfectly with a 100% TPR. Attacks that involve two segments, mimic, confusion and spoof furthermore have an increased detection rate at the 800MHz bandwidth possibly due to the fact that almost every datapoint needs to be manipulated.

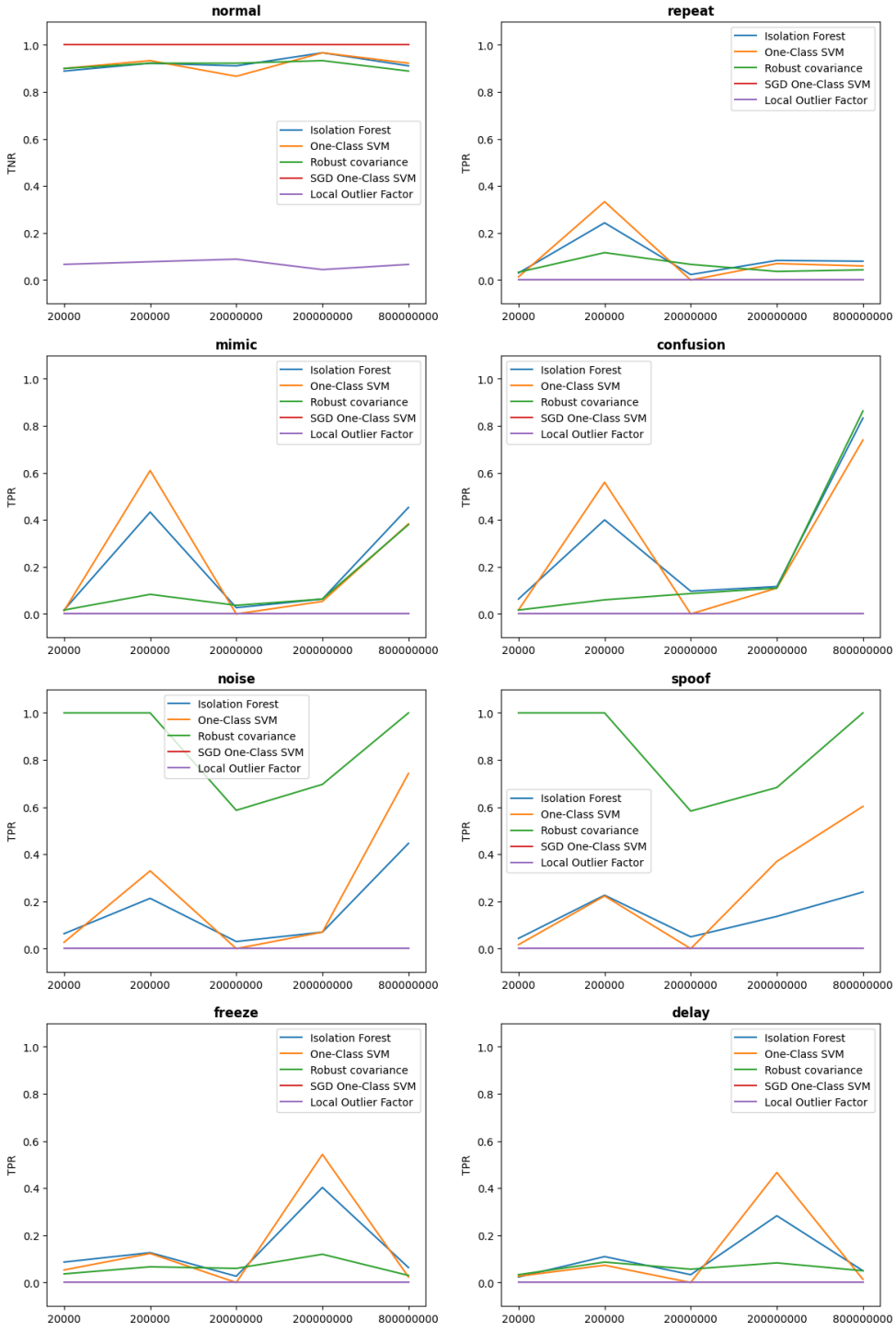


Figure 7.14: Comparison of the different behaviors

7.3 Comparison with previous study

Figure 7.15 shows the heatmap of the system calls for the SSDF attacks implemented with files. The data is from monitoring 60 seconds running in the same device as in this thesis,

the Raspberry PI 3 Model B. The bandwidth of the attack is unknown, but it shows a significant difference between normal and attack data. This data visualization suggests that the attack and normal data are linearly separable, which has been proven to be valid [9]. The SSDF attacks proposed in this thesis on the other hand generate a significantly less colorful heatmap as seen in Figure 7.7.

