Department of Computer Science

BSCCS Final Year Project Report
2023-2024

23CS102

# CodeBreach: Code-Level Attacks on EM-based Anomaly Detection

(Volume __ of __)

| | | |
|---|---|---|
| Student Name | : | **XIONG Yifan** |
| Student No. | : | **55200406** |
| Programme Code | : | **BSCEGU4** |
| Supervisor | : | **Prof HUANG, Jun** |
| 1st Reader | : | **Prof CHAN, Chung** |
| 2nd Reader | : | **Prof WANG, Cong** |

**For Official Use Only**

## Student Final Year Project Declaration

I have read the project guidelines and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I hereby declare that the work I submitted for my final year project, entitled:

CodeBreach: Code-Level Attacks on EM-based Anomaly Detection

does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Project Guidelines.

| | | | |
|---|---|---|---|
| Student Name: | XIONG Yifan | Signature: | XIONG YIFAN |
| Student ID: | 55200406 | Date: | April 5, 2024 |

# Table Of Content

# Abstract

Embedded systems have become an important component of the modern world, and they are widely used in different industries.

However, recent works demonstrate that embedded systems are vulnerable to cascading attacks. In these circumstances, EM-based anomaly detectors have gained much attention in protecting embedded systems from malicious attacks. This paper describes a new attack approach, CodeBreach, which can inject malicious code blocks with various payloads into embedded systems under the protection of an EM-based anomaly detector. CodeBreach constructs a new injection code block based on the malicious code block. The injection code block can simulate the EM spectrum of the original code with the function of the malicious code block.

This paper presents a new method to cheat the state-of-the-art EM-based anomaly detector. It simulates the workload and structure of the loop to receive a similar pre-iteration time and then tricks the anomaly detector using our falsified loop. Our evaluation achieved a 40% decrease in the detected possibility against the state-of-the-art EM-based anomaly detector.The new approach can help increase knowledge about the EM pattern of program execution and design appropriate countermeasures and better monitors.

# Acknowledgments

I would like to express my heartfelt gratitude to the following individuals for their valuable contributions and support during the research and writing of this paper:

First, I have to extend my most profound appreciation to my supervisor, Prof. Jun Huang, for providing me with this precious opportunity to access the field of embedded systems. Thanks for your unwavering guidance, valuable insights, and encouragement throughout this journey. Besides, I would like to show my sincere appreciation to Mr. Liuqun Zhai. This work would not have beenbe possible without your help, and I apologize for not achieving the results we had hoped for. I also need to thank my roommate, Mr. Yang Lou. I have benefited from your advice and enlightenment. Thanks for your tolerance and understanding over the last nine years, Stacy. Your company means a lot to me.

Finally, I would like to dedicate this paper to my mother, who left me forever at the end of March. I wish there is no pain in heaven.

# Chapter 1: Introduction

Embedded systems have become increasingly popular in various industries, such as industrial automation and consumer electronics. The advancement of microcontroller and microprocessor technologies also promotes the development of more powerful and compact embedded systems. The miniaturization of embedded systems contributes to their wide application in robotics and medical devices. Offensive and defensive battles alternate between protecting embedded systems from growing malicious behaviors. In order to build a comprehensive defense system, it is important to figure out the possible attacks embedded systems face.

In this paper, we describe a new attack approach, CodeBreach, which can inject malicious code blocks with various payloads into embedded systems under the protection of an EM-based anomaly detector. Our attack constructs a new injection code block based on the malicious code block. The injection code block can simulate the EM spectrum of the original code with the function of the malicious code block.

The new approach can help increase knowledge about the EM pattern of program execution and designing appropriate countermeasures and better monitors. In summary, this paper makes the following contributions:

1) Present a new approach, CodeBreach, to inject malicious code into systems under the protection of EM-based anomaly detectors and monitors.
2) Demonstrate the feasibility of CodeBreach by a proof-of-concept.
3) Evaluate various malware code blocks with various payloads, achieving an average 40% reduction of detection possibilities on average, showing that CodeBreach is generic with different malware payloads.
4) Evaluate different test cases with various peak amplitudes, and determine the influence of amplitude on CodeBreach.

This is how the rest of the paper is structured. Chapter 2 is a comprehensive overview of CodeBreach, Chapter 3 provides an overview of the attack approach. Chapter 4 presents the technical details of CodeBreach. Chapter 5 provides the experiment details and the result of our experimental evaluation, and Chapter 6 is the conclusion of this paper.

# Chapter 2: Literature Review

## 2.1 Development of attack approach on embedded systems

Embedded systems are a kind of computer hardware system based on the microprocessor, they are designed to execute dedicated functions. Embedded systems are widely used as parts of larger systems or independent IoT systems [1]. IoT embedded systems are often connected to the internet, which exposes them to various security risks.[2] presents how to intercept communication of wireless devices and then receive the encryption key even at a long distance (40 miles), while [3] demonstrates the remote attack against SCADA devices via the Modbus protocol. The most famous defense approach is anti-viruses, a type of software that detects and removes harmful programs like viruses, worms, spyware, etc., by scanning the protected devices [4]. However, You & Yim found that attacks by malicious additions, removals, or tempers to the program are hard to detect by anti-viruses [5].

To solve this problem, dynamic detectors are designed to find suspicious activities during program execution. Since there are various complex program activities, it costs a vast amount of arithmetic power to build a model for normal execution. Unfortunately, limited by capacity and power consumption, dynamic detectors for PCs or other larger systems are unsuitable for embedded systems. A lightweight, exogenous detector is required for embedded systems.

Recent work has presented some anomaly detectors based on side channels. Most of them are based on leakage power signals [10], [11], while some focus on the field of EM emanations [7], [8], [15], [16]. Under the monitoring of the side-channel-based detector, traditional, arbitrary malicious attacks such as Shellcode or Ransomware generate obvious differences in the EM or power spectrum. Those malicious codes generate quite different features than normal program execution [15].

