Power-based Malware Detection in Linux Boot

Ge Li[†], Alexander Cathis[†], Andrew Lu[*], Shijia Wei[†], Ross Brown[‡], Alexander Liu[‡], Mohit Tiwari[†], Andreas Gerstlauer[†] and Michael Orshansky[†]
Department of Electrical and Computer Engineering, The University of Texas at Austin[†]
Liberal Arts and Science Academy[*]
Applied Research Laboratory, The University of Texas at Austin [‡]
{lige, alexander.cathis, shijiawei, tiwari, gerstl, orshansky}@utexas.edu
lu.andrew@gmail.com {rbrown, aliu}@arlut.utexas.edu

ABSTRACT

We develop the first non-intrusive power-based malware detection method to ensure security of boot in an embedded system with the Intel Xeon-class CPU. Existing approaches to power-based malware detection are not effective for boot sequence because they consider a full power trace non-discriminately. Yet in a long boot power trace, security-relevant information is contained in a limited time interval. We demonstrate that the features based on optimal strategically-chosen phases are better in classification accuracy than those based on the full trace, under different feature and classifier selections.

We investigate two threats: an untrusted device and a compromised kernel. We evaluate a supervised approach, requiring prior knowledge of malware, and an unsupervised approach, capable of zero-day detection. Identifying optimal phases for each attack leads to 10% improvement in average accuracy, compared to baseline classifiers trained with full trace features directly. We demonstrate an ensemble classification scheme which can be constructed with a reduced training cost (a 4.3X reduction). The classifier ensembles optimal phase classifiers of a limited, representative set of attacks. Our results show that the ensemble classifier improves average accuracy by 5% (to 77%) compared to the baseline full trace classifiers.

KEYWORDS

malware detection, secure boot, power measurement, machine learning

1 INTRODUCTION

An increasing number of malicious exploits poses serious threat to embedded systems deployed in various applications including general purpose computers [3], mobile phones [9], medical devices [8] and SCADA systems [18]. Investigation of more effective malware detection techniques against the increasingly sophisticated malware is needed.

Signature-based malware detection extracting fingerprint from the executable has been commonly adopted. However, polymorphic and metamorphic malware which changes code in each infection, while keeping the same functionality, can easily evade the signature-based detection techniques [1, 21]. To overcome the limitation, detection techniques based on program behavior observation have been investigated [4, 24]. Among the various behavior-based detection techniques, those monitoring the out-of-band side channel information, such as power consumption or EM emanation, appear promising [3, 5, 8, 9, 11, 16, 17, 27, 30].

The boot sequence is the initial process executed on any computing system. Said process is especially important to an embedded system since it is responsible for loading the kernel image and setting up the operating system properly by executing a fixed set of tasks. Boot is often the target of malicious exploits [7, 12, 15]. The adversary attempts to tamper the components involved in boot, including the bootloader [7] or BIOS [15], with the goal of gaining control of the operating system or for other malicious goals. Conventionally, various secure boot schemes that rely on cryptographic primitives have been proposed [10, 20, 22]. Since the boot sequence is well-defined, secure boot schemes aim to check integrity of the next stage before stage transition. This integrity checking is conventionally achieved by digital signature schemes [22] or hash functions [10]. Although these schemes provide strong security guarantees, implementation of a cryptographic module, execution of the cryptographic primitive, and the storage of the signatures incurs a large overhead that embedded systems can often not afford. Furthermore, several attack vectors based on fault injection [2] and malicious hardware insertion [13] have been reported to circumvent secure boot schemes. In contrast, power-based malware detection has the potential to overcome these challenges. However, no prior work has investigated the feasibility of using power-based malware detection for boot security.

Compared to other approaches, which require dedicated software or hardware support, power-based malware detection is non-intrusive and has low overhead. The basic premise is that a device's power signature relies on the behavior of the workload. Power-based malware detection techniques require the following steps: (1) instrumenting front-end sensors to measure power draw of the device, (2) training a classification algorithm using power traces under benign and malicious workloads (under supervised approach), and (3) predicting in real-time whether the test workload is benign or malicious using the observed power trace.

Power-based malware detection using machine learning (ML) has been shown to achieve high performance in various applications. In [8], Clark et al. split the entire power trace into equal length segments and extract time and frequency domain features from each segment. The authors treat each segment as an individual instance non-discriminately, and train ML classifiers with both benign and malicious samples. The investigated malware exhibits periodic pattern across the entire trace, and good accuracy is achieved. In [3], Bridges et al. extract and characterize features from the entire trace of a benign workload. They construct an ensemble of one-class anomaly detectors each based on a single feature. Their approach demonstrates great performance on the general-purpose

Table 1: A summary of leading works for power-based malware detection.

Methodology	Work	Target Attack	Platform	System Operating Frequency	Sampling Rate	Algorithm
Direct Workload Classification	Clark et al. [8]	Untargeted malware for clinical and industrial environments	Compounder, Substation computer	664MHz	250KS/s	3-NN, RF, MLP
	Bridges et al. [3]	Rootkits	General purpose computer	NA	59S/s	Detector ensemble, SVM
	Caviglione et al. [5]	Covert channels	Smartphone	NA	1S/s	NN, DT
	Wei et al. [27]	Micro-architectural attacks	SoC	2GHz	200S/s	LSTM, ocSVM
	This work	**Boot sequence tampering**	**Multi-core server-class CPU system**	**1.6GHz**	**2KS/s**	**SVM, LR, DT, ocSVM**
Instruction Sequence Construction	Liu et al. [17]	Instruction sequence modification	Microcontroller	11MHz	1.25GS/s	HMM
	Park et al. [19]	Instruction sequence modification	Microcontroller	16MHz	2.5GS/s	SVM, Naive Bayes
	Yilmaz et al. [29]	Instruction sequence modification	FPGA with processor	50MHz	25.6MS/s	Direct profile comparison

computer rootkits which leave fingerprint across the entire power trace. In [5], Caviglione et al. investigate both regression-based and classification-based approaches to detect covert channels on a mobile phone. The regression-based approach trains a regressor based on the power trace of a clean state, and labels anomaly if the real-time power trace deviates too much from prediction. The classification-based approach trains a classifier based on features extracted from each segment split from the entire trace. Both are demonstrated to be effective on covert channels which leave repetitive patterns on a clean state power trace. In [27], Wei et al. aim to detect anomaly in the various benign workloads with short and repetitive patterns, such as face detection and room navigation. The authors perform the Discrete Wavelet Transform (DWT) on each segment of a full trace. One-class SVMs and LSTMs are trained for classification which use transformed segments as features. The detector is demonstrated to be effective on both ordinary and power mimicry malware, which emulates the power signature of benign applications. These prior works secure run-time applications which typically have short, regular or periodic behavior. *Compared to earlier work on power-based malware detection, the boot process consists of multiple unique stages and is much longer.*

Instead of distinguishing malware directly by power signature, another line of work aims to first construct the underlying instruction sequence from power measurements [17, 19, 29]. The recovered instruction sequence is compared against known benign workloads. These methods require expensive measurement setups, and have been demonstrated only on relatively simple target system, such as microcontrollers operating at tens of MHz [17, 19]. Although instruction-level tracking can provide superior detection accuracy, it is challenging to use this idea on the boot sequence. Boot is much longer than the commonly profiled benchmarks. The cost to track every instruction along the entire sequence is expensive. It also has not been demonstrated on a modern multi-core system operating at GHZ range frequency, which is our target platform. Therefore, we follow the direct workload classification methodology established by [8], [3], [5] and [27]. We summarize the works mentioned above in Table 1.

