CASPER: An efficient approach to detect anomalous code execution from unintended electronic device emissions

Hira Agrawal[a], Ray Chen[b], Jeffrey K. Hollingsworth[b], Christine Hung[a], Rauf Izmailov[a], John Koshy[a], Joe Liberti[a], Chris Mesterharm[a], Josh Morman[a], Thimios Panagos[a], Marc Pucci[a], Işil Sebüktekin[a], Scott Alexander[a], and Simon Tsang[a]

[a]Vencore Labs, 150 Mt Airy Rd, Basking Ridge, NJ, US
[b]University of Maryland, College Park, MD, US

ABSTRACT

The CASPER system offers a lightweight, multi-disciplinary approach to detect the execution of anomalous code by monitoring the unintended electronic device emissions. Using commodity hardware and a combination of novel signal processing, machine learning, and program analysis techniques, we have demonstrated the ability to detect unknown code running on a device placed 12" from the CASPER system by analyzing the devices RF emissions.

Our innovations for the sensors subsystem include multi-antenna processing algorithms which allow us to extend range and extract signal features in the presence of background noise and interference encountered in realistic training and monitoring environments. In addition, robust feature estimation methods have been developed that allow detection of device operating conditions in the presence of varying clock frequency and other aspects that may change from device to device or from training to monitoring. Furthermore, a band-scan technique has been implemented to automatically identify suitable frequency bands for monitoring based on a set of metrics including received power, expected spectral feature content (based on loop length and clock frequency), kurtosis, and mode clustering.

CASPER also includes an auto-labeling feature that is used to discover the signal processing features that provide the greatest information for detection without human intervention. The system additionally includes a framework for anomaly detection engines, currently populated with three engines based on n-grams, statistical frequency, and control flow. As we will describe, the combination of these engines reduces the ways in which an attacker can adapt in an attempt to hide from CASPER.

We will describe the CASPER concept, components and technologies used, a summary of results to-date, and plans for further development. CASPER is an ongoing research project funded under the DARPA LADS program.

1. INTRODUCTION

The DARPA Leveraging the Analog Domain for Security (LADS)[1] program started in 2016 based on results that examined the internal state of computers using unintended emissions. Those prior efforts[2–5] looked primarily into ways by which cryptographic keys could be extracted from a computer performing decryption of a known text. The side channels exploited included RF, power, and acoustic.

The LADS program moves from the offensive to the defensive. It asks whether it is possible to extend the techniques in the prior work in order to learn the normal behavior of a computing system and then to monitor that system for deviations from the normal behavior. Thus, for example, as Internet of Things (IoT) devices become more common, many are easily attacked.[6] Moreover, in many cases, IoT devices are quite simple reducing the ability to additionally deploy traditional security infrastructure. However, by using a system that collects unintended emissions from the device, it becomes possible to monitor the device in a very stealthy way and in a way that is very hard for an attacker to subvert.

In the initial phase of LADS (completed at the end of last year), a key program goal was to be able to determine if a IoT-class device is running known or unknown code. This paper focuses on that challenge.

Further author information: Send correspondence to Scott Alexander: E-mail: salexander@vencorelabs.com

2. CASPER OVERVIEW

The CASPER (Combined Adaptive Sensing for Pervasive Event Reconstruction) system offers a lightweight, multi-disciplinary approach to detect the execution of anomalous code by monitoring the unintended electronic device emissions at a distance. Using commodity hardware and a combination of signal processing, machine learning, and program analysis techniques, we have demonstrated the ability to detect unknown code running on a device placed 12" from the CASPER system by analyzing the devices RF emissions.

CASPER addresses several key technical challenges: (1) determining the analog channels that provide sufficient information to determine the internal state of the device, (2) mapping the internal state of the device to analog emissions that can be used for detecting attacks, (3) tracking these analog emissions with sufficient fidelity while investigating the effects of environmental considerations at different distances, and (4) balancing access to the device and processing costs with the ability to capture sufficient information about the state of the device to accurately detect attacks.

CASPER employs a new approach to involuntary analog emission monitoring using a cross-layer combination (see Figure 1) of advanced signal processing techniques implemented in a subsystem called Multi-modal Adaptive Sensors (MAS), machine learning techniques implemented in a subsystem called Emanation Model Analytics (EMA), program analysis techniques implemented in a subsystem called Processor & Program Analysis (PPA), and multi-pronged model divergence detection techniques implemented in a subsystem called Monitoring Anomaly Detectors (MAD). These subsystems work closely together in an iterative process to create a detailed model of normal device behavior to detect attacks.

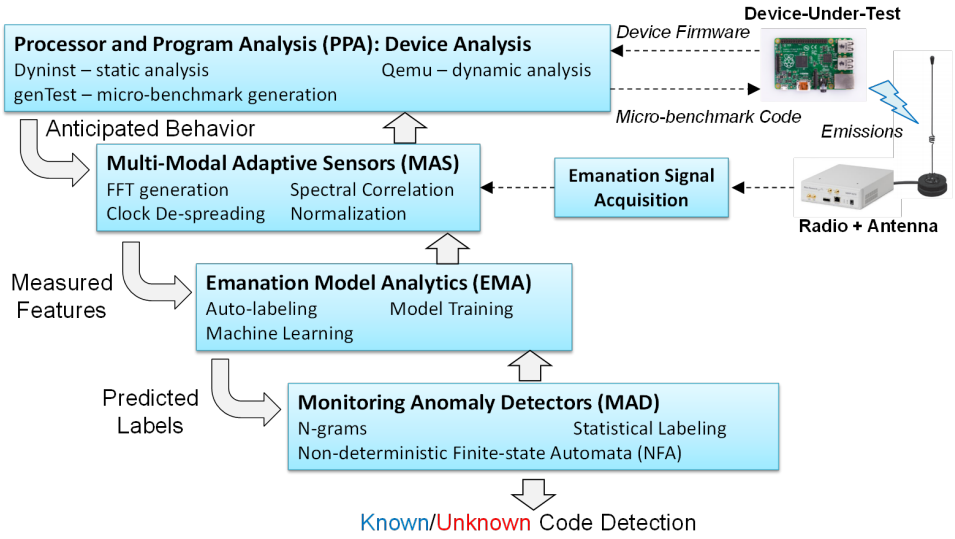

Figure 1. The CASPER Architecture

Briefly, MAS is responsible for interfacing with the sensors and bringing signals into the digital domain. In phase 1 of LADS, this has entailed using Gnu Radio[7] with a Universal Software Radio Peripheral (USRP)[8] to collect RF emanations. Algorithms, described in more detail in Section 3, are used to find faint signals, remove sources of error such as drift, and ultimately provide FFT bins for the rest of the system. EMA (Section 4) uses machine learning techniques in order to label the data, create models of normal behavior, and then match emanations to labels during the monitoring phase. MAD (Section 6) currently uses three approaches, statistical, n-gram, and non-deterministic finite-state automata (NFA) to look at the emanations in different ways to determine whether divergence from normal behavior has occurred. PPA 5 is responsible for providing analysis to assist the other subsystems. It generally collects and applies information about the particular processor architecture and performs program analysis in order to generate micro-benchmarks and other tools to aid the rest of the system.

2.1 CASPER Phases

Conceptually, CASPER operation is broken into three distinct phases: Device Characterization, Training (also called Normal Behavior Characterization), and Monitoring. The phases and the role of the CASPER subsystems is illustrated in Table 1 and described below. The phases are iterative. Over the course of device under test (DUT) operation we anticipate there will be a need to periodically jump back to earlier phases as the DUT is updated or the background environment changes.

Table 1. CASPER phases and subsystems

CASPER Phase	PPA Subsystem	MAS Subsystem	EMA Subsystem	MAD Subsystem
Device Characterization	genTest Static loop analysis Dynamic analysis	Band Scan Feature generation Normalization Array processing Spectral correlation Clock de-spreading	Cluster analysis	
Training	Static loop analysis	Feature generation Normalization Array processing Spectral correlation Clock de-spreading	Autolabeling Model training Machine learning	N-gram training Statistical training NFA training
Monitoring		Feature generation Normalization Array processing Spectral correlation Clock de-spreading	Classification	N-gram monitoring Statistical monitoring NFA monitoring

2.1.1 Device Characterization

The goal of this phase determine the key characteristics of the DUT so that CASPER can best capture its emanations and learn its operating behavior. The MAS Band Scan tool is applied to determine the radio frequencies at which the device exhibits emanations distinct from the background environment. This provides a useful set of frequencies to explore for further characterization or to begin the Training phase. With the DUT running, various MAS techniques such as array processing, normalization, spectral correlation, clock de-spreading are applied to determine which combination techniques extracts the best features from the captured emanation signals. EMA Clustering techniques are used to determine if the resulting emanations generate useful, distinguishable clusters, thus developing an understanding of what code generates distinguishable emanations.