## 2.2 Development of EM-based detector

Electromagnetic (EM) emanations are generated by the varying current flow in electronic circuits within computing devices. Since the program activities result in the change of current flow, these EM emanations caused by current flow often carry some information related to the program activities in the system. Plenty of research in the EM emanations field focused on the risk of using EM emanations to obtain leaked system information, i.e., as a method for adversaries to collect sensitive data(such as AES decryption[6], [18], [19] or eavesdrop [17], [20]).

Recently, aside from collecting sensitive data values, many researchers have focused on the relationship between program activities and the corresponding EM malicious. In some of the work, they used EM emanations to detect malicious activities in program executions [7], [8], [15], [16] since EM emanations can be used to profile program activity[9], [12]. EM emanations can be used in both the time domain and frequency domain. The time-continue EM emanation signal is used as the record of program execution, and the various amplitudes of EM emanation can be identified as a feature of program execution [21]. Some other works use the discrete EM frequency spectrum in continuous time to analyze the features of program activities[7],[8],[22]. An observation found in [12] is that loop execution on an embedded system device tends to generate spikes in the EM frequency spectrum, which are related to its pre-iteration time(period). It means that the program activity can be efficiently profiled by observing the leaked EM frequency spectrum.

While [12] has shown that it is possible to use the EM spectrum to analyze which loop is executing in the current program, Nader[7] has presented an anomaly detector of program execution by monitoring the EM spectrum. In the training phase, the observed EM spectrum over time is gathered as the training data about the EM spectrum related to each part of program execution. During monitoring, it compares the observed spectrum with those received in the training phase and then reports any anomaly detected during monitoring.

## 2.3 Spikes on the EM spectrum

To demonstrate this observation in 2.2, Fig.1 displays a frequency spectrum of an AM-modulated loop activity in Raspberry Pi. As we can see, there are three peaks in Fig.1. The middle peak is at 700.005MHz, which is the processor clock frequency of Raspberry Pi. The two spikes on the left and right side of the clock frequency(699.986MHz and 700.014MHz) are generated by the loop activity in the *Basicmath* benchmark from the Mibench [13] suite. Inspired by this observation, we present a new method to cheat the state-of-the-art EM-based anomaly detector. We simulate the workload and structure of the loop to receive a similar pre-iteration time and then trick the anomaly detector using our falsified loop.

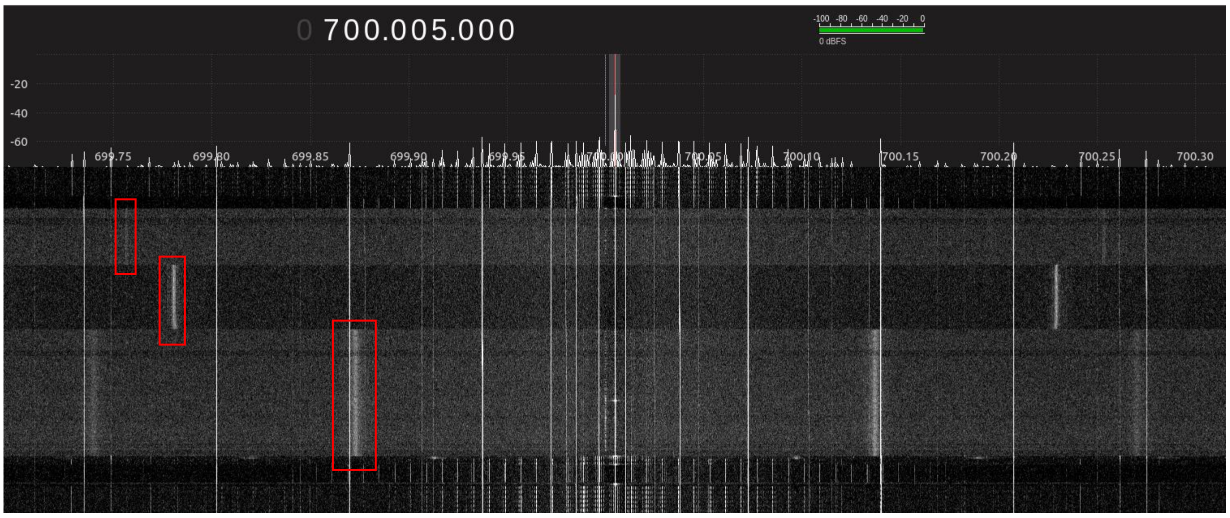


Figure 1: Spikes generated by Basicmath Benchmark

# Chapter 3 Threat Model

The overall idea of CodeBreach is to collect the spikes pattern of victim loop activity in the EM frequency spectrum, generate a reference library about where the normal spikes of the target loop should look like, and build an injection code block that generates spikes at the positions of the harmonic of victim loop based on target malware code, then inject it into the source code to attack the victim loop.

## 3.1 Attack Goal

In our attack approach, we aim to inject malware with various payloads into loop activities executing in embedded system devices that are under the protection of an EM-based frequency domain malware detector. The injected malware should cause improper operation, such as wrong program behaviors, of the target device and will be detected by malware detectors with a lower probability.

## 3.2 Attacker Capability

Our attack model assumes the presence of an adversary seeking to tamper with the loop concept of a program being monitored by an anomaly detector.

We consider a white-box attack setting where the adversary can access another device of the same type as the program executed. We further assume that the adversary has the ability to monitor the Em spectrum using an EM emanation receiver.

During the attack implementation phase, the adversary can gain and modify the source code of the whole program, and the adversary can test the tampered program an unlimited number of times.

## 3.3 Target hardware

The hardware we target is embedded system boards. The representative one is Raspberry Pi, which has an ARM processor clocked at around 700MHz. During our experiments, we positioned a magnetic probe close to the processor but without physical contact with the board.

# Chapter 4: Design of CodeBreach

This section describes our technical details. In Chapter 4.1, we describe our attack's target detection algorithm, followed by an explanation of our attack design in Chapter 4.2.