We develop a detector for the boot process of the Linux operating system which is widely used in embedded systems. **Existing approaches are not suitable for the boot sequence because they consider a full power trace non-discriminately. In reality, boot sequence tampering signature is contained in a limited time interval out of a long signal**. We propose to extract features from the strategically-chosen phases of a trace to improve classification accuracy. In order to identify such phases efficiently, instead of performing an exhaustive brute-force search on the full trace, we generate a limited set of candidate time intervals, guided by the events in the boot process. The optimal phase is picked from this set of candidate intervals. This methodology is generalizable to different systems, as the boot sequence remains fixed across different deployments.

Two threat models are investigated. The first is the untrusted device threat model, which considers untrusted devices plugged in a confidential system. The security of a confidential system can be compromised by untrusted devices. The untrusted devices, owned by an adversary, that are attached to the confidential system can inject malware [14], exfiltrate sensitive data [23], or even physically damage the system [25]. We aim to detect untrusted devices plugged in the system at boot time. Since the operating system loads device drivers for the plugged-in devices during the boot process, we emulate the untrusted devices threat model by loading un-intended device drivers during boot.

The second model is the compromised kernel threat model and it considers the system boots into a compromised kernel. Linux kernel is the core of the Linux operating system and is responsible

for managing the processes and hardware in the system. A compromised Linux kernel can expose the system to various threats, such as privilege escalation, information leakage, and denial-of-service [6, 28]. The threat we consider is that the adversary tampers with the boot loading process, such that a compromised kernel is loaded. The second model is emulated by loading a Linux kernel of a different version. We design an experimental set-up for collecting power traces from an embedded system with an Intel Xeon-class CPU to demonstrate the effectiveness of the power-based malware detection.

Several commonly adopted training approaches in literature are evaluated, under the supervised approach, requiring prior knowledge of malware, and the unsupervised approach that is able to achieve zero-day detection. We show that features extracted from the optimally chosen phases are significantly better than those based on the full-trace analysis in both supervised and unsupervised settings. The conclusion holds for different combinations of features and classifiers. We show that the use of statistical features with Support Vector Machine (SVM) or Logistic Regression (LR) leads to optimal performance. We show that the technique outperforms other leading classification-based and regression-based techniques used in [8], [3], [5] and [27]. We investigate multiple cases with varying amounts of deviation from the baseline boot, for each threat model, and derive the operating range of the technique.

Our results show that even a single-driver difference anomaly can be detected with 72% accuracy. The minimal kernel version difference anomaly can be detected with 97% accuracy. We compare our method to other techniques used in [8], [3], [5] and [27]. Other work shows sub-optimal performance even with large differences in observed workloads: we achieve a 9% and a 1% improvement in accuracy under the supervised setting, for 20-driver/maximum kernel version difference, respectively, and a 21% and a 13% improvement under the unsupervised setting. Across different attacks, using optimal phases for each attack leads to a 10% improvement in average accuracy, compared to the baseline classifiers trained with the full-trace features directly. We demonstrate an ensemble classification scheme, which can be constructed with a reduced training cost (a 4.3X reduction). The classifier combines optimal phase classifiers of a limited, representative set of attacks. Our results show that the ensemble classifier improves average accuracy by 5% (to 77%) compared to the baseline full-trace classifiers.

2 ML-BASED DETECTION USING BOOT POWER: OVERVIEW

The basis for our methodology is the premise that deviation in boot process is reflected in the power signature, and that such change can be identified via an optimally constructed detection technique.

Boot process has unique characteristics compared to various run-time programs. Fig.1 shows an averaged power trace of Linux boot for kernel 4.15.0-45-generic with a default set of device drivers. We see that the power signature has *long duration compared to run-time workloads, large variance, and a non-periodic pattern.*

However, stealthy malware that tampers with the boot process can only leave its footprint at specific stages instead of the entire boot sequence. This means that the deviation in power signature due to the change of the boot process can only occur in limited highly localized time intervals. These characteristics of boot result in the classifiers, that are trained on the entire trace *non-discriminately* to perform poorly. This is because the power signature of other events, along the entire trace, can hide the critical signal of malware.

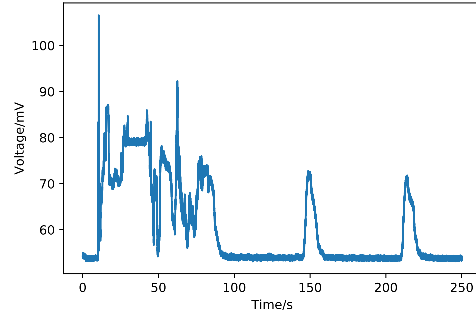


Figure 1: A typical power trace of Linux boot has long duration, large variance, and a non-periodic pattern.

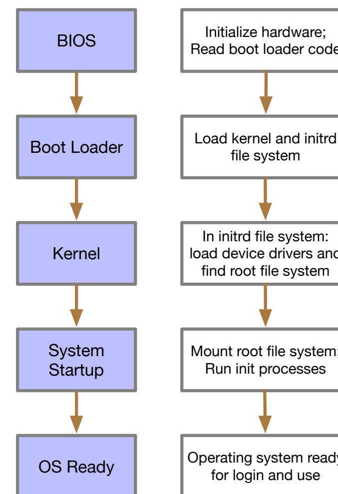


Figure 2: A high-level view of different phases and events in Linux boot.

To improve classification accuracy, we propose to focus on several optimally selected phases, that represent the largest deviations in power signature due to a change in boot. We explore different feature extraction techniques and identify the statistical features to be the optimal. We categorize the techniques to train the classifier into supervised and unsupervised approaches. The supervised approach involves the power signatures of both benign and malicious workloads in training data set, which requires prior knowledge of potential malware [5, 8, 16]. The approach trains the classifier with instances of all possible labels. The unsupervised approach *only* uses power signatures of benign workloads in the training set, aiming to achieve zero-day detection of potential malware [3, 5, 27]. In this case, the classifier is trained with instances of only a single class.