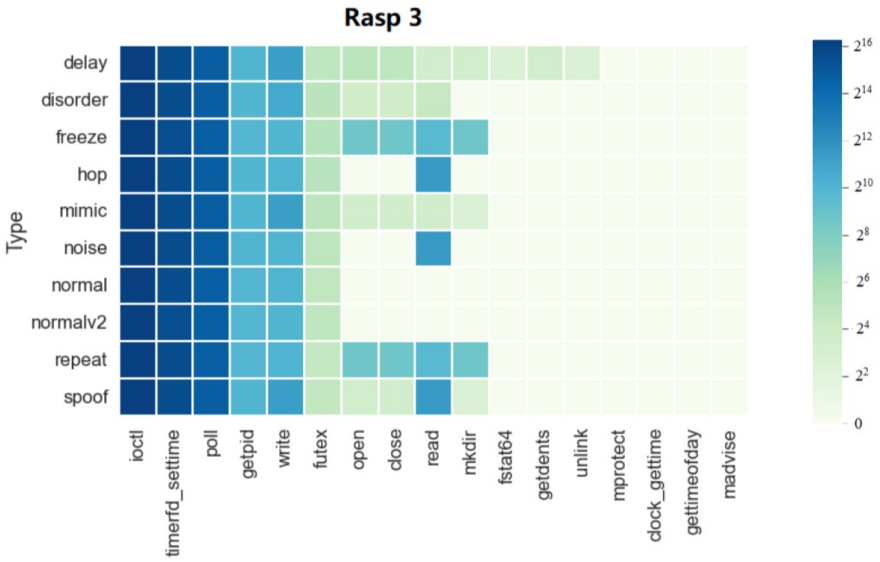


Figure 7.15: System call frequency heatmap for SSDF attacks implemented with files

Chapter 8

Summary, Conclusions and Future Work

The final chapter summarizes the thesis and concludes the final takeaways including the limitations. Following from that, a proposition for some possible future work is done.

8.1 Summary and Conclusions

The thesis had two main goals: First, a new implementation of SSDF attacks to check the robustness of existing detection techniques. In order to achieve this objective, a sensory device being susceptible to the given malicious behavior had to be setup. This was done with the help of the open-source crowd-sourced platform ElectroSense. A sensory device consisting of an antenna connected to a Raspberry PI was deployed connected to the internet. Via ssh connection, the device was evaluated in a first step. The ElectroSense source code and previous implementations of the attack provided a good base to start. From there, the attacks were implemented not relying on writing the sensing data to a temporary file but making use of variables inside the program. First implementations could then be assessed with the help of waterplots in the ElectroSense web front-end, which later on had to be replaced by print statements to the terminal due to unreliable uptime. Nonetheless, the attacks were implemented in the intended program sections and could be run user friendly thanks to some other source code manipulations. In hindsight, two of the attacks seemed to have a little flaw, since the noise and spoof attack still relied on accessing a file, although be it not to save temporary sensing data, but to generate a random number from a distribution to add to the proper sensing data.