If we have access to the devices firmware, the PPA genTest tool creates several micro-benchmarks (small programs consisting of segments of processor instructions or known code) with associated identifying labels for which the device generates significant analog emanations to differentiate among them. The PPA Dyninst tool may also insert additional code into the binaries ("boosting") to generate more identifiable emanations in important parts of the program in order to make it easier to separate different portions of the software. Program loops are known to produce well-defined device emanations in the radio-frequency domain. The PPA Dyninst tool applies static loop analysis to determine the devices loop characteristics and thus predict where emanations are likely to occur. The loop analysis also provides information about the DUTs operation that will help the rest of the CASPER system determine its operating parameters such as: MAS capture frequency, sampling rate, FFT size, EMA training times, clustering parameters, and MAD window lengths.

2.1.2 Training (Normal Behavior Characterization)

The goal of this phase is to learn the DUTs normal behavior. The DUT is exercised over a period of time using the MAS capture parameters and techniques determined during the device characterization phase and generates features (e.g. FFT vectors) that represent the emanations. EMA autolabeling generates labels from these MAS

features and EMA machine learning to generate a model of labeled emanation patterns. Based on the EMA-generated sequences of labels, MAD trains the various detectors (n-Gram, statistical, NFA). The MAD models provide the timing and sequence characteristics about the DUTs normal behavior. PPAs static loop analysis provides information such as average loop sizes, and number of distinct code branches that can be used by EMA and MAD to determine how much training is needed and set windowing parameters.

2.1.3 Monitoring

The goal of this phase is to continuously monitor the DUT and determine if the DUTs emanations, timing or sequence characteristics deviate from the models learned during the training phase. MAS applies the same capture parameters and techniques used during training to generate features for EMA to evaluate. EMA classification compares the emanation feature against the model developed during training. If the feature does not fit the model, EMA generates an "unknown" label, otherwise EMA generates the previously-learned label. The EMA-generated labels are then passed to MAD which determines if the timing and sequencing characteristics of the labels over time conform to its previously learned models. If it does not, MAD generates an "unknown code" alert, otherwise it reports "known code."

2.2 Implementation

The CASPER system currently runs on the Ubuntu 16.04 operating system* and builds on open source libraries such as GNU radio and Scikit. The CASPER software has been developed primarily using Python 2 and Python 3. The basic experimental setup is illustrated in Figure 2.

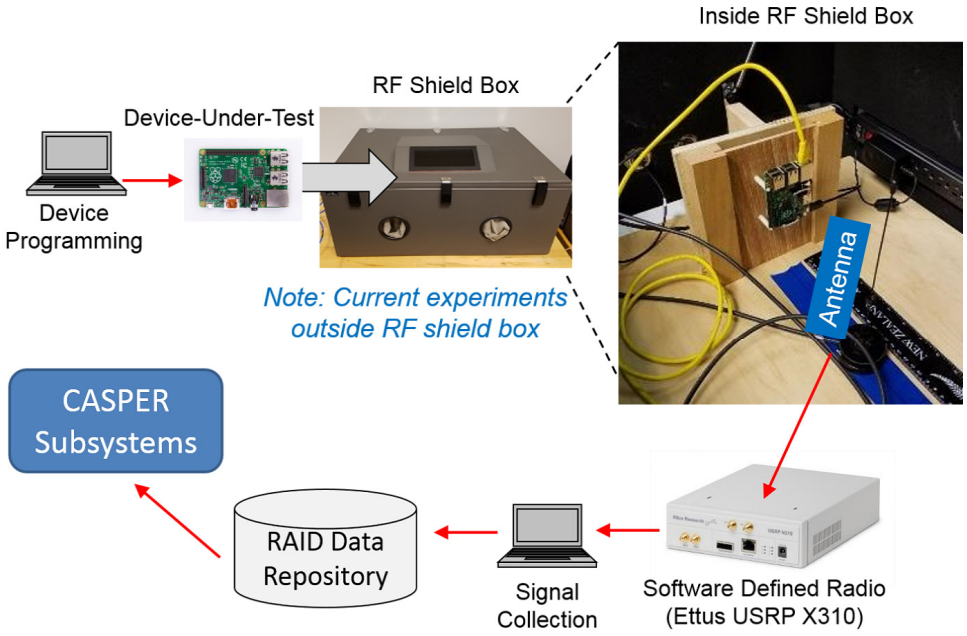

Figure 2. Experimental Setup

The CASPER software runs on a single Ubuntu server. We have successfully tested the CASPER software on a range of new and old commodity computing platforms including laptops and rack-mounted servers. The majority of experiments have used a single Ettus USRP X310 Software Defined Radio attached to a single antenna. We later describe advanced techniques using antenna arrays. These experiments use the same USRP equipped with dual or quad daughter cards. We use commodity antennas including the MobileMark IMAG5-1350 and RFSpace UWB2 600 MHz – 6 GHz Aluminum Vivaldi Antenna.

*http://releases.ubuntu.com/16.04.4/

To minimize the environmental noise, the majority of our experiments were conducted with the device-under-test and antenna inside a Ramsey STE 600 Shield Box. Going forward, our focus is on evaluating CASPER in noisy environments.

2.3 Initial Results

For our initial self-evaluation, we tested CASPER with the device-under-test placed 12" from the antenna. We evaluated the CASPER system against the following devices under test (illustrated in Figure 3):

1. Arduino UNO with ATmega328P processor running at 16MHz clock speed
2. Arduino Zero with ARM Cortex M0+ processor running at 48MHz clock speed
3. Raspberry Pi Model B+ v1.2 with ARMv6 processor running at 700MHz

The device test code exercised CASPERs ability to detect unknown program states and code execution sequences from multiple "known" program states and sequences of states. For the Raspberry Pi, we ran realistic code that used the SHA and StringSearch code from the Mibench test suite.[9] Furthermore the Raspberry Pi code was executed on top of the full Raspbian Linux OS with networking, so the detected emanations were a combination of both the test code and the Linux operating system.

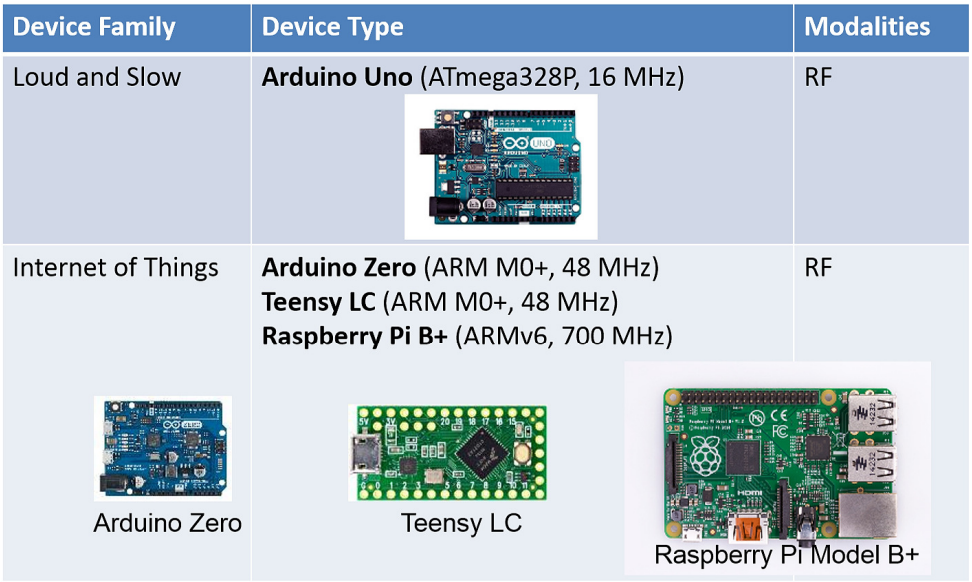

Figure 3. Initial Experimental Devices

For the purposes of our self-evaluation, we defined known code accuracy as $TP/(TP + FN)$ and unknown code accuracy as $TN/(TP + FN)$ where TP is the true positive rate, TN is the true negative rate, FP is the false positive rate, and FN is the false negative rate.

The unmodified Arduino Zero device did not produce emanations that were detectable at the 12" distance, so those rows are grayed out in the table. We compensated by also testing with a modified Arduino Zero in which we soldered a wire to the board. This wire acted as a boost antenna for emanations. The modification was successful as the modified Arduino Zero produced measurable emanations at distance.

Table 2 provides a summary of our evaluation results. The results show for each DUT, the known code and unknown code accuracy for each of the CASPER MAD detectors. Overall, CASPER was able to successfully detect and distinguish known and unknown code with the DUT 12" from the antenna.