## 4.2: Detection Algorithm

In this section, we will introduce the state-of-the-art EM-based anomaly detector (EDDIE), which is also CodeBreach's target victim detector. We first describe the signal profiling procedure for collected signals. Following this, we show the detector's algorithm, including the use of the K-S test on Spike frequencies to identify anomalous activity and the checking logic from spectrum to loops.

### 4.2.1: Signal Profiling

There are three main procedures in signal profiling:

- find the possible signal sequences in which the loop may execute
- collect enough reference data that can cover different situations of loop execution
- identify the spikes that may generated by loop activities in the spectrum

The possible sequence analysis starts from the traditional control flow graph (CFG) of the program. EDDIE then merges all the nodes within a CFG that belong to each loop nest into separate loop-region nodes. In our paper, a program can be divided into multiple loop nests, and the separate loop nests are the target of injection attacks. To identify the accurate pre-iteration execution time, time instrumentation is added before and after each loop nest. In each run of reference sets, the detector collects the signal with its entry and exit times recorded by those instrumentations.

The program will be executed multiple times to enhance the coverage of loop execution since the EM emanation of the same loop is changed by the various program inputs. During the profiling of the Basicmath benchmark, the reference set consists of 20 runs. For each loop, The signals collected will be divided into multiple Short-Term Spectra (STSs) by sliding window. As demonstrated in Chapter 2.3, the detector can identify the spikes generated by loop activates for each STS in each loop. Afterward, we gather

STSs from each training dataset. To ensure consistency across all reference sets, we standardized the length of the shortest signal in the 20 runs.

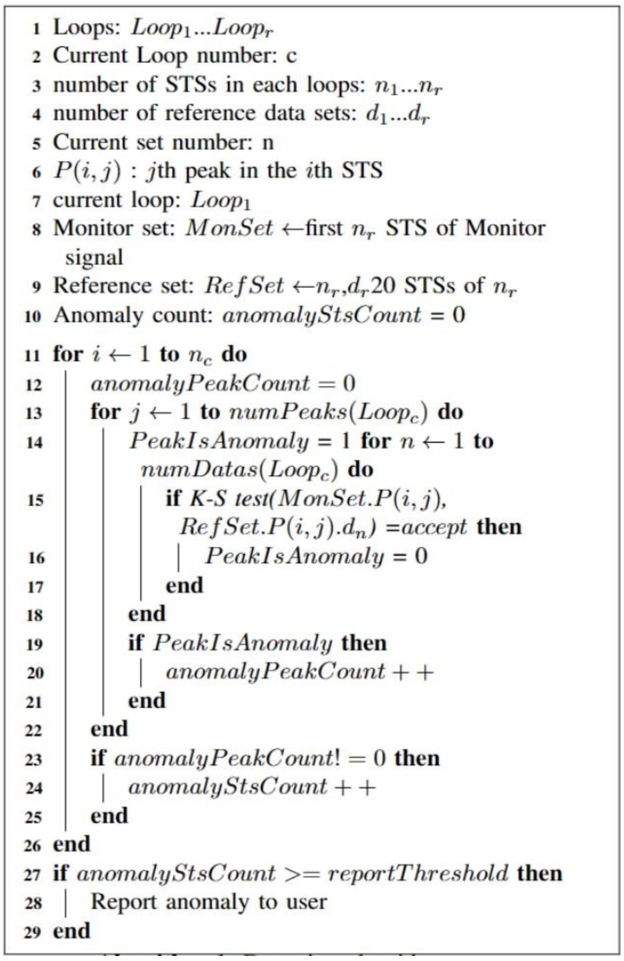
Finally, a reference set including STSs with identified spikes for each loop in the program will be established for anomaly detection. Then, the statistical test described in Chapter 4.2.2 will assist in identifying anomaly spikes.

## 4.2.2: Statistical Test

The critical idea of anomaly detection is to determine whether the two activities are equivalent. Obviously, it cannot be based on human observation or empirical judgment. EDDIE's decision-making is based on statistical tests[7]. A *statistical test* is a tool used in hypothesis testing to determine whether a predictor variable has a statistically significant relationship with an outcome variable, or to estimate the difference between two or more groups. The detector selects *the Kolmogorov-Smirnov* (K-S) test [14], one of the best-known non-parametric test methods.

## 4.2.3: Checking Logic

Figure 2 illustrates how the anomaly detector checks through spectrum and loops and decides when to report the anomaly. In each loop, the K-S test is applied to every peak of the STS being examined. For the position of each peak. Eddie will compare the monitor set with all reference sets, and an anomaly will only be counted if none of the reference sets have the same distribution as the monitor set (refer to Line 15). Returning to the spectrum level, an STS will be considered an anomaly if any peak is unmatched. When all STSs are checked, EDDIE checks if the counted anomalies exceed



```
1  Loops: $Loop_1 ... Loop_r$
2  Current Loop number: c
3  number of STSs in each loops: $n_1 ... n_r$
4  number of reference data sets: $d_1 ... d_r$
5  Current set number: n
6  $P(i, j)$ : $j$th peak in the $i$th STS
7  current loop: $Loop_1$
8  Monitor set: $MonSet \leftarrow$first $n_r$ STS of Monitor
     signal
9  Reference set: $RefSet \leftarrow n_r, d_r$20 STSs of $n_r$
10 Anomaly count: $anomalyStsCount = 0$
11 for $i \leftarrow 1$ to $n_c$ do
12     $anomalyPeakCount = 0$
13     for $j \leftarrow 1$ to $numPeaks(Loop_c)$ do
14         $PeakIsAnomaly = 1$ for $n \leftarrow 1$ to
             $numDatas(Loop_c)$ do
15             if K-S test($MonSet.P(i, j)$,
                 $RefSet.P(i, j).d_n$) =accept then
16                 $PeakIsAnomaly = 0$
17             end
18         end
19         if $PeakIsAnomaly$ then
20             $anomalyPeakCount++$
21         end
22     end
23     if $anomalyPeakCount! = 0$ then
24         $anomalyStsCount++$
25     end
26 end
27 if $anomalyStsCount >= reportThreshold$ then
28     Report anomaly to user
29 end
```

Figure 2: Detection Algorithm

the threshold. Setting the threshold also impacts the True Negative (TN) and False Positive (FP) of detection result.