2.1 Identifying Critical Time Intervals

The first set of features we select are based on statistical information about the time series. Global mean, variance, and higher-order statistical moments have been shown to be effective for time series classification. We use the following features:

- Mean, variance, skew and kurtosis
- Root-mean-square (RMS)
- Maximum and minimum values
- Interquartile range (IQR)

The first 4 features, corresponding to the first to the fourth moment, capture the shape of the power distribution. The RMS captures the average of the sum of the squared power values. The max/min values take the largest/smallest power values in a given time interval, and the IQR calculates the difference between the 75% and 25% quantiles of the data, which reflects the spread of the power values.

We further explore features based on the Principal Component Analysis (PCA) and the Fast Fourier Transform (FFT). For the PCA-based features, we select the components that preserve over 90% of the training set's variance. For the FFT-based feature extraction, we first divide the $(0, f_s/2)$ frequency range equally into 8 intervals, where f_s stands for the sampling frequency. Then, we use the energy from each frequency interval of the power signal's spectrum as features.

We find the optimal time intervals by searching over a set of candidate intervals reflecting different phases of boot (Details presented in Section 3.3). We first partition the entire trace into multiple intervals, each corresponding to a different phase of the boot sequence. Afterwards, we do a finer adjustment, and partition the kernel load interval, which is the most critical point in boot, to obtain a more localized characterization of the stage. Finally, we train a classifier based on the features extracted from each interval, and select the time intervals yielding the best classification performance.

2.2 Linux Boot

The Linux boot process consists of 4 phases: the Basic Input Output File System (BIOS), the boot loader, the kernel, and the system startup, Fig. 2. The BIOS phase executes the firmware code to initialize hardware, and reads the boot loader code from the Master Boot Record (MBR), which is the first sector of the boot hard disk. Next, the boot loader code is executed and the kernel and initial ram disk (initrd) file system is loaded. The initrd file system of the kernel is a temporary file system in memory which constrains the size of the kernel and allows to load the necessary device drivers dynamically. The kernel phase executes the init script in the initrd file system which loads device drivers and locates the root file system. Finally, in the system startup phase, the real root file system is mounted and the init processes start to execute. These processes perform initialization and begin the startup events to bring up a fully functional operating system.

3 EXPERIMENTS

3.1 Threat Models

We emulate the untrusted devices threat model by loading unintended device drivers during boot. We pick 20 randomly-chosen pre-compiled device drivers supported by the operating system. We configure the system to load a varying number of said drivers in addition to the default drivers. The boot of the default set of device drivers is selected as baseline. Ubuntu 16.04.6 LTS is used as the operating system. The pre-compiled device drivers for a specific kernel version can be accessed under the directory `/lib/modules/<kernel_version>`. At boot time, the system will load the extra drivers listed in file `/etc/modules`, in addition to the default drivers. Table 2 shows all the device drivers used in the study.

Table 2: Extra device drivers and Linux kernel versions.

Extra Device Drivers	Kernel Versions
bfusb, bluecard_cs, btusb,	4.15.0-45-generic
clk_pwm, amdgpu, panel,	4.15.0-91-generic
horizon, pata_marvell, altera-msgdma,	4.15.1
mic_x100_dma, leds-regulator, dell_rbu,	4.16.0
apple-gmux, thunderbolt, phy-qcom-usb-hs	4.4.0-21-generic
hwmon-vid, max34440, gpio_charger,	
netconsole, thunderbolt-net	

Table 3: Kernel source code changes increase with kernel version difference. Baseline: kernel 4.15.0-45-generic.

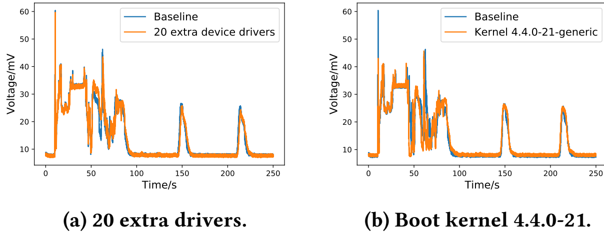
Kernel Version	4.15.0-91	4.15.1	4.16.0	4.4.0-21
Changed Lines	197960	126517	540217	3599691

We emulate the compromised kernel model by configuring the system to boot into a kernel version different from the baseline version. We install 5 different versions of Linux kernel and arbitrarily select one as the baseline. Table 2 shows the kernels used. Ubuntu OS uses GRUB2 as the boot loader. To configure the boot loader to automatically load a specific kernel image at boot time, we modify the GRUB_DEFAULT entry to the desired kernel version in file `/etc/default/grub`. Next, we run the `update-grub` command to update the boot loader configuration. The selected kernel version is loaded in subsequent boots.

While the boot sequence deviation in the untrusted devices threat model can be quantified by the number of unintended drivers, the measure of deviation in the compromised kernel model is not straightforward. Since the Linux kernel version number indicates a modification to current kernel source code, we assume that the boot sequence deviation is proportional to the kernel version difference. We run the `diff` command to quantify the number of lines of kernel source code that were changed, using kernel 4.15.0-45-generic as baseline, and summarize results in Table 3. In general, the number of changed lines increases with kernel version difference.

3.2 Experimental Setup

We use a Portwell PCOM-B700G Computer On Module (COM) housing a 8-core Intel Xeon D-1539 CPU running at 1.6 GHz with 16GB DDR4 RAM. This COM sits on a PCOM-C700G carrier board which provides power connectivity and additional peripheral interfaces. We take power measurements from the 12V power rail, which supplies the CPU and memory system. In order to obtain power measurements, we splice the power rail and instrument an Adafruit INA169 analog DC current sensor breakout board in series. The breakout board consists of a 0.1Ω shunt resistor and a current sense



(a) 20 extra drivers. (b) Boot kernel 4.4.0-21.

Figure 3: Power trace differences for two threat models. The largest deviation from baseline appears in kernel and system startup phases.

amplifier measuring voltage drop across the resistor. We capture amplifier output with a Picoscope 2408B oscilloscope.

We capture a single power trace for each run of Linux boot at a 2KS/s sampling rate. The data buffer is configured to store 250 seconds of power samples. We note that the actual boot sequence takes less than 250 seconds to finish. For example, with kernel 4.15.0-45-generic the time between launching the `reboot` command and displaying the login screen is about 65 seconds, corresponding to time interval 10s to 75s in Fig. 1. We configure a larger data buffer size in order to capture the power signature of more system startup events and guarantee one entire boot process finishes within the 250-second interval, since we observe the system will halt for a while for some runs of boot after launching `reboot` command. The halt behavior appears randomly in the boot runs and power traces of such runs are discarded in data analysis.