Table 2. CASPER Initial Results

Detector	Device	Known Code Accuracy	Unknown Code Accuracy
N-gram	Arduino UNO	100% [0.881, 1]	100% [0.985, 1]
	Arduino Zero (m)	100% [0.767, 1.00]	Mode 9: 100% [0.876, 1.00]
			Mode 13: 91% [0.855, 1.00]
	Raspberry Pi	86.3% [0.808, 1]	100% [0.986, 1]
Statistical	Arduino UNO	100% [0.934, 1]	100% [0.985, 1]
	Arduino Zero (m)	100% [0.767, 1.0]	Mode 9: 100% [0.875, 1.0]
			Mode 13: 92.5% [0.750, 1.0]
	Raspberry Pi	100% [0.983, 1.0]	100% [0.986, 1]
Statistical	Arduino Zero (m)	100% [0.767, 1.0]	Mode 9: 100% [0.875, 1.0]
			Mode 13: 92.5% [0.875, 1.0]
	Raspberry Pi	88.8% [0.816, 1.0]	100% [0.986, 1]

3. MULTI-MODAL ADAPTIVE SENSOR SUBSYSTEM

The goal of the Multi-modal Adaptive Sensors (MAS) system is to (1) measure RF spectral features from a Device Under Test (DUT) in the presence of noise and interference, (2) package device emanation features for the Emanation Model Analytics (EMA) system, and (3) perform data reduction so that a manageable amount of data is passed between systems, enabling real-time operation in subsequent phases. An overview of MAS signal processing is shown in Figure 4.

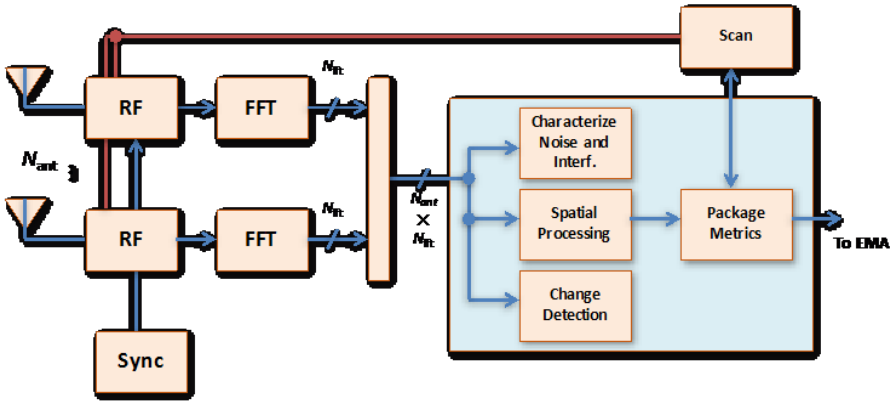

Figure 4. Overview of MAS subsystem signal processing components

MAS accomplishes this task by utilizing array processing to extend range, mitigate noise, and excise interference. Following array processing, data is reduced from the raw sample rate to a lower feature sample rate. The raw sample rate is $f_{in} = N_r \times f_s \times M_s$, where N_r is the number of receive antennas, f_s is the raw sample rate, and M_s is the number of bytes per sample. For an 8-antenna system operating at 20 Msamples/sec/antenna, with 4 bytes per sample, this gives $f_{in} = 640$Mbytes/sec. The feature sample rate is $f_{out} = N_{fft} \times M_f/(N_{avg} \times T_{fft})$ where N_{fft} is the number of points in the FFT, M_f is the number of bytes per feature, N_{avg} is the number of FFT windows that are averaged for each metric, and T_{fft} is the duration of each FFT. This can be rewritten as $f_{out} = f_s \times M_f/N_{avg}$. When 4-byte floating point values are used for features with $N_{avg} = 10$, the resulting rate is 8 Mbytes/sec.

3.1 Signal Models and Assumptions

In phase 1 of the LADS program, the RF signatures used for detection of microprocessor activity consisted of collections of narrowband, tone-like signals. As shown in Figure 5, each program mode, here labeled "A" and "B," is associated with a particular spectral signature.

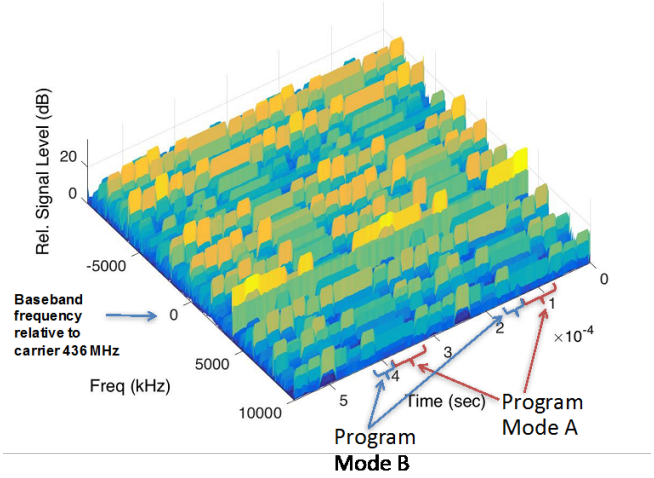

Figure 5. Example of the signal structure of RF emissions in a 20 MHz band centered at 436 MHz from an AVR microprocessor in which different program modes are active

The specific pattern of spectral features varies from one model of processor to another, as well as from one program mode to another; however, as verified during phase 1, consistency exists in the spectral signature for a given mode from one instance of a program mode to another instance of the same program mode, and the signature for a given program mode is sufficiently distinct from signatures from other program modes to allow them to be distinguished.

In the example in Figure 5, it was shown that the approach of identifying and discriminating spectral signatures could be used to distinguish program operating modes involving different forms of memory access (e.g. EEPROM and DRAM), computational mixes, NOOPS, etc. In addition, the length of loops can be distinguished as shown in Figure 6.

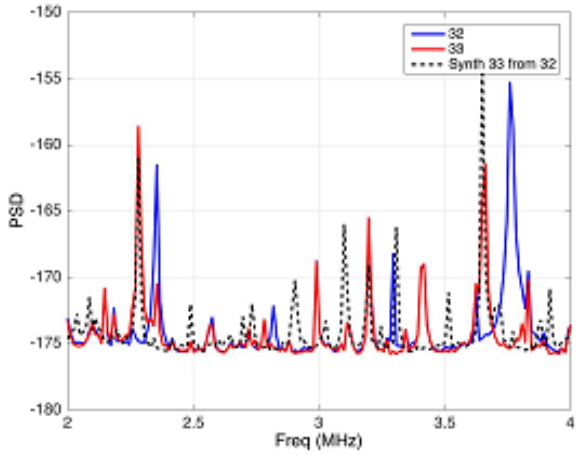

Figure 6. Spectral features measured from an AVR processor in the 1200 MHz band with different loop lengths

In the example in Figure 6, signatures resulting from loop lengths of different lengths are shown, with the blue curve corresponding to a loop length of 32 instructions, and the red curve corresponding to a loop length of 33 instructions. The dashed line is obtained by resampling and stretching the 32-length data by a factor of 33/32, in which we see that several of the synthesized 33-length spectral features align with the measured 33-length spectral features, giving insight into the phenomenology that leads to the differentiation of loop lengths.

3.2 Robust Feature Extraction

One of the practical requirements of MAS provide features to EMA which eliminate the effects of varying measurement conditions and device differences. Positioning of the antenna, temperature in the environment, distance to the DUT can all influence the magnitude of emanations received by MAS. Other factors can influence the frequency bins in which emanations are present.

The AVR family of devices, the Arduino Uno ATMega board design being an example, use a simple clock generation circuit, powered from a poorly regulated supply. While these shortcomings are negligible for the expected usage of AVR devices, precise measurements reveal a causal relationship between the values of registers written to the stack and the frequency of the clock. There is a measurable effect between writing all logic ones versus all logic zeroes to a stack variable. This effect manifests as a change in the drain on the power supply, causing the voltage supplied to the clock to change, resulting in a change in the clock frequency. This frequency change will cause a shift in the location of frequency emission spikes, which if uncorrected will confuse EMA. Additionally, apart from temporal stability issues, the frequency of the oscillator on Arduino UNO boards under no processor load varies greatly. With ARM devices (such as the Raspberry Pi), which have more stable clocking components, this effect is negligible.

In addition to feature variations due to the frequency offsets between the training and monitoring phase due to clock stability or a low quality power supply, simple feature vectors based on raw FFTs may also exhibit variations in amplitude and/or frequency when the training and testing are done at different locations, or more generally speaking, under different channel states.

For all of these reasons, it is important to develop features that faithfully capture the unique electromagnetic signature of a code without any extraneous amplitude and frequency modulation effects.

3.2.1 DFT-based Features

In the initial development of CASPER, the primary set of features calculated by MAS are based on the DFT. In order to capture signal features which exhibit time varying spectral characteristics, we employ Short-Time Fourier Transform (STFT) by taking Discrete Fourier Transforms via the FFT algorithm at fixed intervals on the input RF signal. The STFT is simply a windowed and shifted output of a DFT function, where the input signal $x(m)$ is emphasized by the window function for a particular region in time.

$$X_n = \sum_{m=-\infty}^{\infty} w(n-m)x(m)e^{-j\omega n}$$

The window function applied before the DFT becomes an important parameter in tuning frequency resolution and spectral leakage. Generally, we have relied on the Hann window as a balance between these effects.