## 4.3 Attack Design

As illustrated in Fig.3, we divide the workflow of our CodeBreach method into three main aspects: Normal signal library creation, injection code build and Injection. We present each part in details in the following sections.

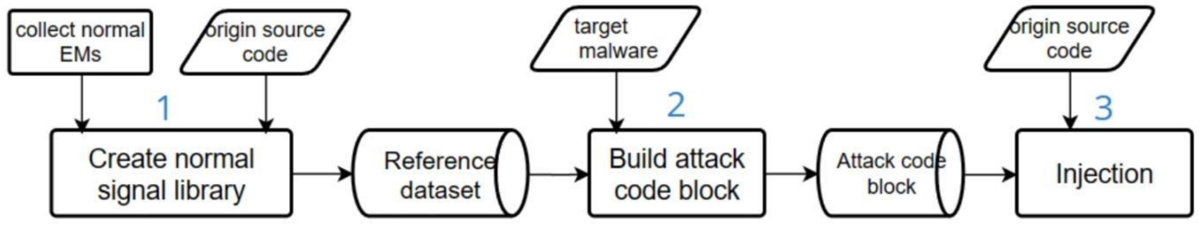


Figure 3. framework of the workflow of CodeBreach

### 4.3.1 Create normal Signal Library

In order to generate a malicious code block in a simulation, CodeBreach has to collect enough signals corresponding to the target loop activity.

As mentioned in Chapter 4.2.1, we add some instrumentation before and behind the target loop code nest to calculate the loop period. This instrumentation allows us to locate the accuracy signal section corresponding to the target loop activity from the origin signal collected by SDR. The instrumentation only contains the time information, which is lightweight. At the same time, the instrumentation is added outside the loop nests; hence, the instrumentation would not affect the normal activity of the target loop. After trying several methods, we select the Unix timestamp as the instrumentation, which can fulfill the requirement of being accurate and lightweight.

Then codeBreach runs the target program multiple times to improve the coverage of the loop activity, each time with different inputs. Different inputs may have different EM patterns. Meanwhile, we use the shortest signal length as a representative signal length of this loop, and then all those signals are trimmed to the same length for comparison.

These signals are grouped as a signal library, which stands for a loop activity. In our experiment, the signal library of a victim loop consists of 20 runs, with each signal having the same length.

The signals collected are then divided into multiple sample windows by a sliding window. With the constant windows size and slide step, all the signals from the same loop have the exact count of sample windows. For each window, CodeBreach uses the short-term Fourier Transform to convert an EM signal segment into a frequency spectrum. As mentioned in Chapter 2.3, loop activities always generate spikes related to their period in the EM frequency spectrum. Thus, We can find some spikes in the spectrum, which include the origin loop spike and its harmonics. In order to locate the origin loop spike, CodeBreach calculates the loop frequency with the operation time provided by instrumentation.

At the end of data processing, CodeBreach stores all those spikes information in the execution order into a reference dataset as the standard signal library..

### 4.3.2 Build Injection Code

Based on the observation in Chapter 2.3, loop execution on an embedded system device always appears on the primary frequency and its harmonic. The critical aspect of CodeBreach's deception is to inject a code block with an appropriate length, which leads to the harmonic spike generated by the loop after injection appearing at the same position as the spike generated by the standard loop.

Because the spikes generated by loop activities are related to its pre-iteration period, the problem transformed from building a code block with uncountable "payload" to a code block with countable execution "time", which can be recorded by instrumentation. This means if the pre-iteration time of a loop is $T$, we can find spikes on the corresponding spectrum at frequencies around $f = \frac{1}{T}$, and we need to build an injection code block whose pre-iteration time is $nT$. After injection, the loop will generate spikes at $f = \frac{1,2,\ldots,n+1}{T}$, and its $n + 1$ times harmonic at the same frequency as the peak of normal loop activity.

For a victim loop with a period $t$, if we want to inject a malware code block whose pre-iteration time is $T, nt < T < (n + 1)t$, CodeBreach will add some idle code, e.g. power operation, to build the injection code block. CodeBreach can modify the parameter setting of added code, to make the period of adjusted injection code block $T' = (n + 1)t$.

### 4.3.3 Injection

In order to inject the injection code block into the target loop, we have to modify the source code and then reload it into the target device. Thus, the modification right is pivotal in our attack assumption. It is essential to point out that our attack can only protect the injection code with constant execution time from the malware detector, and that is why we would not compare CodeBreach with other code injection attack methods, e.g., the Shellcode attack. Further implementation details can be found in Chapter 5.3.

# Chapter 5: Experiment and Result

In this section, we first display our experiment setup (Chapter 5.1), then we display the experiment result of the feasibility test (Chapter 5.2), followed by the expected results of attacking EDDIE on Raspberry Pi (Chapter 5.3).

## 5.1 Experiment Setup

We use Raspberry Pi model Zero W Rev 1.1 as the IoT device. It is a single-board computer with an ARM 1176JZF-S processor and a Debian 10 Linux operation system. We collect the emanated EM signal with a small magnetic probe close to the device's processor, and the signal is recorded using a bladeRF 2.0 micro XA9 SDR. The setups ensure that the target detector can work efficiently.