With the first goal partly accomplished, the next phase was to implement a ML framework based on gathered data from system call monitoring. Based on previous work done at the Communication Systems Group at UZH [9] a system monitoring module was created, consisting of a main program, monitoring script and a preprocessing python file. Roughly a week of data was collected for each SSDF attack and the normal behavior of the sensor for different attacking bandwidths. This raw data then went through a ML pipeline consisting of data cleaning and feature extraction. Different NLP techniques have been applied to create numerical values from the system call traces. The final step was the

training of different ML algorithms to predict normal or malicious behavior with mixed results. Considering the findings in Chapter 7, the main conclusions to be drawn are:

- Five of seven SSDF attacks have been implemented successfully with using variables instead of files stored in disk into the ElectroSense sensor software.
- The novel implementations are successful in a sense that promising detection techniques like behavioral fingerprinting based on system calls combined with ML algorithms can not reliable detect the new SSDF attacks. This might be due to the fact that the new implementation stays in user space from a memory point of view. Accessing files is what requires higher privileges and therefore needs a system call to access the kernel space. Anomalies like this are increasing the chance of detecting the attack.
- The bandwidth of an attack and therefore the overall impact on the system has an influence on the detection performance.

8.2 Future Work

The goal of this thesis generally speaking was to implement the SSDF attacks based on another criterion and check its detection with a behavioral fingerprinting technique based on system calls. It has been shown, that monitoring the system calls with the configurations used in this thesis, has not been an accurate observing approach to detect and prevent the newly implemented SSDF attacks. This thesis has been limited by computational power so it was not capable of implementing all proposed configurations in a proper way. Although the direction of this approach did not look promising, increasing the computational power and changing some variables might open new doors. This has to be treated with caution though as the IoT sensors are resource limited anyway. Since behavioral fingerprinting has been proposed as a promising method, another way of creating this fingerprint can be evaluated with the latest attacks. Instead of kernel space traces like the system calls, another possibility would be to monitor user space traces like function calls.

Bibliography

[1] N. Marchang and W. N. Singh, "A Review on Spectrum Allocation in Cognitive Radio Network," *International Journal of Communication Networks and Distributed Systems*, vol. 23, no. 1, p. 1, 2019.

[2] T. Yucek and H. Arslan, "A survey of spectrum sensing algorithms for cognitive radio applications," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 1, pp. 116–130, 2009.

[3] S. Rajendran, R. Calvo-Palomino, M. Fuchs, B. Van den Bergh, H. Cordobes, D. Giustiniano, S. Pollin, and V. Lenders, "Electrosense: Open and Big Spectrum Data," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 210–217, Jan. 2018.

[4] J. Pan and Z. Yang, "Cybersecurity Challenges and Opportunities in the New "Edge Computing + IoT" World." ACM, Mar. 2018, pp. 29–32.

[5] J. Li, Z. Feng, Z. Feng, and P. Zhang, "A survey of security issues in Cognitive Radio Networks," *China Communications*, vol. 12, no. 3, pp. 132–150, Mar. 2015, conference Name: China Communications.

[6] M. Hasan, M. M. Islam, M. I. I. Zarif, and M. Hashem, "Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches," *Internet of Things*, vol. 7, p. 100059, Sep. 2019.

[7] P. M. S. Sánchez, J. M. J. Valero, A. H. Celdrán, G. Bovet, M. G. Pérez, and G. M. Pérez, "A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1048–1077, 2021.

[8] A. H. Celdrán, P. M. S. Sánchez, G. Bovet, G. M. Pérez, and B. Stiller, "CyberSpec: Intelligent Behavioral Fingerprinting to Detect Attacks on Crowdsensing Spectrum Sensors," *arXiv:2201.05410 [cs]*, Jan. 2022, arXiv: 2201.05410.

[9] C. Feng, "Intelligent Analysis of System Calls to Detect Cyber Attacks Affecting Spectrum Data Integrity in IoT Sensors," Master's thesis, Universität Zürich, 2022.

[10] J. R. Hoehm, J. C. Gallagher, and K. M. Sayler, "Overview of Department of Defense Use of the Electromagnetic Spectrum," Defense Technical Information Center, Tech. Rep., Aug. 2020.

[11] K. Riad, T. Huang, and L. Ke, "A dynamic and hierarchical access control for IoT in multi-authority cloud storage," *Journal of Network and Computer Applications*, vol. 160, p. 102633, Jun. 2020.

[12] Pratibha, S. Thangjam, N. Kumar, and S. Kumar, "A Survey on Prevention of the Falsification Attacks on Cognitive Radio Networks," *IOP Conference Series: Materials Science and Engineering*, vol. 1033, no. 1, p. 012021, Jan. 2021.

[13] J. N. Soliman, T. A. Mageed, and H. M. El-Hennawy, "Taxonomy of security attacks and threats in cognitive radio networks," in *2017 Japan-Africa Conference on Electronics, Communications and Computers (JAC-ECC)*, Dec. 2017, pp. 127–131.

[14] S. Shrivastava, A. Rajesh, P. K. Bora, B. Chen, M. Dai, X. Lin, and H. Wang, "A survey on security issues in cognitive radio based cooperative sensing," *IET Communications*, vol. 15, no. 7, pp. 875–905, Apr. 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1049/cmu2.12131

[15] B. Bezawada, I. Ray, and I. Ray, "Behavioral fingerprinting of Internet-of-Things devices," *WIREs Data Mining and Knowledge Discovery*, vol. 11, no. 1, p. e1337, 2021.

[16] J. Zhang, K. Zhang, Z. Qin, H. Yin, and Q. Wu, "Sensitive system calls based packed malware variants detection using principal component initialized MultiLayers neural networks," *Cybersecurity*, vol. 1, no. 1, p. 10, Dec. 2018.

[17] "Kernel Space Definition." [Online]. Available: http://www.linfo.org/kernel_space.html

[18] D.-K. Kang, D. Fuller, and V. Honavar, "Learning classifiers for misuse and anomaly detection using a bag of system calls representation," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, Jun. 2005, pp. 118–125.

[19] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*, first edition ed. O'Reilly Media, 2017.

[20] W. Khreich, B. Khosravifar, A. Hamou-Lhadj, and C. Talhi, "An anomaly detection system based on variable N-gram features and one-class SVM," *Information and Software Technology*, vol. 91, pp. 186–197, Nov. 2017.

[21] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, Jul. 2009.

[22] C. Aggarwal, *Outlier analysis*, 2nd ed. New York, NY: Springer Science+Business Media, 2016.

[23] Z. Cheng, C. Zou, and J. Dong, "Outlier detection using isolation forest and local outlier factor," in *Proceedings of the Conference on Research in Adaptive and Convergent Systems*. Chongqing China: ACM, Sep. 2019, pp. 161–168.

[24] N. Elmrabit, F. Zhou, F. Li, and H. Zhou, "Evaluation of Machine Learning Algorithms for Anomaly Detection," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, Jun. 2020, pp. 1–8.

[25] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *2008 Eighth IEEE International Conference on Data Mining*, Dec. 2008, pp. 413–422.

[26] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support Vector Method for Novelty Detection," in *Advances in Neural Information Processing Systems*, vol. 12. MIT Press, 1999.

[27] P. J. Rousseeuw and K. V. Driessen, "A Fast Algorithm for the Minimum Covariance Determinant Estimator," *Technometrics*, vol. 41, no. 3, pp. 212–223, Aug. 1999.

[28] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 93–104, Jun. 2000. [Online]. Available: https://dl.acm.org/doi/10.1145/335191.335388

[29] F. Salahdine and N. Kaabouch, "Security threats, detection, and countermeasures for physical layer in cognitive radio networks: A survey," *Physical Communication*, vol. 39, p. 101001, Apr. 2020.

[30] L. Zhang, G. Ding, Q. Wu, Y. Zou, Z. Han, and J. Wang, "Byzantine Attack and Defense in Cognitive Radio Networks: A Survey," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1342–1363, 2015.

[31] A. G. Fragkiadakis, E. Z. Tragos, and I. G. Askoxylakis, "A Survey on Security Threats and Detection Techniques in Cognitive Radio Networks," *IEEE Communications Surveys Tutorials*, vol. 15, no. 1, pp. 428–445, 2013.

[32] P. Kaligineedi, M. Khabbazian, and V. K. Bhargava, "Malicious User Detection in a Cognitive Radio Cooperative Sensing System," *IEEE Transactions on Wireless Communications*, vol. 9, no. 8, pp. 2488–2497, Aug. 2010.

[33] R. Chen, J.-M. Park, and J. H. Reed, "Defense against Primary User Emulation Attacks in Cognitive Radio Networks," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 1, pp. 25–37, Jan. 2008, conference Name: IEEE Journal on Selected Areas in Communications.

[34] J. Kelly and J. Ashdown, "Spectrum Sensing Falsification Detection in Dense Cognitive Radio Networks using a Greedy Method," in *NAECON 2018 - IEEE National Aerospace and Electronics Conference*, Jul. 2018, pp. 144–151.

[35] R. Chen, J.-M. Park, and K. Bian, "Robust Distributed Spectrum Sensing in Cognitive Radio Networks," in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, Apr. 2008, pp. 1876–1884.

[36] P. K. Varshney, *Distributed Detection and Data Fusion*. Springer Science & Business Media, 1997.

[37] Z. Gao, X. Huang, and M. Wang, "An heuristic WSPRT fusion algorithm against high proportion of malicious users," in *2015 International Conference on Wireless Communications & Signal Processing (WCSP)*, Oct. 2015, pp. 1–5.

[38] W. Wang, H. Li, Y. Sun, and Z. Han, "CatchIt: Detect Malicious Nodes in Collaborative Spectrum Sensing," in *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*, Nov. 2009, pp. 1–6, iSSN: 1930-529X.

[39] Y. Fu and Z. He, "Bayesian-Inference-Based Sliding Window Trust Model Against Probabilistic SSDF Attack in Cognitive Radio Networks," *IEEE Systems Journal*, vol. 14, no. 2, pp. 1764–1775, Jun. 2020.

[40] P. S. Chatterjee and M. Roy, "Maximum match filtering algorithm to defend spectrum-sensing data falsification attack in CWSN," *International Journal of Wireless and Mobile Computing*, vol. 15, no. 2, p. 113, 2018.

[41] S. Nath, N. Marchang, and A. Taggu, "Mitigating SSDF attack using k-medoids clustering in Cognitive Radio Networks," in *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct. 2015, pp. 275–282.

[42] C. Zhao, W. Wang, L. Huang, and Y. Yao, "Anti-PUE Attack Base on the Transmitter Fingerprint Identification in Cognitive Radio," in *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*, Sep. 2009, pp. 1–5.

[43] "Visual Studio Code: Developing on Remote Machines using SSH." [Online]. Available: https://code.visualstudio.com/docs/remote/ssh

[44] "ElectroSense Frontend." [Online]. Available: https://electrosense.org/

[45] "Software used in Electrosense nodes." [Online]. Available: https://github.com/electrosense/es-sensor

[46] C. Feng, "IoT_sensors_security_analysis," Jul. 2022. [Online]. Available: https://github.com/luke-feng/IoT_Sensors_Security_Analysis

[47] "Perf." [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

[48] "SSDF Attacks github." [Online]. Available: https://github.com/RobinWassink/BT_SSDF_Attacks

[49] "Detection github." [Online]. Available: https://github.com/RobinWassink/BT_Detection

[50] "Screen - GNU Project - Free Software Foundation." [Online]. Available: https://www.gnu.org/software/screen/

Abbreviations

IoT	Internet-of-Things
RF	Radiofrequency
SSDF	Spectrum Sensing Data Falsification
ML/DL	Machine Learning and Deep Learning
ML	Machine Learning
CRN	Cognitive Radio Network
PUEA	Primary User Emulation Attack
NLP	Natural Language Processing
PSD	Power Spectrum Data
IF	Isolation Forest
LOF	Local Outlier Factor
OC-SVM	One Class Support Vector Machine
SVM	Support Vector Machine
DoS	Denial of Service
RSS	Received Signal Strength
TPR	True Positive Rate
TNR	True Negative Rate

List of Figures

List of Tables

Appendix A

Installation Guidelines

SSDF Attacks

The code for the SSDF attacks can be found on github [48]. Clone the repository into a Raspberry PI and follow the installation guide in the README.

Detection

The code for monitoring is in the github of the previous subsection as it happens inside the sensory device. The ML framework is available at github as well [49]. The README explains further steps.

Appendix B

Contents of the zip file

The zip file accompanying this thesis contains:

- The final version of the bachelor thesis as a pdf
- The LaTeX source code of this bachelor thesis as a zip
- The slides of the midterm presentation from 05.05.2022