In practice to achieve a gain in SNR, adjacent STFT outputs can be averaged together across each bin index, which we refer to as the time-averaging parameter for the DFT based features. We have evaluated various time-averaging factors in finding the ideal tuning parameters for a given processor

$$X_{n_{avg}} = 20\log_{10}\sum_{k=1}^{K}|X_n|$$

Since we are not interested in reconstructing the signal, we do not need to overlap the DFT measurements for the STFT output vectors, instead just output the magnitude of the averaged DFT bins.

In order to combat the effect of amplitude differences due to measurement conditions (distance from the antenna, antenna position or rotation, other time varying effects), we normalize the DFT bins.

$$X_{n_{norm}} = X_{n_{avg}}/\mu$$

where μ is the mean of a single feature vector across the frequency bins.

3.2.2 Template Matching

Basic DFT-based feature vectors may exhibit variations in the location (frequency) and magnitude of the spectral peaks due to a variety of factors for the same program. But even when all other factors are kept constant, ambient noise may bias the basic electromagnetic signature of a particular code. Thus, multiple measurements of emanation due to the same code will look slightly different. Therefore, during the monitoring phase, depending on the level of noise and the proximity of the codes in the feature space, it is possible that a particular measured signature may look a little bit like its "actual" signature and little bit like that of another code. In this case, a reasonable heuristic for maximizing classification accuracy would be to take each measured signature and try to see how close it is to the "actual" signatures of the various possible codes (or program modes), and then, pick the one to which it is closest. This is the idea behind template matching, where we develop centroids for each program modes, and then, project each sample onto the various centroids to develop a new feature vector capturing its "closeness" to the various modes. This is the new feature vector that is passed on EMA. Figure 7 illustrates the template matching approach. Going from right to left, the rightmost block is the stack of centroids (of size R). There are N blocks in the stack corresponding to the N program modes. The centroid for each mode is determined by averaging the training samples corresponding to that mode. Then, the training and monitoring samples, are projected on to these centroids (aka "mode templates") to obtain a new feature vector that of length N. Example centroids and projections for an $N = 5$ case is shown in Figure 8. These training and testing feature vectors that are each N-long are inputs to EMA. Notice that from EMAs perspective nothing has changed as the raw FFT feature vector in the baseline approach described in Section 3.2.1 has simply been replaced by the new "template matching" feature vector.

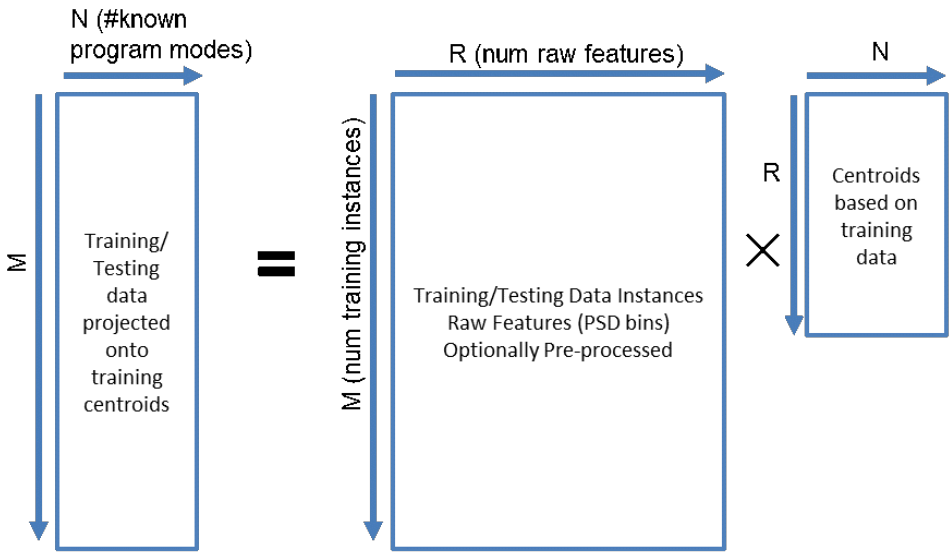

Figure 7. Template Matching Approach

To accommodate frequency offsets between training and monitoring data, we propose a more practical version of Template Matching where, during the projection step, we sweep over a range of frequency shifted versions of the centroid and pick the maximum of those values. Without this step, the projection may result in zeros when there is a frequency offset between training and monitoring samples.

3.2.3 Spectral Correlation

With certain device families and measurement conditions, we have noticed that the training and monitoring DFT-based feature vectors corresponding to a particular program mode vary mostly through a frequency offset. Now, the autocorrelation function, by definition, for a WSS process, $|S(f)|^2$, is only dependent on the relative frequency: $R(\Delta f) = \int_{f_{min}}^{f_{max}} |S(f)|^2 S(f + \Delta f)|^2 df$. This makes the autocorrelation function a good choice for a modified feature vector that is robust to simple frequency shift variations between the training and monitoring

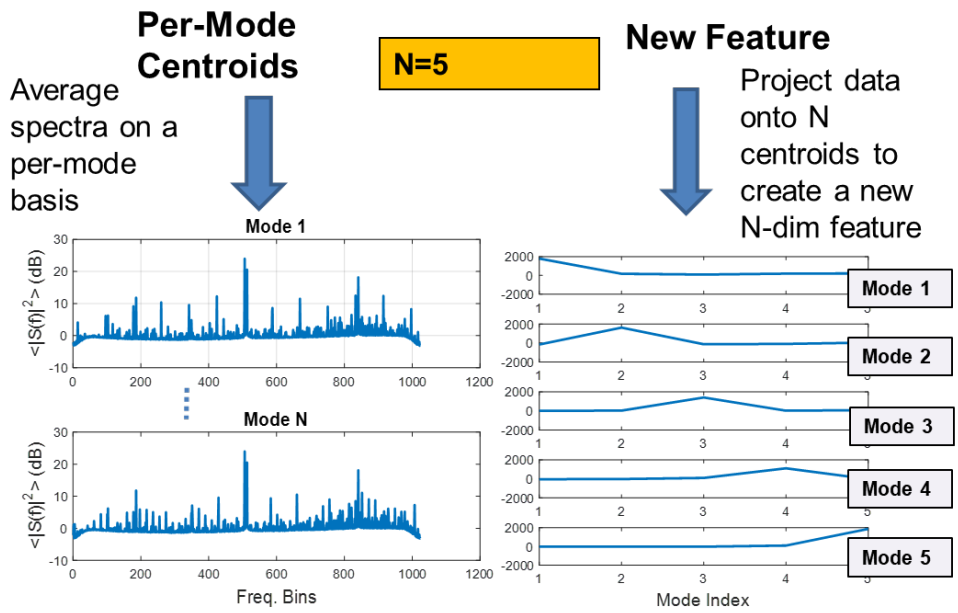

Figure 8. Components of the Template Matching Approach

phase. Figure 9 shows an example feature vector generated for a 5-mode case. The original DFT-based feature vector is given as $|S(f)|^2$. The new feature vector is computed as $R(\Delta f) = \int_{f_{min}}^{f_{max}} |S(f)|^2 |S(f+\Delta f)|^2 df$ followed by a normalization step: $R(\Delta f) = R(\Delta f) / \sum R(\Delta f)$. A tuning parameter in this case is the maximum lag, f_{max}, over which the autocorrelation is computed.

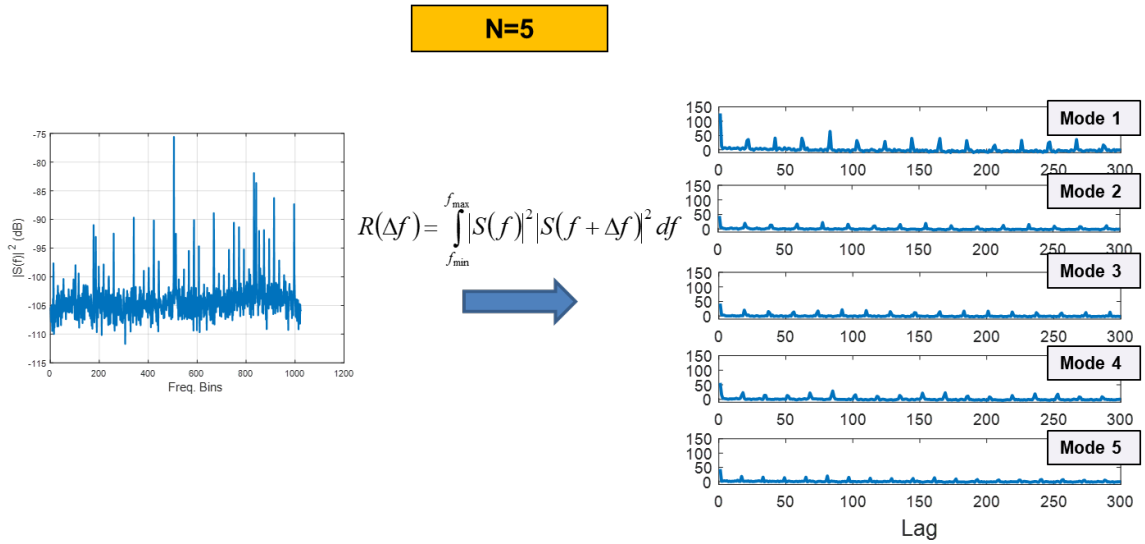

Figure 9. Spectral Correlation as a robust feature for linear frequency offsets.

3.2.4 Array Signal Processing

A key challenge is to extract the low level unintended emission from a device under test in the presence of noise and interference. We first describe hardware to support investigation of array processing methods. Next, we present algorithms for extracting DUT signals in static and dynamic environments, and finally we discuss metrics used to assess the background.

Array signal processing methods are often used in communications problems to extract signals in the presence of noise and interference. In order to do that sort of signal separation, known features of the desired signal (e.g. embedded training sequences, known modulation alphabets, or other signal characteristics, such as a constant envelope) are exploited to develop an adaptive spatial filter which mitigates interference without prior knowledge of the spatial channel.[10] When the desired signal to be detected is unknown, but features of the interference are known, those features can be exploited, an example of which is Self-Coherence Restoral (SCORE) which can use the known baud interval of an interfering signal to spatially null it.[11]

When both the desired signal and the interfering signal are unknown, approaches such as Independent Component Analysis (ICA)[12] can be used to separate signals based on expected statistical distribution; however, this approach can be complicated in environments where interference can have both sub-Gaussian and super-Gaussian characteristics.

If both the desired signal and interference have unknown characteristics, then potential interference mitigation methods can rely on known direction-of-arrival, and known time intervals with the desired signal and interference are active. The former approach is not typically relevant for the current problem because (1) our arrays not calibrated, so we cannot readily associate direction-of-arrival and spatial signature, (2) we may operate in indoor multipath environments, and (3) the direction-of-arrival of neither the DUT nor the interference are typically known a priori. The latter approach is potentially applicable.

3.3 Automated Band Scan

Initially, center frequencies used for RF collection were determined manually by sweeping an RF receiver over spectral bands while operations were performed on the device. A human operator observed the spectral features using a spectrogram, and the bands where time-varying emissions were observed were selected for future processing. This approach is operator-intensive and not scalable, so an automated band scan approach was developed to facilitate characterization of a new devices. Additionally, it has the disadvantage that the human observer may miss features that are significant to the algorithms used by CASPER. The automated band scan approach is illustrated in Figure 10.

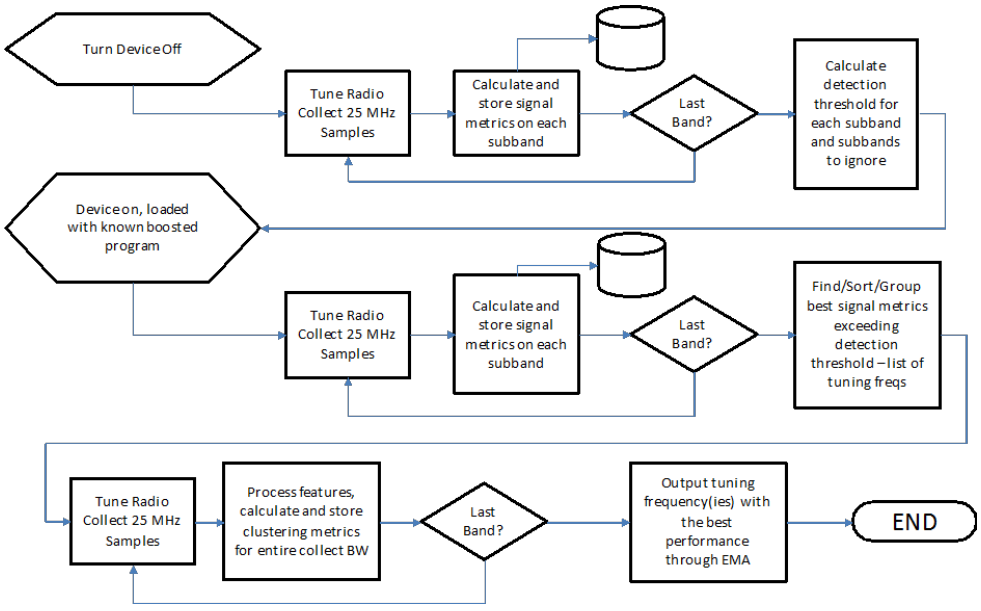

Figure 10. Automated band scan procedure

Since a reasonably-priced SDR can only tune a finite bandwidth, one of the initial tasks of the CASPER system is to find the frequency regions that will lead to the best classification results across program modes. In a noisy environment, it is helpful to first characterize the noise with the device powered off to be able to detect

the frequency bins where the emanations are stronger than the background noise. The program that is running on the device can be boosted to further highlight program state changes.

We are proposing a multi-stage process for searching the spectrum for relevant emanations, with implementation ongoing.

1. Perform a baseline scan to calculate a decision threshold on each subcarrier and characterize interference.
2. Scan with the device on and calculate signal metrics (kurtosis, probability distributions) for each subband.
3. Sort potential tuning frequencies by grouping subbands that exceed decision threshold.
4. For each tuning frequency (or combinations of frequencies), capture data, process features, and perform clustering, sorting the results by the best performance through EMA.

The ultimate goal in the scan is to determine regions that will classify well in EMA processing, but since that piece is computationally expensive to do across the entire spectrum, signal metrics are expected to filter out frequency regions where the program-related emanations do not appear to be strong.

In order to determine which signal metrics will most closely align with the eventual classifiability of the program modes, ground truth data must be obtained. By using the GPIO-labeled collection setup, the classifiability of each frequency tuning location can be determined to determine the viability of the detection threshold.

4. EMANATION MODEL ANALYTICS SUBSYSTEM

This section describes the various techniques employed by EMA. The purpose of the Emanation Model Analytics component is to use machine learning techniques for ingesting emanations to generate a sequence of predicted modes for the DUT. Specifically, input-output mapping engendered by EMA converts the incoming emanations and transforms them into a discrete sequence of modes that correspond to different activities on the DUT. Thus the operations of EMA include both the interaction with MAS to determine the structure of time series instances and the interaction with PPA to validate a suitable set of modes to be subsequently identified by the embedded machine learning classifier.

4.1 Supervised Learning

Our initial technique uses standard supervised machine learning. Specifically, EMA receives labels that are generated by MAS using a novelty detector. One challenge with this approach is that when the mode is changing, the novelty detector can be fooled by the transition. To ameliorate this issue, additional filtering removes instances around any change in mode. For example, by removing 2 instances on each side of the suspected transition instance, we can ensure that if the true transition occurs anywhere in these 5 instances, it will not generate incorrectly labeled instances for training. The goal is to reduce the label noise for EMA-received instances. In addition to the labels, MAS generates 2,000-dimensional feature vector for each instance; the components of this vector are derived from FFT of the signal. The significant size of this feature vector can lead to several complications (curse of dimensionality), including both computational complexity and, more importantly, reduced accuracy of our predictor. (Robustness of classification rules in large-dimensional space is also usually compromised by inevitable shifts in the underlying probability distributions.) In order to address this issue, we compute the mutual information for each of these individual 2,000 features with the goal of selecting a (significantly) smaller subset of them while still retaining the bulk of information contained in the original feature vector. In our experiments, we were able to reduce the number of features by two orders from magnitude (from 2,000 to about 20) with no perceptible loss of classification accuracy. We estimated the accuracy using SVM with various kernel functions, including linear. The latter proved to be the most robust, delivering consistently good performance with low sensitivity to changes in the underlying statistical distributions.

4.2 Multi-class Approach for Unknown Modes

Of course, one of the major challenges with using machine learning for mode prediction is the code coverage problem. It is difficult to determine when there has been sufficient code coverage of the target program and, hence, sufficient training particularly as we look for a general approach that will work without access to firmware[†].

[†]In the future, PPA will provide program analysis and coverage tools to help solve this problem.

Because of this, during monitoring, it is possible to encounter previously undefined modes, which describe a normal behavior, but a normal behavior not seen during training. The challenge is to distinguish this from abnormal behavior, which CASPER is intended to detect.