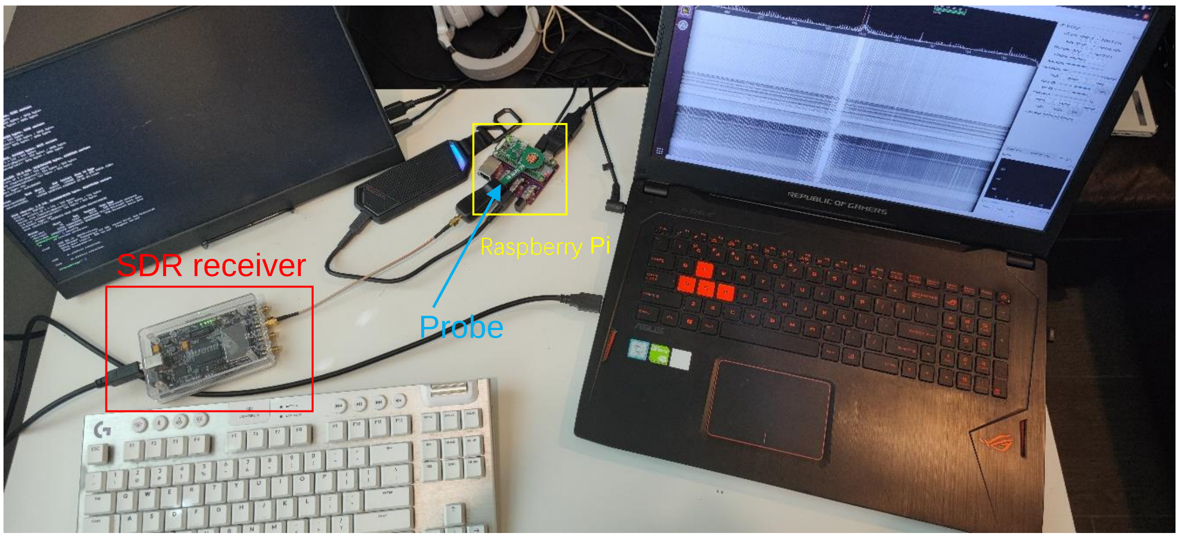


Figure 4: Experiment Setup

In our experiments, we use five kinds of benchmarks from the MiBench [13] suite to attack the EDDIE [7] algorithm. We execute each benchmark 20 times to build our reference library. We also use Unix timestamp as the instrumentation. Since it is important to keep time sync between the recording device and the IoT device as the Unix timestamp is used to record the accurate time information of loop execution, we sync the time of Raspberry Pi and PC by the same source before each signal recording.

We execute the injection test sets five times each and use the average result as the detection result. In order to demonstrate the effect of CodeBreach, we define the likelihood of an injection block being detected equal to the number of STSs marked as anomaly divided by the number of all STSs. With the new standard, the impact of our approach can be observed more obviously. In our measurement, we use a 5 ms window with a 1.25 ms slip step. The parameter setting makes a trade-off between computation efficiency and frequency resolution with a 10MHz sampling rate.

## 5.2 Attack Feasibility

The central concept of our attack is to simulate the spikes generated by the origin loop activity using the harmonic of the injected loop. In our attack on Basicmath, we concentrate on the first loop in the source code, which aims to solve the cubic equation problem. With the aid of instrumentation, we obtain the per-iteration execution time T.

Fig.4 shows the spectrum of the details of our attack. Loop 1 is the origin loop of solving the cubic equation problem. The primary and harmonic signals can be found in 218KHz and 436KHz on the spectrum. By adding some adversary code into Loop 1, we

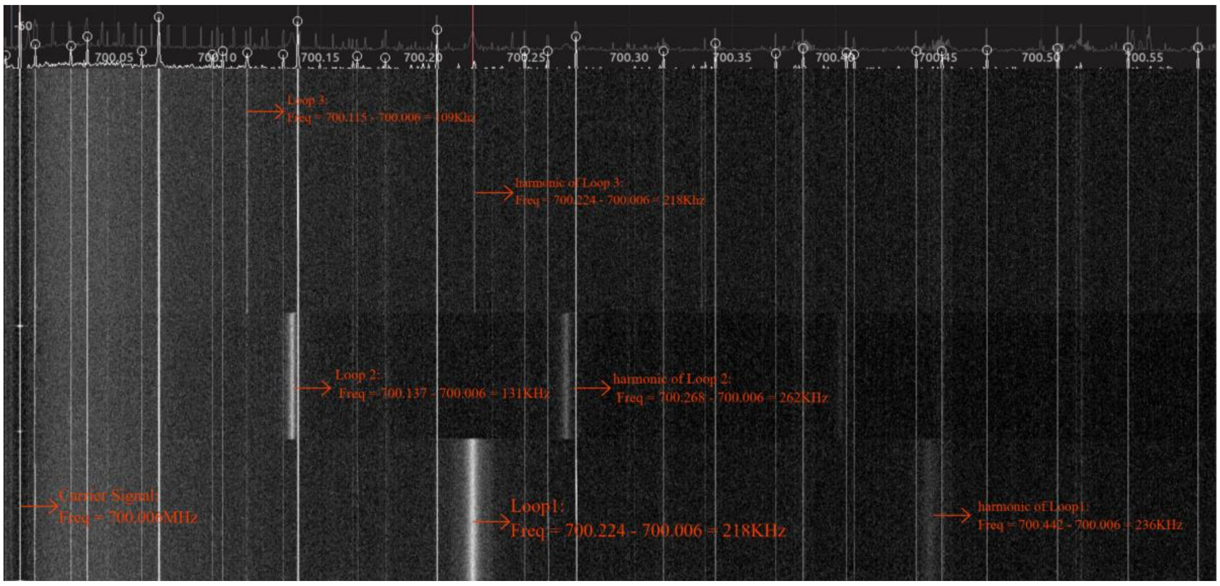


Figure 5: EM Spectrum of Feasibility Test

can get Loop 2, and the generated spike obtains a lower frequency than Loop 1's because of the longer loop period. In Loop 3, we add some filler into Loop 2, making its total period twice that of Loop 1. As we observed in Fig.4, the harmonic of Loop 3 matches the frequency of the primary spike of Loop 1.Based on the observation that the amplitude of the harmonic of Loop 3 is significantly lower than the basic spike of Loop 1, we also find that the amplitude can affect the attack result. Further implementation details will be presented in Chapter 5.5.