3.3 Extracting Features from Selected Time Interval

We now describe our methodology for finding time intervals with the largest power signature deviation produced by boot modification. For illustration, we consider the cases producing the largest deviation to baseline. In the untrusted device model, 20 extra drivers are loaded (we call it TM1-20-driver case). In the compromised kernel model, the kernel with the largest difference, kernel 4.4.0-21-generic is loaded (we call it TM2-kernel-4 case). Each training set consists of approximately 200 boot power traces, and each test set consists of approximately 90 boot power traces. Both data sets have similar number of benign and malicious boot runs. Fig. 3 shows the averaged power traces of benign and malicious boot runs for both TM1-20-driver case and TM2-kernel-4 case. We observe that in both cases the largest divergence of power signatures occurs in a small time range within the kernel and system startup phases. *This supports our hypothesis that the boot attacks leave a signature only in a limited time interval out of the long signal.*

We first pre-process each power trace by removing the global minimum, i.e. background power. Next, we apply a moving average filter with a window size of 500 samples to reduce noise and smooth the time series.

We then reduce the search space to a set of candidate time intervals based on actual boot events. Optimal features are found via an extensive grid search on the interval set.

Table 4: Performance of models trained with features from different time intervals in TM1-20-driver case (Accuracy/ROC-AUC). Optimal phase features are significantly better than full trace or sub-optimal phase features. Optimal interval: (40, 60), (60, 90).

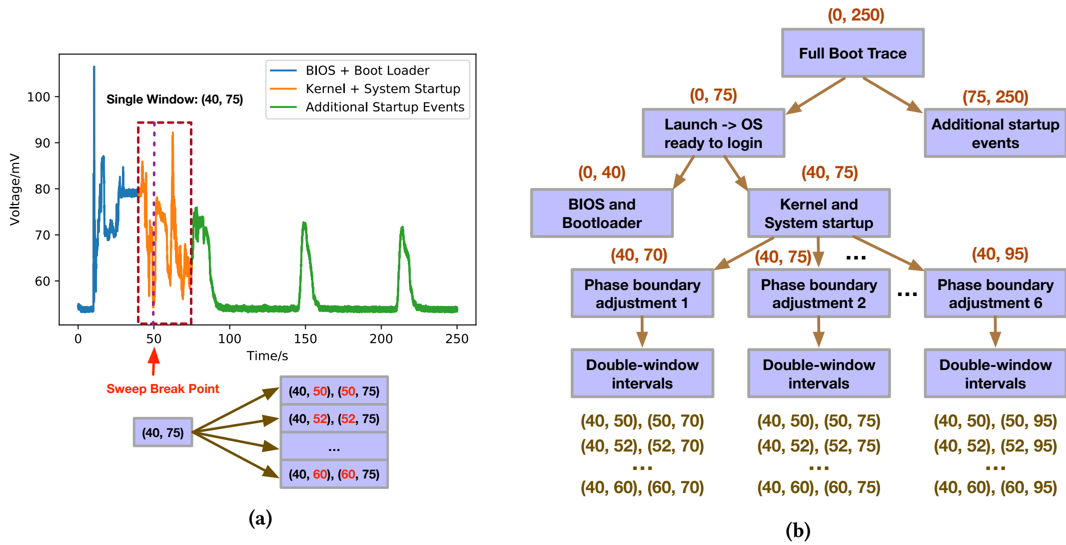
Interval	(0, 250)	(0, 75)	1-window avg.	2-window avg.	**Optimal**
SVM	0.73/0.76	0.74/0.74	0.75/0.69	0.82/0.89	**0.86/0.95**
ocSVM	0.61/0.54	0.73/0.75	0.63/0.66	0.64/0.66	**0.82/0.92**
Avg.	0.67/0.65	0.73/0.75	0.69/0.68	0.73/0.78	**0.84/0.94**

Table 5: Performance of models trained with features from different time intervals in TM2-kernel-4 case (Accuracy/ROC-AUC). Optimal phase features are significantly better than full trace or sub-optimal phase features. Optimal interval: (40, 52), (52, 75 or 90 or 95).

Interval	(0, 250)	(0, 75)	1-window avg.	2-window avg.	**Optimal**
SVM	0.99/1.00	0.97/0.98	0.95/0.98	1.00/1.00	**1.00/1.00**
ocSVM	0.87/0.99	0.72/0.74	0.78/0.87	0.90/0.95	**1.00/1.00**
Avg.	0.93/0.99	0.84/0.86	0.87/0.93	0.95/0.98	**1.00/1.00**

We use the said optimal features to train several ML classifiers. For a supervised setting, we perform experiments with the Support Vector Machine (SVM), Logistic Regression (LR), and Decision Tree (DT). For each trace from a test set, we perform the above pre-processing steps. We adopt one-class SVM (ocSVM) for the unsupervised learning setting. To train ocSVM, we remove traces of the malicious runs from the training sets of both TM1-20-driver and TM2-kernel-4 cases.

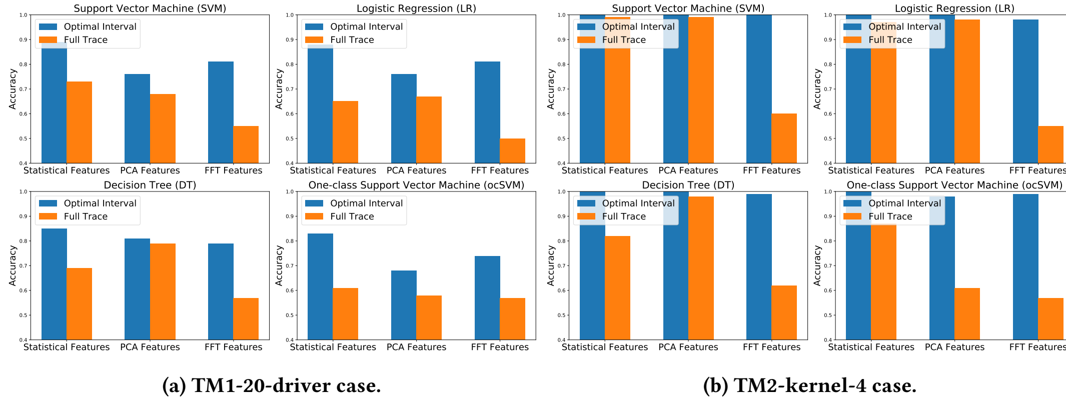
We now describe how to generate the set of candidate time intervals, noting that a similar process can be applied to different systems, since the boot process executes a well-defined sequence of tasks independent of a specific deployment. We start by partitioning the entire time window into two time intervals: from launching to when the OS is ready for login, (0s, 75s), and the rest of system startup events, (75s, 250s). We further partition the launch to login window (0s, 75s) into two intervals: one corresponding to BIOS and boot loader phases, and the other corresponding to kernel and system startup phases. As the single-window time interval provides a nice localized view of the boot trace, we explore finer-grained localization. We split the single-window interval of kernel and system startup phase into two concatenated time intervals, and extract features individually from each. The main intuition comes from the fact that the two phases involve different boot events, such as loading device drivers and mounting root file system, and the observation that the power samples corresponding to the two phases have a large variation. Evaluating two phases as a whole cannot precisely capture tiny malware signature within each individual phase, due to the large cross-phase variance. By concatenating features from each window of the double-window interval, we can better monitor the activities within the phases and improve classifier sensitivity



(a)

(b)

Figure 4: (a) A double-window interval generation. A single (40s, 75s) interval is used to generate 6 double-window intervals by sweeping the boundary. (b) Generation of candidate time intervals.



(a) TM1-20-driver case.

(b) TM2-kernel-4 case.

Figure 5: Performance of models trained with different ML pipeline configurations in TM1-20-driver case and TM2-kernel-4 case.

to malicious samples. Specifically, we split the (40s, 75s) time interval. Since there is no clearly defined break point between the two phases, we sweep the location of break point starting from 50s until 60s with a step of 2s. This results in 6 double-window time intervals, Fig. 4a. Finally, we adjust the upper bound of the (40s, 75s) time interval to control the amount of information regarding system startup events. We sweep the upper bound from 70s until 95s with a step of 5s. For each resulting single-window time interval, we repeat the above process to generate multiple double-window time intervals. We finally obtain a set with 46 candidate time intervals (including single-window intervals, double-window intervals and full trace). Fig. 4b illustrates the described candidate time interval generation process.

For each candidate time interval, we train SVM, LR, DT, and ocSVM. We extract the statistical features, PCA features, and FFT features, described in Section II, from each single-window interval and the doubled number of features from each double-window interval (concatenated by the features of each window). The implementation of the classifiers is based on the Python scikit-learn package. For SVM and ocSVM, we use different kernels (Linear, RBF, and Poly) and perform a grid search on the remaining tunable parameters (regularization, kernel coefficient, and polynomial degree). For LR, we vary the solver type, regularization strength and stopping criteria. For DT, we vary the splitter type, split quality measure, and the parameters to control the regularization strength. The process results in 54 SVM models, 150 LR models, 80 DT models and 60 ocSVM models for each candidate time interval. We show the performance of the optimal models in Fig. 5a and Fig. 5b.

As shown in Fig. 5a and Fig. 5b, the models trained with the strategically-chosen phase features are better than the models trained with the full-trace features, regardless of the feature selection and the classifier used. Using statistical features with SVM/LR is optimal, with a near 90% accuracy for the TM1-20-driver case. The models trained with the FFT features extracted from the full

trace show the worst performance. We anticipate this is due to the non-periodic nature of the boot sequence. Since the optimal SVM model and the optimal LR model have close accuracy, we keep the ML pipeline, which adopts statistical features and SVM (ocSVM for the unsupervised setting), for all remaining experiments. Table 4 and 5 elaborates the performance of SVM and ocSVM models trained with features from different time intervals. We report the interval with the highest average accuracy and ROC-AUC score over the SVM model and ocSVM model as the optimal time interval.

Beyond the superior accuracy against the full trace models, we also observe in Table 4 and Table 5 that models trained with optimal time intervals show a higher accuracy compared to models trained on the (0s, 75s) time interval, which corresponds to the duration of the actual boot sequence from launching to when the OS is ready for login. *This confirms that the localized extraction of features improves model performance.* In addition, we observe that models trained with features from single-window time intervals are sub-optimal compared to double-window time intervals in general. The identified optimal phases for both TM1-20-driver and TM2-kernel-4 cases are double-window time intervals. The superior performance is not simply a result of the increased number of features. The main contribution comes from the further partition of the kernel and system startup phase of boot sequence. We observe that the power signature of kernel and system startup phases itself has a large variation across time since power-consuming system events such as disk access and mounting root file system are involved in the phases. These events dominate the statistics of power samples, and hide the small change in power caused by malware executing in limited locations. In contrast, we can reduce the effect of those power-dominant events by having a more localized view of power sample statistics achieved by the further partition to kernel and system startup phases. The features extracted from double-window time interval are more sensitive to the boot sequence changes resulting in a higher classification accuracy.

The optimal time intervals of TM1-20-driver case and TM2-kernel-4 case have different break points for the kernel phase and system startup phase. The TM1-20-driver case has a longer interval for kernel phase. This is due to the event of loading device drivers which happens in later kernel phase is more critical to the TM1-20-driver case, while the TM2-kernel-4 case is more sensitive to the start of the kernel phase where the initrd file system is being loaded. The initrd file system is kernel specific. The system startup events after OS is ready to login also help to distinguish malicious cases. The optimal time intervals for both cases include system startup information beyond 75s. We can further observe that SVM models out-perform ocSVM models, as prior knowledge of malicious instances is included in the training process.

3.4 Comparison with Algorithms in Related Work

We compare our approach, where features are extracted from optimally selected phases, to other approaches demonstrated in literature, where the full trace is considered non-discriminately. To represent supervised approaches, we implement the algorithms of [8], [5] and [3]. The algorithm in [8] first splits an entire trace into 5-second segments. Each segment corresponds to a single training instance, and its label is determined by whether the workload is benign or malicious. Features are extracted from both time and frequency domains. Random Forest and Neural Networks are used for classification. The algorithm in [5] creates segments using a sliding window over the full trace. Features, extracted from each segment, are fed into a Decision Tree classifier. In [3], sophisticated features of the time series, such as permutation entropy and data-smashing distances, are utilized besides statistical moments. An SVM is used for classification. We apply the same data pre-processing steps (removing background power and applying a moving average filter) on traces from the training set of both TM1-20-driver and TM2-kernel-4 cases, train the algorithms, and report their accuracy in Table 6.

Table 6: Supervised learning approaches: accuracy

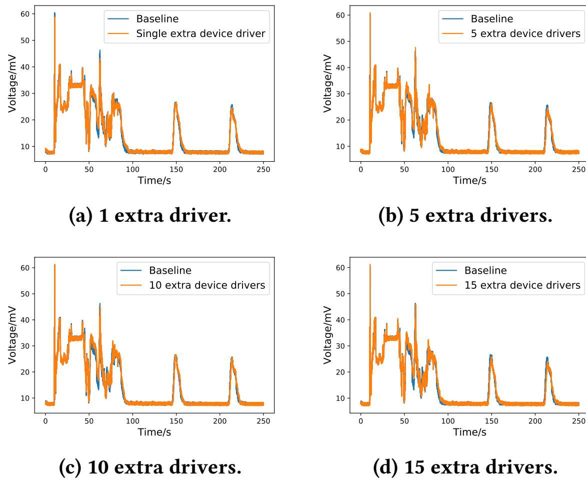
Algorithm	TM1-20-driver	TM2-kernel-4
Statistics of optimal interval + SVM [This work]	0.86	1.00
Statistics of each segment + Random Forest [8]	0.72	0.72
Statistics of each segment + Neural Net [8]	0.68	0.69
Statistics of each segment + Decision Tree [5]	0.67	0.57
Statistical features of full trace + SVM [3]	0.77	0.99

Similarly, for unsupervised approaches, we implement the algorithms from [27], [3] and [5]. These techniques assume no prior knowledge of malware during training phase. They are typically based on one-class SVMs or regression. In [27], the full trace is first broken into segments, and a Discrete Wavelet Transform (DWT) is performed on each segment. In the bag-of-words technique [26], each transformed segment of the full trace is first converted to a code-word, and the histogram of the code-words is used as a training feature. In the LSTM regression technique, a vanilla LSTM regressor is trained using the transformed segments. The regressor predicts the last value of each segment using the previous values of the segment. During inference, if the difference between the actual and predicted power is greater than a threshold, the workload is classified as an anomaly. The techniques in [3] construct an ensemble of classifiers with a single feature. Each classifier is based on the mean and standard deviation of the corresponding feature from the benign workloads. The majority voting is used when classifying a workload. Finally, we evaluate Decision Tree based regression on each power trace segment [5]. We follow the same steps as described in section 3.3 to pre-process data. Accuracy of models used in prior works are shown in Table 7.