In this case, EMA predicts the reserved label 0, which denotes the "unknown mode." However, since we do not have access to mode 0 at training time, we need a non-standard learning technique to handle these unseen labels in a proper manner. Our first approach is to solve this problem using the Support Vector Machine (SVM) library based on the Python library scikit-learn.[13] This industry standard library (widely used for data analytics and machine learning tasks) has an implementation of SVM that, given a monitoring instance that could belong to any of pre-defined training modes, can estimate the conditional probability that the given instance belongs to any of these modes. This probability is conditioned by the intermediate SVM score that indicates how far the given instance is located from the decision boundary - the larger is that distance, the larger is the probability, or confidence, that the given classification decision is correct. Our algorithm returns mode 0 if none of the training modes has a sufficiently large (above a threshold) conditional probability. In our current setting, we use the probability threshold of 0.5 as a reasonable default since no more than one mode can have a probability greater than 0.5. (The sum of all conditional probabilities has to be equal to 1).

4.3 Anomaly detection for unknown modes

Yet another alternative for detecting unknown modes (described in the previous subsection) is to employ anomaly detection. Our algorithm for labeling instances during monitoring is a straightforward combination (cascade) of anomaly detection algorithms and SVM. If an instance is classified as an anomaly for all our training mode anomaly algorithms, then our algorithm labels it as "unknown mode." Otherwise, the instance is passed to the SVM that then predicts the label. In this case, the prediction corresponds to the instance that has the highest probability based on the learned SVM model. While it is theoretically possible to carry out anomaly detection on the full 2000-dimensional FFT feature vector, the results will be very susceptible to the curse of dimensionality. Statistically speaking, in spaces of such high dimensionality, almost all the points will be located "at the boundary" of the dataset, thus making most data analytics methods (especially unsupervised ones) practically useless. Therefore, we perform anomaly detection in a lower dimensional space created by PCA algorithms (which rotates the data in its 2,000 dimensional embedding space in order to align it along the coordinate axes with largest variability). As can be seen in Figure 11 (which shows clusters for 11 modes seen on a simple Arduino UNO program), even a three-dimensional PCA space can provide a quite informative representation of the training modes.

4.4 Clustering to Generate Training Labels

Given a good PCA representation of our data, it makes sense to consider clustering to label our training data - in other words, using *de facto* cluster IDs created by the clustering algorithms as artificial labels. Indeed, the actual goal of CASPER is to flag malicious device behavior, and, whether that behavior is encoded with semantically correct device-specific computer modes or whether it is encoded with artificial clustering-derived labels - all that matters is that we can detect the change within that encoding system. Initially, labels were generated by the novelty detector which looks for large novel changes in the emanations to mark a change in the label. However, for more complex devices and programs, the novelty detector has difficulty differentiating legitimate transitions between modes from sporadic noise. Noise becomes an important issue when dealing with more complex programs and devices.

A clustering algorithm can be effective in removing noise since it can simply identify that an instance is not part of any recognizable cluster and label it as "unknown code." We use a simple and efficient clustering algorithm, DBSCAN,[14] that is especially effective on low-dimensional spaces that have well defined clusters. The DBSCAN algorithm has two primary parameters; however, since it is used within our training process, we allow a human to interact with the system to adjust the parameters for finding an acceptable clustering that matches the general program behavior. This human calibration and fine-tuning can leverage the knowledge of the characteristics of the known code, or it simply can be based on visual assessment of the clustering partition within the obtained three-dimensional PCA space. Currently, this fine tuning is relatively easy for a non-expert, and we plan to automate this process which will also allow the exploration of higher dimensional PCA spaces.

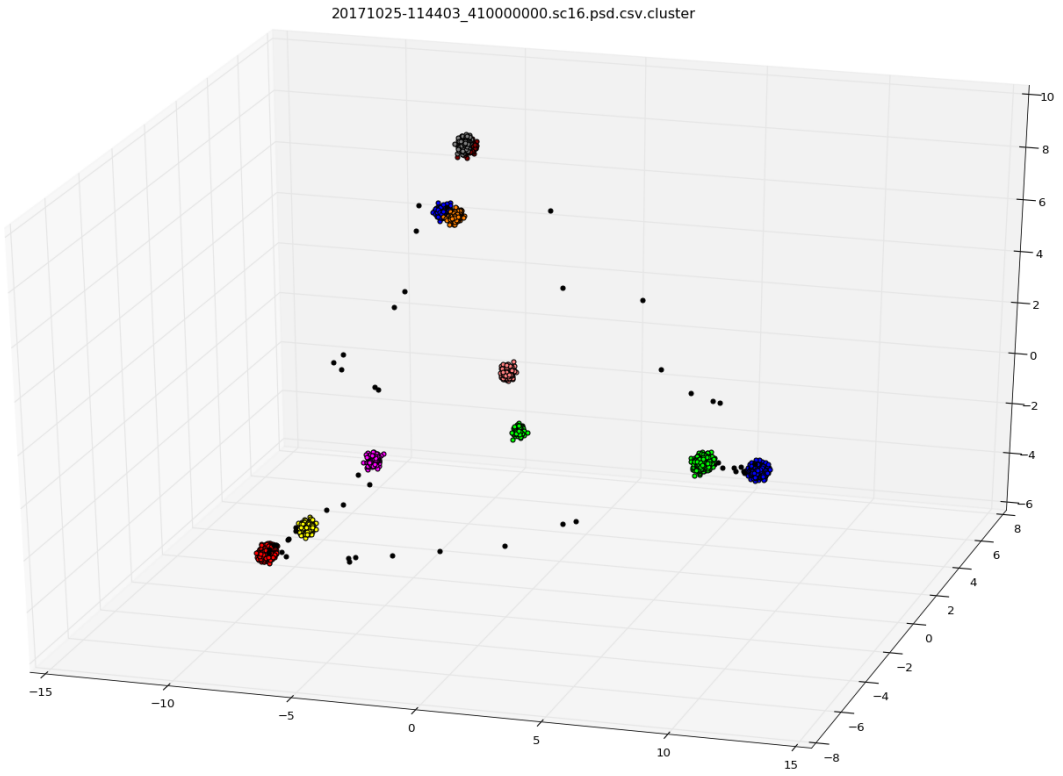

Figure 11. PCA space for UNO with 11 modes

We associate with each cluster a mode (artificial label). As mentioned above, while it may be possible to attempt to associate a DUT meaning with each mode, it is not necessary since it is sufficient to send a stream of consistent mode predictions to PPA. In other words, current PPA techniques only require that instances from the same cluster get a unique label; they do not require a semantic meaning be assigned to those clusters. However, for more rigorous study, we can attempt to connect the modes to the DUT by recording information such as the amount of time spent in sections of the program code and seeing if they correlate to sequences of instances inside clusters.

4.5 Clustering for Monitoring Mode Predictions

Our final algorithmic refinement is to exploit the DBSCAN clustering for monitoring. It is difficult to envision a situation where an instance belongs to cluster 1, yet the algorithm predicts mode 2 based on an SVM trained with the same label information that was generated by the clustering algorithm. Instead, we can simplify our algorithm both conceptually and computationally by carrying out classification based on the cluster-based artificial labels. More precisely, at the monitoring stage, we map the new instance to the PCA space and generate an ϵ-ball around the point. The algorithm predicts based on the majority label of training instances in that ball. If there are no instances in the ball, then the algorithm predicts label 0. Notice that there are two ways label 0 could be predicted. The algorithm will predict mode 0 if the ϵ-ball is empty or if the majority of labels in the ball have label 0. The setting of the ϵ parameter can be identical to the parameter used for labeling the training data, but it can be useful to use a slightly larger value to provide some margin for error. This extra margin is still robust to noise as the majority label of the instances inside the ϵ-ball should still predict the correct label even for clusters that are close in the PCA space. While a naive implementation of this nearest neighbor based approach creates an expensive monitoring algorithm, scikit-learn has an efficient implementation that can use a k-d tree which is extremely effective for computing ϵ-balls in low dimensional spaces.

5. PROCESSOR AND PROGRAM ANALYSIS SUBSYSTEM

Processor and Program Analysis (PPA) is the subsystem responsible for keeping processor-specific information. PPA does the majority of its work in the Micro-benchmarking and Normal Behavior Characterization Phases. While our efforts in this area are on-going, we present our initial approach.

Currently, given a new processor and even new firmware for an existing processor, a subject matter expert spends time trying to learn where we are most likely to see useful emanations. In particular, given that we are limited by hardware (and physics) in the portion of the spectrum that can be practically monitored, it is critical to identify a compatible band that contains enough information to allow decisions about the state of the processor. To do this, we consider both general information about sources of emanations and information specific to the processor.

As one example of the former, on many processors as a loop is run for more or fewer iterations, it produces a signal at higher or lower frequency. In processor-based micro-benchmarking, PPA will create programs with variations in loop iterations to determine if this property holds true for the new processor. Assuming that it does, then program analysis can find loops and determine the range of possible iterations in order to predict what signals should be seen. Based on that prediction, firmware-specific micro-benchmarks are generated and run to determine whether that prediction holds.