| dataset | Loop Num. | period | Spike frequency |
|---|---|---|---|
| Reference set | Loop 1 | 4.66$\mu s$ | 218KHz |
| Reference set | Loop 2 | 7.6996$\mu s$ | 130KHz |
| Attack set | Loop 1 | 15.4$\mu s$ | 66Khz |
| Attack set | Loop 2 | 9.17$\mu s$ | 109KHz |

Table 1: Periods and Frequencies of Loops in Feasibility test

## 5.3 Attack Results For Different Benchmarks

In this section, we inject the origin and adjusted code blocks by CodeBreach into the same region of each application. The injection clock contains a memory allocation, a memory write, and a memory delete, adding $2 \sim 5\ \mu s$ to each loop execution time. We tested five representative programs [15] from the Mibench [13] suite, including *Basicmath, Bitcount, Susan, Qsort*, and *FFT*. We have 20 reference sets to collect signal data from standard loop execution. To ensure the effectiveness of the EDDIE detection algorithm, there are also two validation sets without malware injected. We execute the injected programs five times each, and we use the average result of the five times as the representative result.

| Benchmarks | False positive (%) | Origin block (%) | Adjusted block (%) |
|---|---|---|---|
| Baiscmath | 8.47 | 89.9264 | 42.5876 |
| Bitcount | 5.33 | 94.3277 | 37.9672 |
| Susan | 6.72 | 79.5902 | 44.2450 |
| Qsort | 1.68 | 0.039 | 0.2173 |
| FFT | 5.87 | 82.7912 | 32.8122 |

Table 2: Result of Attack on EDDIE Detector

Table 2 shows the result of the detection and Attack of Raspberry Pi. The first column displays the programs, and the following columns show the false positive rate and the possibilities of injected loops being detected before and after CodeBreach. False positives are the percentage of those STSs reported as anomalous but do not contain an injection block. The average false positive is around 5%, which is low enough to prove the reliability of the EDDIE detector. The possibility indicates the ability of an injected malware to trick the detector, including the origin injection block and the injection block modified by CodeBreach. The possibility of being detected is the percentage of the number of STSs detected as anomalous within the number of STSs of the loop injected. Based on the result, the EDDIE algorithm can detect anomalous spectrums efficiently (86.659%). However, the *Qsort* is an expectation. According to [12], the *Qsort* program does not generate a stable EM leakage. The loop activities in std::qsort library are various, and no significant peak activity exists on the EM spectrum. Thus, the EDDIE detector cannot distinguish the normal STSs and anomalous STSs of *Qsort*cannot. We also minimize the influence of borders between loops by using the Unix timestamp as the instruction. The instructions with high accuracy can enforce the detection effects. After the modification by CodeBreach, the average possibility of being detected has decreased to a significantly low level, which is only 39.402%. Comparing the results with and without CodeBreach, we can find that EDDIE has the ability to detect the anomaly loop activities and identify the correct loop execution. However, our attack method can also trick the anomaly detector effectively.

## 5.4 Attack Results With Different Payloads

To convince that CodeBreach is general in different kinds of injection malware and to gain more knowledge about the scale of our approach, in this section, we inject several kinds of code blocks with various payloads into the same application. The first injection block contains a series of memory operations, including allocate, assign, and delete. The memory block will increase around $3\mu s$ execution time for each loop. The second block calls some functions executed in other loops from the header file. The loop period will increase 2 to $5\mu s$ depending on the functions. We also test the code block with a long execution time to test the multiple harmonics. We define a “long” injection block with at least 1 time of the original loop period. Except for the three malware that result in the change of loop period, we also test the Shellcode attack to clarify the scale of CodeBreach. All those blocks are injected into the same loop of *Basicmath*, and each executes five times.