As shown in Table 6 and Table 7, both the classification-based and regression-based detection techniques demonstrated in related work show sub-optimal performance on boot. *The main reason is that these algorithms consider the full power trace non-discriminately.* While these algorithms can achieve excellent performance on regular run-time workloads, which have periodic patterns and a relatively short duration, they are not suitable for boot where the largest deviation in signature due to tampering occurs only in a few time intervals.

Table 7: Unsupervised learning approaches: accuracy

Algorithm	TM1-20-driver	TM2-kernel-4
Statistics of optimal interval + ocSVM [This work]	0.82	1.00
DWT + bag-of-words + ocSVM [27]	0.61	0.87
DWT + vanilla LSTM [27]	0.50	0.53
Statistical features of full trace + single feature ensemble [3]	0.55	0.47
Decision Tree Regression on each segment [5]	0.46	0.52



(a) 1 extra driver. (b) 5 extra drivers. (c) 10 extra drivers. (d) 15 extra drivers.

Figure 6: Averaged power traces of benign and malicious boot runs for untrusted devices cases. The power signature difference increases with the number of extra device drivers to load.

3.5 Operating Range: Untrusted Devices

We explore the operating range of power-based malware detection for attacks in which a smaller number of extra device drivers is loaded during boot. Specifically, we investigate the use of 1, 5, 10 and 15 extra device drivers. We load an increasing number of device drivers following the order in Table 2, to create an increasing amount of deviation from baseline. For the single extra device driver case, we collect power traces where one of the bfusb, bluecard_cs, btusb, clk_pwm and amdgpu drivers (the first 5 drivers in Table 2) is loaded during the boot process. A similar number of power traces is collected for each driver. The data set construction for each case follows similar specifications as TM1-20-driver case (Section 3.3). Fig. 6 shows the averaged power traces of benign and malicious boot runs.

Fig. 6 shows that power signature deviation from baseline increases as the number of extra device drivers to load increases. As before, the largest deviation in power signature occurs in the kernel and system startup phases. We further evaluate the achievable detection performance, using the same data pre-processing and optimal time search as in the TM1-20-driver case. The optimal time

Table 8: Model performance of untrusted devices cases (Accuracy/ROC-AUC).

Driver Amount		1 driver	5 drivers	10 drivers	15 drivers
SVM	Optimal	0.72/0.75	0.80/0.85	0.76/0.82	0.85/0.90
	(0, 250)	0.66/0.64	0.67/0.66	0.68/0.71	0.73/0.76
	(0, 75)	0.69/0.70	0.65/0.62	0.73/0.72	0.70/0.75
ocSVM	Optimal	0.67/0.67	0.75/0.84	0.76/0.83	0.85/0.92
	(0, 250)	0.56/0.54	0.61/0.57	0.57/0.62	0.59/0.56
	(0, 75)	0.67/0.70	0.69/0.65	0.64/0.62	0.67/0.66

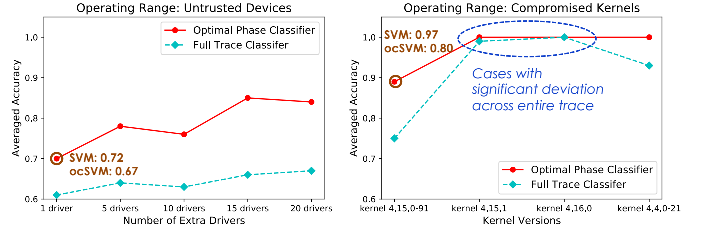


Figure 7: Averaged accuracy over SVM and ocSVM models for different attacks. Strategically-chosen features perform better than full-trace features overall.

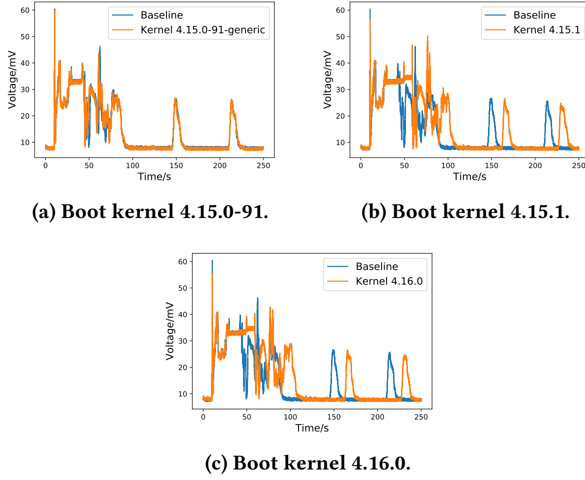
interval is selected via the average accuracy and ROC-AUC score over the SVM and ocSVM. We summarize the results in Table 8 and plot the operating range in Fig. 7.

Overall, the classifier performance increases as the number of extra drivers increases. The classifier with 1 extra driver achieves a 72% accuracy in supervised training, and 67% accuracy in unsupervised training. The classifier with 15 extra drivers achieves a 85% accuracy in both approaches. This is within expectation since different amount of change is introduced to boot. Models trained with optimal features out-perform models trained with a full power trace, Fig. 7. The run-to-run variance of noisy boot power trace causes some instances to be placed at incorrect sides of decision boundary. This leads to non-perfect detection accuracy, and possible evasion of attacks which only leave small signature in boot trace. We expect a finer measurement setup will boost accuracy and will explore the EM-based malware detection in future work.

3.6 Operating Range: Compromised Kernel

Now we continue exploring operating range for the compromised kernel threat model. We investigate cases where kernel 4.15.0-91-generic, 4.15.1 and 4.16.0 are loaded during the boot process. Kernels with smaller version difference to the baseline are expected to show smaller deviation in boot sequence. The data set construction is similar as before. We plot the averaged benign and malicious boot power traces of the various compromised kernel cases in Fig. 8.

We observe that the signature deviations of kernel 4.15.1 and kernel 4.16.0 from baseline are significantly greater than that of 4.15.0-91-generic and even 4.4.0-21-generic demonstrated in Fig. 3b. It takes longer for the boot loader to load kernel 4.15.1 and 4.16.0. However, among all kernel versions that we investigated, kernel 4.4.0-21-generic has the largest version difference to baseline. Although we observed that kernel code line changes increase with kernel version difference in Table 3, it does not guarantee that



(a) Boot kernel 4.15.0-91. **(b) Boot kernel 4.15.1.**

(c) Boot kernel 4.16.0.

Figure 8: Averaged power traces of benign and malicious boot runs for compromised kernel cases. The power signature difference is kernel-specific.

signature deviation increases accordingly. Some specific changes in code can cause larger deviation in boot behavior, such as kernel 4.15.1 and 4.16.0. Therefore, we cannot draw a clear conclusion that signature deviation from baseline is proportional to kernel version difference. We follow the same procedures as before to train the model and select optimal time interval, and show the results in Table 9 and plot the operating range in Fig. 7.

The optimal models achieve near perfect accuracy across the compromised kernel cases, Table 9. The classifiers trained with features from both selected time intervals or full power traces achieve perfect or near perfect detection accuracy in kernel 4.15.1 and kernel 4.16.0 cases. This is because the significant power signature deviation from baseline over the entire time window. However, the kernel 4.15.0-91-generic case exhibits similar behavior as the previous cases: the models trained with optimal phase features show significantly higher accuracy than the full trace features or sub-optimal phase features. The reason is that the power signature deviation only occurs in limited phases and a localized feature extraction increases SNR of the critical signal.

4 UNIFIED-WINDOW CLASSIFIER

While classifiers based on optimal features of each unique attack have the highest detection accuracy, the complexity of training increases with the number of attacks. In this section, we evaluate the performance of a classifier which trains on a limited set of attacks, identifies corresponding optimal intervals, and then adopts features from the same time intervals across different attacks.

To monitor the entire boot sequence and leverage the advantage of localized feature extraction, we train classifiers at optimal phases of the attacks as well as the remaining phases of boot. The classifier ensemble predicts anomaly if *any* phase predicts anomaly. Specifically, we train 4 ocSVMs: one based on optimal phase of TM1-20-driver case, one based on optimal phase of TM2-kernel-4 case, one based on BIOS and boot loader phase, and the last one based on additional system startup events. The final prediction

Table 9: Model performance of compromised kernel cases (Accuracy/ROC-AUC).

Kernel Version		4.15.0-91-generic	4.15.1	4.16.0
SVM	Optimal	0.97/0.99	1.00/1.00	1.00/1.00
	(0, 250)	0.78/0.86	0.99/1.00	1.00/1.00
	(0, 75)	0.80/0.84	1.00/1.00	1.00/1.00
ocSVM	Optimal	0.80/0.86	1.00/1.00	1.00/1.00
	(0, 250)	0.71/0.72	0.99/1.00	0.99/1.00
	(0, 75)	0.74/0.74	0.99/1.00	0.99/1.00

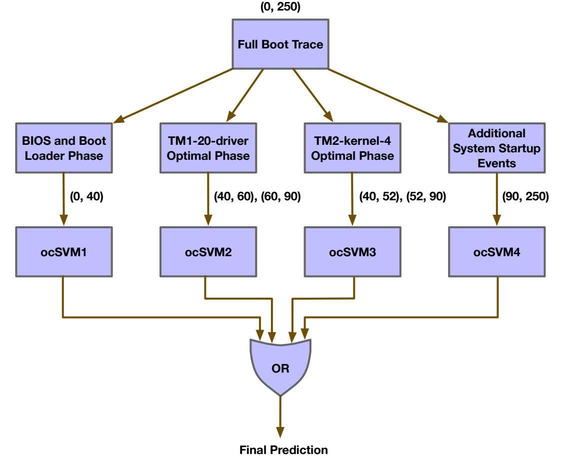


Figure 9: The construction of the classifier ensemble from classifiers based on different boot phase.

is given by a logical OR of predictions from each classifier. Fig. 9 illustrates the classifier construction.

We train the classifier ensemble with TM1-20-driver and TM2-kernel-4 cases. We first perform a grid-search for optimal phases, using the same procedure as in Section III-C. The optimal models deployed in the classifier ensemble are ocSVM2 and ocSVM3, Fig. 9. Next, we search for hyper-parameters of the other ocSVMs to obtain optimal accuracy. The same ensemble is adopted for other attacks. We note that the classifier ensemble requires smaller training effort compared to an earlier method, which determines optimal phase and model for each attack. It only trains on two attacks (malicious cases). For example, searching the optimal phases of the 9 untrusted device and compromised kernel attacks requires training on $9 \times 46 = 414$ time intervals. In contrast, the classifier ensemble only trains on $2 \times (46 + 2) = 96$ time intervals. This leads to a 4.3X reduction in training effort.

Since we aim to evaluate a unified classifier across different attacks, the baseline classifier is selected to be a fixed ocSVM model, trained with features extracted from the full boot trace. Its hyper-parameter is selected such that the optimal accuracy is achieved for both TM1-20-driver and TM2-kernel-4 cases. We report the performance of both the classifier ensemble and the baseline classifier across all untrusted devices and compromised kernel attacks in Table 10.

We observe that the classifier ensemble has worse performance compared to the optimal classifier trained for each attack. It has a 8% drop in averaged accuracy. The degradation in performance is

Table 10: Model performance of the unified classifier and optimal classifiers: accuracy.

Malicious cases	Classifier ensemble	Full-trace classifier	Optimal classifier
1-driver difference	0.67	0.56	0.67
5-driver difference	0.74	0.53	0.75
10-driver difference	0.76	0.52	0.76
15-driver difference	0.77	0.53	0.85
20-driver difference	0.80	0.56	0.82
Kernel 4.15.0-91-generic	0.64	0.71	0.80
Kernel 4.15.1	0.85	0.36	1.00
Kernel 4.16.0	0.85	0.36	1.00
Kernel 4.4.0-21-generic	0.89	0.85	1.00
Average	**0.77**	**0.55**	**0.85**

due to non-optimal selection of features, and false positives from the classifiers of other non-optimal phases. We also observe that the classifier ensemble still out-performs the baseline full-trace classifier in most cases. This confirms the advantage of the localized feature extraction for Linux boot. Although the optimal time intervals for the untrusted devices threat model, and those for the compromised kernel threat model are different, we can combine the optimal intervals from both classes to capture the anomaly signature for both and improve the generality of the deployment.

5 CONCLUSION

In this work, we develop the first power-based malware detection technique for Linux boot. We demonstrate the technique's effectiveness on an embedded system with an Intel Xeon-class CPU. Conventional approaches are not effective for the boot sequence because they consider a full power trace non-discriminately. In a boot power trace, security-relevant information is contained in a limited time interval of the full boot duration. We propose to extract statistical features from the strategically-chosen phases of a trace to improve classification accuracy. We investigate two threats: an untrusted device and a compromised kernel. We evaluate a supervised approach, requiring prior knowledge of malware, and an unsupervised approach, capable of zero-day detection. Identifying optimal phases for each attack leads to a 10% improvement in average accuracy, compared to the baseline classifiers trained with full-trace features directly. We demonstrate an ensemble classification scheme which can be constructed with a reduced training cost (a 4.3X reduction). The classifier combines optimal-phase classifiers of a limited, representative set of attacks. Our results show that the ensemble classifier improves average accuracy by 5% (to 77%) compared to the baseline full-trace classifiers.

REFERENCES

[1] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh. 2013. A survey on heuristic malware detection techniques. In *The 5th Conference on Information and Knowledge Technology*. 113–120.

[2] Martijn Bogaard. 2019, accessed (June 27, 2019). Secure Boot Under Attack: Simulation to Enhance Fault Attacks & Defenses. https://www.riscure.com/publication/secure-boot-under-attack-simulation-to-enhance-fault-attacks-defenses/.

[3] R. Bridges, J. Hernández Jiménez, J. Nichols, K. Goseva-Popstojanova, and S. Prowell. 2018. Towards Malware Detection via CPU Power Consumption: Data Collection Design and Analytics. In *TrustCom/BigDataSE*. 1680–1684.

[4] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. 2011. Crowdroid: behavior-based malware detection system for Android. In *SPSM*. 15–26.

[5] L. Caviglione, M. Gaggero, J. Lalande, W. Mazurczyk, and M. Urbański. 2016. Seeing the Unseen: Revealing Mobile Malware Hidden Communications via Energy Consumption and Artificial Intelligence. *IEEE Transactions on Information Forensics and Security* 11, 4 (2016), 799–810.

[6] Haogang Chen, Yandong Mao, Xi Wang, Dong Zhou, Nickolai Zeldovich, and M. Frans Kaashoek. 2011. Linux Kernel Vulnerabilities: State-of-the-Art Defenses and Open Problems. In *APSys*.

[7] Catalin Cimpanu. 2020, accessed (July 29, 2020). 'BootHole' attack impacts Windows and Linux systems using GRUB2 and Secure Boot. https://www.zdnet.com/article/boothole-attack-impacts-windows-and-linux-systems-using-grub2-and-secure-boot/.

[8] Shane S. Clark, Benjamin Ransford, Amir Rahmati, Shane Guineau, Jacob Sorber, Wenyuan Xu, and Kevin Fu. 2013. WattsUpDoc: Power Side Channels to Nonintrusively Discover Untargeted Malware on Embedded Medical Devices. In *USENIX HealthTech*.

[9] B. Dixon and S. Mishra. 2013. Power Based Malicious Code Detection Techniques for Smartphones. In *TrustCom*. 142–149.

[10] T. C. Group. 2012. TCG PC Client Specific Implementation Specification for Conventional BIOS.

[11] 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 *CCS*. 1095–1108.

[12] Trammell Hudson and Larry Rudolph. 2015. Thunderstrike: EFI Firmware Bootkits for Apple MacBooks. In *SYSTOR*.

[13] Nisha Jacob, Johann Heyszl, Andreas Zankl, Carsten Rolfes, and Georg Sigl. 2017. How to Break Secure Boot on FPGA SoCs Through Malicious Hardware. In *CHES*, Vol. 10529. 425–442.

[14] R. Langner. 2011. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Security & Privacy* 9, 3 (2011), 49–51.

[15] X. Li, Y. Wen, M. H. Huang, and Q. Liu. 2011. An Overview of Bootkit Attacking Approaches. In *MSN*. 428–431.

[16] Lei Liu, Guanhua Yan, Xinwen Zhang, and Songqing Chen. 2009. VirusMeter: Preventing Your Cellphone from Spies. In *RAID*, Engin Kirda, Somesh Jha, and Davide Balzarotti (Eds.). 244–264.

[17] Yannan Liu, Lingxiao Wei, Zhe Zhou, Kehuan Zhang, Wenyuan Xu, and Qiang Xu. 2016. On Code Execution Tracking via Power Side-Channel. In *CCS*. 1019–1031.

[18] Igor Nai Fovino, Andrea Carcano, Marcelo Masera, and Alberto Trombetta. 2009. An experimental investigation of malware attacks on SCADA systems. *International Journal of Critical Infrastructure Protection* 2, 4 (2009), 139 – 145.

[19] Jungmin Park, Xiaolin Xu, Yier Jin, Domenic Forte, and Mark Tehranipoor. 2018. Power-based Side-Channel Instruction-level Disassembler. In *DAC*. 1–6.

[20] G. Pocklassery, W. Che, F. Saqib, M. Areno, and J. Plusquellic. 2018. Self-authenticating secure boot for FPGAs. In *HOST*. 221–226.

[21] V. Rastogi, Y. Chen, and X. Jiang. 2014. Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks. *IEEE Transactions on Information Forensics and Security* 9, 1 (2014), 99–108.

[22] Brian E. Richardson. 2013. Uefi Secure Boot in Modern Computer Security Solutions.

[23] Gaurav Shah, Andres Molina, and Matt Blaze. 2006. Keyboards and Covert Channels. In *USENIX Security*.

[24] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. 2014. Unsupervised Anomaly-based Malware Detection using Hardware Features. arXiv:1403.1631 [cs.CR]

[25] USBKILL. 2022. USBKILL V4.0. https://www.usbkill.com/.

[26] Jin Wang, Ping Liu, Mary F.H. She, Saeid Nahavandi, and Abbas Kouzani. 2013. Bag-of-words representation for biomedical time series classification. *Biomedical Signal Processing and Control* 8 (2013), 634–644.

[27] S. Wei, A. Aysu, M. Orshansky, A. Gerstlauer, and M. Tiwari. 2019. Using Power-Anomalies to Counter Evasive Micro-Architectural Attacks in Embedded Systems. In *HOST*. 111–120.

[28] Toshihiro Yamauchi, Yohei Akao, Ryota Yoshitani, Yuichi Nakamura, and Masaki Hashimoto. 2020. Additional kernel observer: privilege escalation attack prevention mechanism focusing on system call privilege changes. *International Journal of Information Security* (2020).

[29] Baki Berkay Yilamz, Elvan Mert Ugurlu, Alenka Zajic, and Milos Prvulovic. 2019. Instruction level program tracking using electromagnetic emanations. In *Cyber Sensing 2019*, Vol. 11011. 56 – 67.

[30] Z. Zhang, Z. Zhan, D. Balasubramanian, B. Li, P. Volgyesi, and X. Koutsoukos. 2020. Leveraging EM Side-Channel Information to Detect Rowhammer Attacks. In *S&P*. 729–746.