We plan to extend the initial version of tool, GenTest, to build tests to determine whether it sees changes in the emanations based on changes of address, different instruction classes (*e.g.,* FPU vs. integer) as well as particular instructions (*e.g.,* addition vs. division), and different types of control flow. This will be combined with the DynInst tool, which analyzes binaries. By combining these functionalities, we intend to use code from the target program, whenever possible, as the basis for our micro-benchmarks.

6. MONITORING ANOMALY DETECTION SUBSYSTEM

CASPER utilizes three different unknown code detection techniques: statistical, n-grams, and NFA. All detectors consist of a training phase and a monitoring phase. During the training phase, each detector constructs an appropriate baseline for what it considers to be normal device state (based on the device running known code). During the monitoring phase, each detector compares current device state to its baseline for determining whether the device is executing unknown code.

The statistical detector relies upon the statistical distributions of EMA label predictions for determining whether a device is running known or unknown code. The n-gram detector relies upon distinct EMA label prediction sequences of a certain length. The NFA detector relies upon a non-deterministic finite automaton that is constructed from transitions between EMA label predictions. Each of these algorithms will be described in detail in subsequent subsections.

Initially, these detectors were used independently of each other and we reported results for each one of them. Each detector has its own strengths and weaknesses, as shown in Figure 12. We are working to integrate them into an ensemble so that we can take advantage of the strengths of each detector under different contexts.

6.1 Statistical Detector

The statistical detector relies on the distribution properties of EMA label predictions. The main premise is that the distribution properties of label predictions for known device code differ significantly from the distribution properties of label predictions associated with unknown device code. Therefore, by representing EMA label predictions as vectors in a multidimensional space where dimensions correspond to labels and values correspond to ratios of label counters, unknown code label vectors will be far (distance-wise) from each known code label vector. The following figure illustrates this for a Raspberry Pi device and several EMA label prediction outputs, with each output corresponding to 10 seconds of captured device emanations.

In Figure 13, the top left heatmap illustrates pair-wise distances for 9 different EMA label prediction vectors associated with known (*i.e.,* "good") device code. The table below this heatmap includes the counters for each of the 5 labels (0-4); one row per 10 second EMA label predictions. The top right heatmap in the same figure illustrates pair-wise distances for 10 different EMA label prediction vectors associated with unknown (i.e., "bad")

Detector	Strengths and Weaknesses
Statistical	+ Handles mislabeling in both training and monitoring data assuming that similar conditions cause mislabeling in training and monitoring + Does not require any a-priori information about device operations - Requires long enough training for statistics to have good convergence - Cannot detect out-of-order executions with similar label distributions
N-gram	+ Simplicity and scalability + Can detect out-of-order execution (when n >= 2) - Cannot detect timing discrepancies in execution durations - N-gram size and detection threshold selection not automated
NFA	+ Learns device state timing distributions during training + Can detect both state transition and state timing anomalies - Control flow anomaly detection only (e.g., out of order executions) - Mostly manual construction of NFA at the current time

Figure 12. Complementary strengths and weaknesses of detector approaches

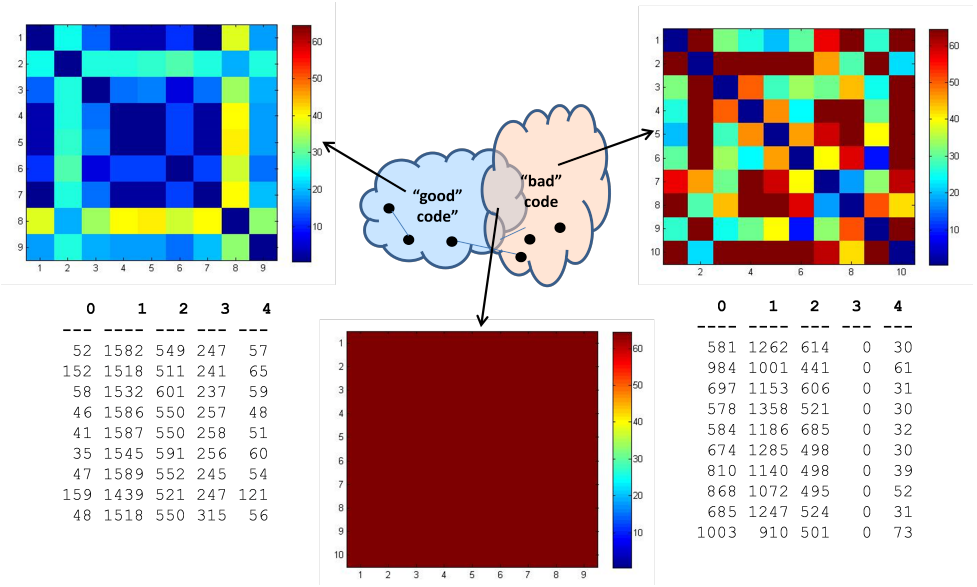

Figure 13. Statistical detector illustration

device code. The table below this heatmap includes the counters for each of the 5 labels (0-4); one row per 10 second EMA label predictions. The heatmap at the bottom center of the figure illustrates the pair-wise distances between known and unknown code EMA label prediction vectors. As it is evident from the latter heatmap, unknown code EMA label prediction vectors are "far" (distance-wise) from known code EMA label prediction vectors.

During training, the statistical detector computes individual label frequencies for a number of emanation collections associated with known device code. For each collection c, a vector of label frequency ratios is computed. Next, the covariance matrix is computed by treating each ratio as a random variable.

During monitoring, the following steps are followed:

1. Compute individual label frequencies for a monitoring duration that is identical to the durations used for training;

2. Compute a vector of label frequency ratios similar to training;

3. Use Mahalanobis distance to compute a distance score for monitoring collection;

4. If score is greater than a threshold, declare unknown code otherwise declare known code.

To avoid the problem of setting a threshold, we use the following algorithm. During training, we compute the threshold by taking 5,000 bootstrap samples from the collections. Each bootstrap sample is based on randomly selecting 5 collections as a holdout set. The covariance matrix is computed for the remaining collections and the Mahalanobis distances is computed for the 5 holdout collections. Since this is repeated 5,000 times, this creates 25,000 distance scores that can be used to empirically estimate the distribution of the Mahalanobis distance when applying an unseen collection of known device emanations.

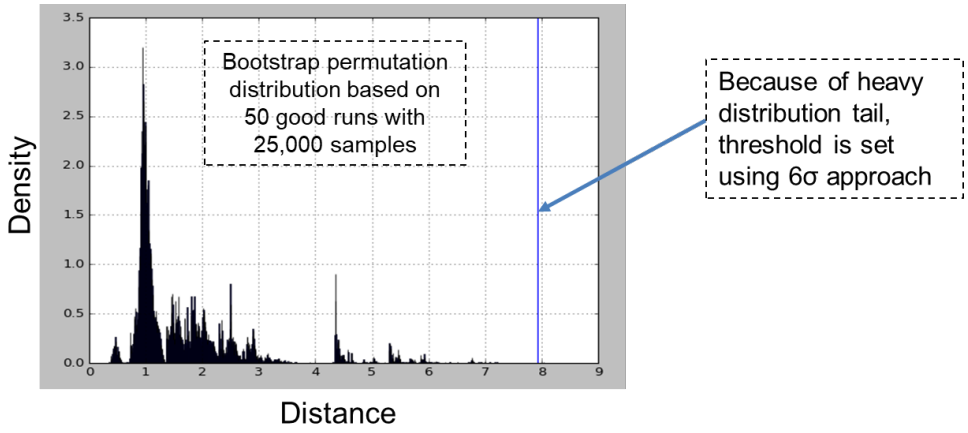

Figure 14. Example bootstrap distribution for Raspberry Pi

To satisfy the metric, we want our algorithm to predict known code with high probability when presented with a collection of known code. Therefore, we use the common 6σ approach and set the threshold to the mean of the bootstrap distribution plus six times standard deviation of the distribution. When there is sufficient training data, this creates a threshold that safely predicts known code collections. An example of the bootstrap distribution for the Raspberry Pi is given in Figure 14.

6.2 N-Gram Detector

An n-gram is a contiguous sequence of n elements from a given sequence of elements. Depending on the domain area, an element may correspond to different entities. For example, in the area of computational linguistics, an element may correspond to letters, words, syllables, or phonemes. In CASPER, an element corresponds to an EMA prediction, *e.g.,* label assigned to a distinguishable device emanation. Because the labels correspond to arbitrary portions of the program, CASPER simply identifies them by number. As described in Section 4, label 0 corresponds to the special case of an unknown mode (which typically occurs when a transition between modes is occurring). The following example shows a sequence of EMA predictions for an Arduino UNO device.

2 2 2 2 2 2 2 2 0 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 0 4 4 4 4 4 4 4 4 4 4 0 3 3 3 3 3 3 3 3 3 3

When computing n-grams for a long sequence of elements, one has to consider whether to allow overlapping or not. If overlapping is allowed, the degree of overlap would have to be specified, *i.e.,* how many elements are shared between two consecutive n-grams. Typically, $n-1$ elements are selected to overlap in n-grams (the last $n-1$ elements of one n-gram are the first $n-1$ elements of the next n-gram). In Table 3, we have taken the list of predictions above and broken them into different size n-grams to illustrate.

The CASPER n-gram detector detects unknown code running on a device by comparing n-grams computed from EMA predictions during device monitoring against n-grams computed from EMA prediction during device training. When more than a specified (configuration parameter) percentage of distinct n-grams computed from monitoring EMA predictions differs from distinct n-grams computed from training EMA prediction, the CASPER

Table 3. CASPER phases and subsystems

3-grams	4-grams	5-grams	6-grams	7-grams	8-grams	9-grams
2 2 2	2 2 2 2	2 2 2 2 2	2 2 2 2 2 2	2 2 2 2 2 2 2	2 2 2 2 2 2 2 2	2 2 2 2 2 2 2 2 0
2 2 2	2 2 2 2	2 2 2 2 2	2 2 2 2 2 2	2 2 2 2 2 2 2	2 2 2 2 2 2 2 0	2 2 2 2 2 2 2 0 8
2 2 2	2 2 2 2	2 2 2 2 2	2 2 2 2 2 2	2 2 2 2 2 2 0	2 2 2 2 2 2 0 8	2 2 2 2 2 2 0 8 8
2 2 2	2 2 2 2	2 2 2 2 2	2 2 2 2 2 0	2 2 2 2 2 0 8	2 2 2 2 2 0 8 8	2 2 2 2 2 0 8 8 8
2 2 2	2 2 2 2	2 2 2 2 0	2 2 2 2 0 8	2 2 2 2 0 8 8	2 2 2 2 0 8 8 8	2 2 2 2 0 8 8 8 8
2 2 2	2 2 2 0	2 2 2 0 8	2 2 2 0 8 8	2 2 2 0 8 8 8	2 2 2 0 8 8 8 8	2 2 2 0 8 8 8 8 8
2 2 0	2 2 0 8	2 2 0 8 8	2 2 0 8 8 8	2 2 0 8 8 8 8	2 2 0 8 8 8 8 8	2 2 0 8 8 8 8 8 8
2 0 8	2 0 8 8	2 0 8 8 8	2 0 8 8 8 8	2 0 8 8 8 8 8	2 0 8 8 8 8 8 8	2 0 8 8 8 8 8 8 8
0 8 8	0 8 8 8	0 8 8 8 8	0 8 8 8 8 8	0 8 8 8 8 8 8	0 8 8 8 8 8 8 8	0 8 8 8 8 8 8 8 8
8 8 8	8 8 8 8	8 8 8 8 8	8 8 8 8 8 8	8 8 8 8 8 8 8	8 8 8 8 8 8 8 8	8 8 8 8 8 8 8 8 8
...	...	...	...	...	...	...

Configuration

```
let ngram = n-gram size
let threshold = alert threshold in [0, 1)
let S = {}   // Set of distinct n-grams
```

Training Phase

```
for each file containing EMA label predictions:
    let label_array = array of EMA label predictions in the file
    for n in range(0 .. len(label_array)-ngram+1)
        token = ''.join(x for x in label_array[n:n+ngram])
        S = S union {token}    // Add token to set of distinct n-grams
```

Monitoring Phase

```
let ntotal = 0.0
let nfound = 0.0
for each file containing training EMA label predictions:
    let label_array = array of EMA label predictions in the file
    for n in range(0 .. len(label_array)-ngram+1)
        token =       .join(x for x in label_array[n:n+ngram])
        ntotal += 1
        if token not in S
            nfound += 1
    if nfound/ntotal >= threshold
       alert( Unknown code detected )
```

Figure 15. N-gram detector algorithm details

n-gram detector generates an alert indicating invalid device operations. Details are presented below in the form of pseudocode.

As mentioned above, the n-gram detector uses two configuration parameters: n-gram size and alert threshold. The selection of the appropriate n-gram size depends on two factors: the number of distinct predictions labels generated by EMA during training, and statistical information about consecutive prediction label sequences. In general, the longer the sequence of consecutive prediction labels the smaller the size of n-gram.

The selection of alert threshold depends on the tolerance one has to uncertainty about whether a device is operating normally or not. Higher thresholds can be used for higher tolerance levels. The need for a threshold is due to the probabilistic nature of EMA predictions and the fact that device emanations can be impacted in various ways by interference and other environmental conditions. Such impacts may result in incorrect EMA predictions, even when the device is operating normally. Such prediction errors may result in monitoring n-grams

that do not appear in the training set of n-grams and, hence, count as a miss. The default threshold setting is 0.0001 (1 wrong n-gram in 10,000 n-grams). We continue to investigate ways in which the threshold can be set based on analysis of the EMA training results.

Note that the n-gram detector algorithm shown in Figure 15 works with files containing EMA label predictions. This mode of operation corresponds to the current CASPER workflow where device emanations are collected for a period of time, processed by MAS, and then EMA creates one or more files containing predictions. However, the overall approach can easily be modified to work in a streaming mode.

6.3 NFA Detector

Another way to detect if a device has been compromised is to build a state machine that represents the correct functioning of that device, and then check, on an ongoing basis, if the emanations resulting from it conform to that state machine. This state machine may be nondeterministic, so we refer to it as a Nondeterministic Finite Automaton, or NFA, for short. This requires two tasks:

1. First, during a training phase, construct that NFA by monitoring the sequence of emanation-labels inferred by EMA during the normal device state.
2. Then, monitor the analogous sequence of emanation-labels, again supplied by EMA, and check if they conform to that NFA. If they do not, declare unknown code.

7. CONCLUSIONS

CASPER is an ongoing effort to provide a comprehensive adaptive approach to unknown code detection by monitoring the devices unintended RF emissions. It operates at near real-time speeds on commodity compute servers and the widely available USRP receive. Our approach works under the conditions described at 12". The CASPER team continues to refine the techniques to increase that distance as well as to allow higher processor speeds and more complex software on the device.

Current work is focusing on extending the system to the power modality and additional devices. Additionally, we are focused on increasing the level of automation in the system both to allow it to self-adapt through techniques like the MAS Band Scan and to adapt to environmental changes or changes in device behavior over time.

ACKNOWLEDGMENTS

The authors are grateful for the support of DARPA I2O and AFRL via contract number FA8650-16-C-7623 and the LADS program. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] "Leveraging the analog domain for security (LADS)," (September 2015).

[2] Genkin, D., Pipman, I., and Tromer, E., "Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs." Cryptology ePrint Archive, Report 2014/626 (2014). https://eprint.iacr.org/2014/626.

[3] Genkin, D., Pachmanov, L., Pipman, I., and Tromer, E., "Stealing keys from PCs using a radio: Cheap electromagnetic attacks on windowed exponentiation," in [*CHES*], 207–228, Springer (2015).

[4] Callan, R., Zajić, A., and Prvulovic, M., "FASE: Finding amplitude-modulated side-channel emanations," in [*Proceedings of the 42nd Annual International Symposium on Computer Architecture*], *ISCA '15*, 592–603, ACM, New York, NY, USA (2015).

[5] Zajic, A. R. and Prvulovic, M., "Experimental demonstration of electromagnetic information leakage from modern processor-memory systems," *IEEE Transactions on Electromagnetic Compatibility* **56**, 885–893 (2014).

[6] Jerkins, J. A. and Stupiansky, J., "Mitigating IoT insecurity with inoculation epidemics," in [*Proceedings of the ACMSE 2018 Conference*], *ACMSE '18*, 4:1–4:6, ACM, New York, NY, USA (2018).

[7] Blossom, E., "Gnu radio: Tools for exploring the radio frequency spectrum," *Linux J.* **2004**, 4– (June 2004).

[8] Ettus, M. and Braun, M., [*The Universal Software Radio Peripheral (USRP) Family of LowCost SDRs*], ch. 1, 3–23, Wiley-Blackwell (2015).

[9] Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., and Brown, R. B., "Mibench: A free, commercially representative embedded benchmark suite," in [*Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*], *WWC '01*, 3–14, IEEE Computer Society, Washington, DC, USA (2001).

[10] Liberti, J. C. and Rappaport, T. S., [*Smart Antennas for Wireless Communications: IS-95 and Third Generation CDMA Applications*], Prentice Hall PTR, Upper Saddle River, NJ, USA (1999).

[11] Agee, B. G., Schell, S. V., and Gardner, W. A., "Spectral self-coherence restoral: a new approach to blind adaptive signal extraction using antenna arrays," *Proceedings of the IEEE* **78**, 753–767 (Apr 1990).

[12] Hyvärinen, A., Karhunen, J., and Oja, E., eds., [*Independent Component Analysis*], Wiley (2001).

[13] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research* **12**, 2825–2830 (2011).

[14] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X., "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in [*Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*], *KDD'96*, 226–231, AAAI Press (1996).