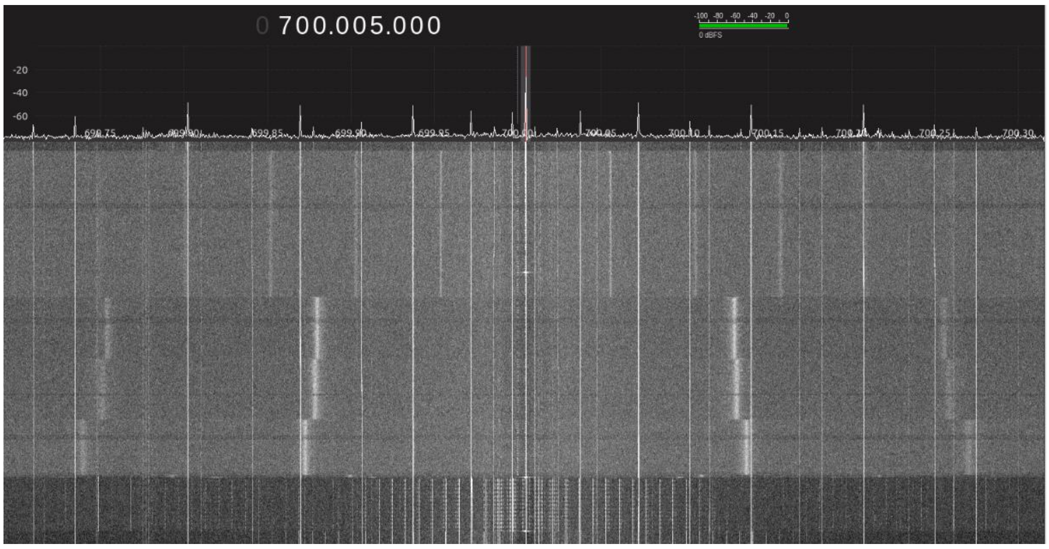


Figure 6: Spikes before CodeBreach

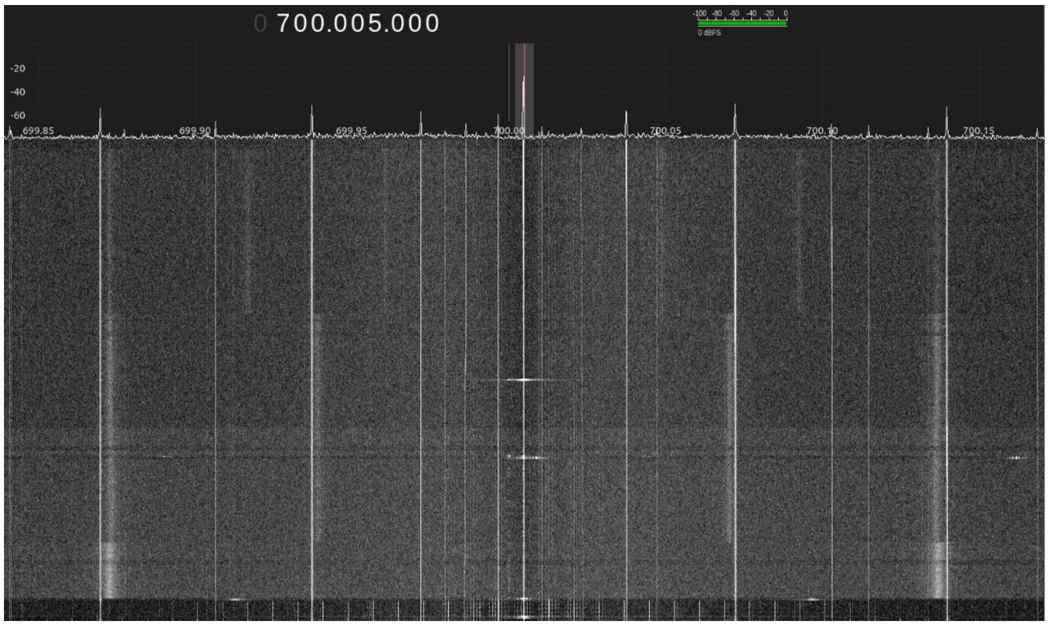


Figure 7: Spike After CodeBreach

Figure 6 displays the EM spectrum of the result of arbitrary injection without CodeBreach; there are four pairs of spikes generated by standard loop activity: the loop with memory modification block, the loop with malicious function block, and the loop with long period block. The amplitude and frequency of those spikes are significantly different. Meanwhile, Figure 7 shows the EM spectrum after the modification by CodeBreach. Those spikes with different injection payloads have been aligned with the normal one, and those spikes are used to challenge the EM-based anomaly detector.

Table 3 displays the results of different block sizes. EDDIE can detect all those malware with a high possibility (more than 85%). The Shellcode attack will create a new terminal, and the victim loop has to wait until the new terminal is closed. The highest detection possibility of the Shellcode attack may result from the apparent variation in the EM spectrum due to the program status change. Meanwhile, we observed that CodeBreach is only available for those malware code blocks with constant execution time. Since the peaks we observed on the EM spectrum are generated by loop activities, the peaks disappear due to malware that may change the program logic or have various execution times. Thus, our approach cannot handle such a situation as a Shellcode attack. CodeBreach efficiently disturbs the detection ability with a significant reduction (around 40%) in the possibilities for memory modification block and malicious functions block. However, the detection possibility of the long-period block only decreases by 12%. We also perform experiments on different harmonic multiples to determine the reason.

| Block types | Origin block (%) | Adjusted block (%) |
|---|---|---|
| Memory Modification | 89.9264 | 42.5876 |
| Malicious Function | 91.4892 | 51.73 |
| Long period block | 85.48 | 73.4074 |
| Shellcode opeartion | 100.00 | - |

Table 3: Result of Different Payloads

## 5.5 Effect of Different Multiple of Harmonic

As mentioned earlier, CodeBreach decorates the multiple harmonics of the injected loop to pretend the normal spike. Theoretically, any harmonic spike with a longer per-iteration period can be decorated to the fake spike. However, CodeBreach always chooses the harmonic spike with minimal multiple since it has the highest amplitude (Figure 1, 5). In order to show how the amplitude of harmonics can affect disturbance efficiency, we compare the attack results based on different multiple harmonics. We set the memory modify block as the target malware and the first loop of *Basicmath* as our target loop. We use double, triple, and quadruple harmonic to simulate the target spike separately. According to the result in Table 4, we can clearly see that the disturbance efficiency of CodeBreach decreases with multiple increases. The observation can also explain why the test case with a long period block reaches a worse result in Chapter 5.4.

| Multiples of harmonic | double | triple | quadruple |
|---|---|---|---|
| Detection possibility(%) | 42.59 | 74.06 | 92.30 |

Table 4: Result of Multples Harmonic

# Chapter 6: Conclusion & Future Work

## 6.1 Conclusion

This paper describes CodeBreach, a new attack approach to inject malicious code blocks with different payloads into embedded devices under the protection of an EM-based frequency domain anomaly detector. Different from the arbitrary injection methods, CodeBreach analyzes the target program by receiving electromagnetic (EM) emanation generated by the loop activities in the embedded device. Then, CodeBreach decorates the malware block to pretend to be the standard loop activity.

During analysis, CodeBreach collects the EM signals of normal loop activities to create the standard signal library, and the period and peak information generated by origin loop activities is stored to modify the target malware code block later. During the attack, to align the peaks generated by normal loop activities and the harmonic of loop activities after injection, CodeBreach builds a new injection block with an appropriate execution time based on the information stored in the standard signal library. Since the CodeBreach decorates an injection code block to a new block with an appropriate length, it leads to the harmonic spike generated by the injected loop at the same position as peaks generated by the standard loop.

We evaluate CodeBreach on Raspberry Pi with different kinds of malware with different payloads and find that CodeBreach can efficiently disturb the EM-based anomalous detector. We also find that CodeBreach is general to different malware with constant execution time. Furthermore, we point out that the amplitude of the harmonic peak significantly influences the attack efficiency.

## 6.2 Future Work

In the future, there is some further work can be done to improve the performance of CodeBreach based on our research:

1. Limited by the computation ability and storage capacity, We cannot improve the resolution of our signals, which has an effect on our precious analysis. Because of the limitation of signal resolution, there are some biases when we

are doing the alignment. A higher resolution can improve the performance of CodeBreach.

2. Since EDDIE did not provide its source code, we have to refactor it based on the algorithm in the paper. Unfortunately, there is an apparent gap between our result and the result in the EDDIE paper.
3. CodeBreach is an attack design with several steps on different devices. During the evaluation, we must manually record the signal, timestamp, and result. If we want to transfer CodeBreach from a theory to a comprehensive framework, we have to increase its automation.

# Reference

[1] Jack Ganssle and Michael Barr. 2003. Embedded Systems Dictionary. CMP Books.

[2] L.ApaandC.M.Penagos, “Compromising Industrial Facilities from 40 Miles Away,” ser.BlackHat,2013.

[3] E.FornerandB.Meixell,Out of Control:SCADA Device Exploitation, Cimation, 2013. [Online]. Available: https://media.blackhat.com/us-13/US-13-Forner-Out-of-Control-Demonstrating-SCADA-WP.pdf

[4] G. McGraw and G. Morrisett, "Attacking Malicious Code: A Report to the Infosec Research Council," in IEEE Software, vol. 17, no. 5, pp. 33-41, Sept.-Oct. 2000, doi: 10.1109/52.877857.

[5] I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, Fukuoka, Japan, 2010, pp. 297-300, doi: 10.1109/BWCCA.2010.85.

[6] Ruize Wang, Huanyu Wang, and Elena Dubrova. 2020. Far Field EM Side-Channel Attack on AES Using Deep Learning. In Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security (ASHES'20). Association for Computing Machinery, New York, NY, USA, 35–44. https://doi.org/10.1145/3411504.3421214

[7] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic and M. Prvulovic, "EDDIE: EM-based detection of deviations in program execution," 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, ON, Canada, 2017, pp. 333-346, doi: 10.1145/3079856.3080223.

[8] Yi Han, Sriharsha Etigowni, Hua Liu, Saman Zonouz, and Athina Petropulu. 2017. Watch Me, but Don't Touch Me! Contactless Control Flow Monitoring via Electromagnetic Emanations. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). Association for Computing Machinery, New York, NY, USA, 1095–1108. https://doi.org/10.1145/3133956.3134081

[9] Robert Callan, Farnaz Behrang, Alenka Zajic, Milos Prvulovic, and Alessandro Orso. 2016. Zero-overhead profiling via EM emanations. In Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016). Association for Computing Machinery, New York, NY, USA, 401–412. https://doi.org/10.1145/2931037.2931065

[10] François Durvaux and Marc Durvaux. 2020. SCA-Pitaya: A Practical and Affordable Side-Channel Attack Setup for Power Leakage--Based Evaluations. Digital Threats 1, 1, Article 3 (March 2020), 16 pages. https://doi.org/10.1145/3371393

[11] Han, Y., Chan, M., Aref, Z., Tippenhauer, N., & Zonouz, S. (2022). Hiding in Plain Sight? On the Efficacy of Power Side Channel-Based Control Flow Monitoring. *USENIX Security Symposium*.

[12] N. Sehatbakhsh, A. Nazari, A. Zajic and M. Prvulovic, "Spectral profiling: Observer-effect-free profiling by monitoring EM emanations," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, 2016, pp. 1-11, doi: 10.1109/MICRO.2016.7783762.

[13] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538), Austin, TX, USA, 2001, pp. 3-14, doi: 10.1109/WWC.2001.990739.

[14]  Kolmogorov–Smirnov Test. 2008 In: The Concise Encyclopedia of Statistics. Springer, New York, NY. https://doi.org/10.1007/978-0-387-32833-1-214.

[15] Nader Sehatbakhsh, Alireza Nazari, Monjur Alam, Frank Werner, Yuanda Zhu, Alenka Zajic, and Milos Prvulovic(2019). REMOTE: Robust External Malware Detection Framework by Using Electromagnetic Signals. IEEE Transactions on Computers. PP. 1-1. 10.1109/TC.2019.2945767.

[16] B. B. Yilmaz et al., "MarCNNet: A Markovian Convolutional Neural Network for Malware Detection and Monitoring Multi-Core Systems," in IEEE Transactions on Computers, vol. 72, no. 4, pp. 1122-1135, 1 April 2023, doi: 10.1109/TC.2022.3184520.

[17] Takatoi, G., Sugawara, T., Sakiyama, K., & Li, Y. (2020). Simple Electromagnetic Analysis Against Activation Functions of Deep Neural Networks. *ACNS Workshops*.

[18] X. Cui, H. Zhang and L. Wang, "Research on AES Cryptographic Chip Electromagnetic Attack Based on Deep Transfer Learning," *2019 IEEE 6th International Symposium on Electromagnetic Compatibility (ISEMC)*, Nanjing, China, 2019, pp. 1-4, doi: 10.1109/ISEMC48616.2019.8986117.

[19] Alam, M., Khan, H.A., Dey, M., Sinha, N., Callan, R.L., Zajić, A.G., & Prvulović, M. (2018). One&Done: A Single-Decryption EM-Based Attack on OpenSSL's Constant-Time Blinded RSA. *USENIX Security Symposium*.

[20] Z. Zhan, Z. Zhang, S. Liang, F. Yao and X. Koutsoukos, "Graphics Peeping Unit: Exploiting EM Side-Channel Information of GPUs to Eavesdrop on Your Neighbors," *2022 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2022, pp. 1440-1457, doi: 10.1109/SP46214.2022.9833773.

[21] Batina, L., Bhasin, S., Jap, D., & Picek, S. (2019). CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel. *USENIX Security Symposium*.

[22] Z. Zhang, Z. Zhan, D. Balasubramanian, B. Li, P. Volgyesi and X. Koutsoukos, "Leveraging EM Side-Channel Information to Detect Rowhammer Attacks," *2020 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2020, pp. 729-746, doi: 10.1109/SP40000.2020.00060.

# Appendix

## List of Figures:

## List of Tables:

## Mouthly Log: