Malware Anomaly Detection on Virtual Assistants

Ni An*, Alexander Duff†, Mahshid Noorani†, Steven Weber*, Spiros Mancoridis†
*Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA
†Department of Computer Science, Drexel University, Philadelphia, PA

***Abstract*—This work explores the application of anomaly detection techniques, specifically one-class support vector machine (SVM) and online change-point detection, to construct a model that can distinguish, in real-time, between the normal operation of an Amazon Alexa Virtual Assistant IoT device from anomalous operation due to malware infections. Despite the current absence of widespread malware for IoT devices, the anticipated rapid growth in deployment and use of IoT devices will likely attract many different malware attacks in the near future. Because of their highly specialized and, hence, predictable expected behavior, malware detection on IoT devices is not difficult given large training sets, long testing vectors, and extensive computational power. The challenge we address in this paper is to ascertain how quickly malware may be detected, *i.e.,* the distribution on the number of system calls before a suitably high confidence decision may be made.**

I. Introduction

The Internet of Things (IoT) refers to the growing network of "smart objects," *i.e.,* computer systems embedded into everyday things to give them sensing and actuating capabilities in order to perform specific functions and share data. The increased convenience and efficiency provided by IoT devices have made them commonplace in home environments. Some of the most popular home IoT devices include smart personal assistants (*e.g.,* Amazon Alexa) and smart thermostats (*e.g.,* Nest). The rapid and widespread adoption of IoT devices by the public has led to the rise of new security concerns, as several of these newly computerized devices are now, and many more will likely be in the near future, targets for malware.

Recently, malicious actors have used botnets of malware-infected IoT devices, such as Internet-connected appliances and home routers, to great effect [1]. These devices are attractive targets for malware due to their lack of cryptographic encryption and weak default authentication. Among attacks that have exploited IoT devices is a botnet malware family named Mirai [2]. The botnets in these attacks primarily consisted of home devices such as home routers, webcams, and printers [3]–[5].

Amazon Alexa Echo is an example of a popular IoT device. This intelligent virtual assistant processes natural language to assist people with tasks around their house. Alexa, similar to most IoT devices, is designed to accomplish simple and specialized tasks, and as such we claim that this narrow spectrum of responsibilities, which is common to IoT devices in general, yields a higher chance of success using anomaly-based detection methods to identify deviations in measured statistics against a normal model of operation of Alexa.

Behavioral malware-detection techniques to protect general-purpose computing platforms typically require a large corpus of pre-existing behavioral malware signatures for training purposes. However, the variety and youth of IoT devices leave us with a corpus of malware that is insufficient to employ classic machine learning algorithms. This makes anomaly detection methods for IoT device security more attractive, especially in the short term, until there are enough behavioral signatures for malware to train more sophisticated machine learning detection models for these devices.

In this paper, we exercise the capabilities of Amazon Alexa and use OS system call data along with one-class Support Vector Machines (SVM) [6] and sequential analysis technique to construct a real-time anomaly detector for Alexa. This anomaly detector is trained using only benign (normal) usage scenarios. The detector is subsequently subjected to both regular usage scenarios and malware infections in order to demonstrate its effectiveness to distinguish normal Alexa operation from anomalous, perhaps malicious, operation in real time.

While we demonstrate the effectiveness of our technique on Amazon Alexa, we anticipate that small variations of our technique can be used to create anomaly detectors for more IoT devices, because these devices are typically highly specialized and, hence, exhibit a predictable normal behavior. This is in contrast to creating an anomaly detector for a general purpose computer.

II. Previous work

There has been prior work on techniques to detect failing software servers in real-time due to software faults [7], such as memory leaks, infinite loops, and

logic errors. This work was later generalized to develop techniques that detect failing servers due to malware infections [8]–[14]. These malware detection techniques were later complemented by malware classification techniques that could classify the detected malware attacking servers into corresponding malware types (*e.g.,* Zbot) and families (*e.g,* Trojan botnet) [15], [16]. The preliminary state-of-the-art on malware detection and classification has been evaluated using thousands of malware samples infecting server platforms that were executing a diverse set of realistic workload scenarios [16]. Both classes of techniques, *i.e.,* malware detection and classification, relied on OS kernel-level system call traces as inputs to a variety of machine learning algorithms.

In the prior art data that was tainted by malware executions was used as part of the model training process. In contrast, this paper focuses on anomaly-based malware detection because of the limitations of traditional signature-based detectors (*e.g.,* ineffective on zero-day attacks) and the fact that current IoT device do not yet have enough malware samples that can be used to build a model. Therefore, our focus is on developing detection techniques that do not require *a priori* information about specific threats, but rather are designed to detect unusual behavior from the IoT device.

We will describe the creation of a behavioral anomaly detection system designed to detect the execution of malware on an Amazon Alexa IoT device, including new IoT malware variants and families such as Mirai [2]. The system has been designed to be used on-line on the actual IoT device, using behavioral features and detection algorithms to provide rapid malware detection, but that also take into consideration the limited processing and memory capability of IoT devices.

III. Anomaly detection model

To the best of our knowledge, currently, there are no malware samples, other than proof-of-concept ones, that target voice-controlled IoT devices that use Amazon Alexa. The lack of malicious training samples impedes researchers carrying out supervised malware detection techniques. Under the assumption that not enough malware samples are available, but will be eventually, anomaly detection techniques are of great interest for protecting IoT devices in the short term. Additionally, the rapid variation and development of malware in general, namely as polymorphic and metamorphic malware [17], highlights a unique advantage of anomaly detection over supervised approaches.

This work employs one-class Support Vector Machines (one-SVM) for anomaly detection along with the Cumulative Sum (CUSUM) [18] or the Shiryaev-Roberts (SR) procedure [19] for sequential analysis. This section begins by describing our data extraction and preprocessing methods in §III-A, and then proceeds with an overview of the anomaly detector.

A. Feature generation and preprocessing

Behavioral anomaly detection uses sequences of system calls and extracts features from them for the anomaly detector. Specifically, our detector relies on the traces of Linux OS kernel-level system calls that are generated by the processes running on the Amazon Alexa device being monitored. These system calls describe what OS functions are used by each process executing on a computing platform. When a malware process starts executing, its system call traces will likely not match any of the benign traces that are currently executing or have executed in the past.

System calls can be viewed as the language of machines, and feature extraction methods in natural language processing can be used to pre-process the system call traces. First, we take a sequence of systems calls collected in an observation window of length L as a data sample. Then, we employ the bag-of-n-grams model [20], [21] to produce the feature vector $\mathbf{x} \in \mathbb{R}^p$, which is a vector of the number of the system call sequences that occurred in a small sliding window of length L data samples. Term frequency-inverse document frequency (TF-IDF) transformation [20], [21] is then used to normalize the feature vectors before providing them as input to the one-class SVM.

B. One-class SVM-based anomaly detection algorithm

The one-class SVM finds a hyperplane that separates the training data from the origin with a margin that is as large as possible. The one-class SVM's objective function minimizes the normalized weights vector $\mathbf{w}$ of the hyperplane (which is equivalent to maximizing the margin), and the objective function is also penalized by points that lie on the wrong side of the hyperplane (*i.e.,* the side wherein the origin lies). The test statistic of the one-class SVM is the distance to the separating hyperplane: $y = \mathbf{w}^\intercal \phi(\mathbf{x}) - \rho$ [22].

C. Run-time detection

Once the trained models introduced in §III-B are created, the detection process is performed sequentially on a system call sequence using a sliding window of

size L, with a stride length S. During the testing period, *i.e.,* when the initial sliding window L starting with the first system call in the sequence is filled up, we use the techniques described previously in §III-A to extract and normalize the n-gram feature vector $\mathbf{x}_1$. The i-th feature vector $\mathbf{x}_i$ is extracted from the sliding window starting at the $1 + S(i-1)$-th system call. The *delay* D of the sequential detection can be defined as the number of observed system calls before making a successful detection.

Previous work on behavioral malware detection [9] was performed at the uni-feature-level, meaning that they treated observations of a single feature as a time series, and combined the sequential detection results of each uni-feature time series using majority voting. For our problem, the time sequence of feature vector $[\mathbf{x}_1, ..., \mathbf{x}_T]^\intercal \in \mathbb{R}^{T\times p}$ is extremely high-dimensional and sparse, especially for system call sequences of $n > 1$.

For our work, it is better not to perform the sequential detection at the single feature-level, but rather to use it on the sequence of the univariate test statistics $\mathbf{y} \in \mathbb{R}^T = [y_1, ..., y_T]^\intercal$ obtained by carrying out the anomaly detection algorithm introduced in §III-B on the feature sequence $[\mathbf{x}_1, ..., \mathbf{x}_T]^\intercal$. For one-SVM, the test statistic of $\mathbf{x}_i$ is its distance to the separating hyperplane $y_i \equiv \mathbf{w}^T\phi(\mathbf{x}_i) - \rho$. This paper considers two sequential detection algorithms: i) the CUSUM test; ii) the SR procedure [19]. The sequential algorithms detect a potential *change point* $t \in [1, T]$ of a series $\mathbf{y}$ by inspecting the likelihood ratio between the probability distribution function (PDF) of $\mathbf{y}$ before the change point and after the change point. Under the assumption that the PDF of the benign sequence and malicious sequence are known as $f_Y(y_i)$ and $g_{\hat{Y}}(y_i)$ respectively. The log-likelihood between y_i being a random sample of malicious distribution $\hat{Y}$ and benign distribution Y is $r_i \equiv \ln \frac{g_{\hat{Y}}(y_i)}{f_Y(y_i)}$. The CUSUM test detects the time:

$$t_{\tau_W} = \min\{i : W_i > \tau_W\}, \tag{1}$$

with $W_i \equiv \max\{0, W_{i-1} + r_i\}$, $W_0 = 0$, and τ_W is a threshold to balance the trade-off between the tolerable false alarm rate and the detection rate of the sequential test.

Similarly, define $l_i \equiv \frac{g_{\hat{Y}}(y_i)}{f_Y(y_i)}$, the SR procedure detects a time:

$$t_{\tau_C} = \min\{i : C_i > \tau_C\}, \tag{2}$$

with $C_i \equiv (1+C_{i-1})l_i, C_0 = 0$, and tunable threshold τ_C.

We simply assume that all test statistics follow a Gaussian distribution, the PDF of benign samples is

$$f_Y(y_i) = \frac{1}{\sqrt{2\pi}\sigma_Y} e^{-\frac{(y_i-\mu_Y)^2}{2\sigma_Y^2}},$$

and the PDF of malicious samples is:

$$g_{\hat{Y}}(y_i) = \frac{1}{\sqrt{2\pi}\sigma_{\hat{Y}}} e^{-\frac{(y_i-\mu_{\hat{Y}})^2}{2\sigma_{\hat{Y}}^2}}.$$

We further assume that the difference between the PDF functions of the benign and malicious test statistics mainly lies in the mean value μ_Y, $\mu_{\hat{Y}}$. It is reasonable to estimate the mean μ_Y and standard deviation σ_Y of the benign samples by the sample mean and sample standard deviation of the benign training data. For one-class SVM, the test statistics $\hat{Y}$ of the malicious samples are typically smaller than those of the benign samples and, thus, the mean of and standard deviation of malicious samples are

$$\mu_{\hat{Y}} = \mu_Y - m\sigma_Y, \quad \sigma_{\hat{Y}} = \sigma_Y,$$

respectively. Note that $m > 0$ is a tunable sensitivity parameter for the sequential test.

IV. Experimental Setup

The source code for the firmware/OS running on *Alexa-Enabled Devices* (devices with built-in access to Alexa services) is not open source. As such, it is not possible to obtain the feature data used for malware detection directly as one would on a traditional device running Linux. However, it is possible to install the Alexa Voice Service on a variety of other devices. For this experiment, we used a *Raspberry Pi 2 Model B* running Alexa Voice Service in place of an *Amazon Echo Dot*, which is one of the most popular smart speakers.

A. Raspberry-Pi setup

The model of the device running the Alexa Voice Service is a Raspberry Pi 2 Model B. The operating system installed on the device is a *Raspbian-Stretch* as provided by the Raspberry Pi foundation website. We use this device as a surrogate for the *Amazon Echo Dot (Gen 2)* for two reasons.

First, the Amazon Echo Dot runs Amazon's "*Fire OS*". Because Fire OS is proprietary, we are unable to do many of the things required to perform our data collection on the device. However, the Raspberry Pi runs the *Raspbian-Stretch* operating system, which is open source and allows us to customize the device and access the OS kernel. While the two operating systems are different from one another, they are both Unix-based, and as such, have a similar lower-level functionality.

Second, the Raspberry Pi 2 Model B's central processing unit is the Broadcom BCM2836 [23], while the Amazon Echo Dot's central processing unit is the MediaTek MT8163V [24], both of which are quad-core processors whose underlying architecture is the ARM Cortex-A53 micro-architecture, implementing the ARMv8-A 64-bit instruction set [23], [25].

B. Alexa installation

The Alexa-Enabled devices available from Amazon (such as the Echo) are not suitable for collecting the data we use as features for our detection model. As such, we installed the *Amazon Voice Service* on a Raspberry Pi 2 Model B. To achieve feature parity with an official Alexa-Enabled device, one must register as a developer with Amazon, register the device upon which the Alexa Voice Service will be installed, and download some libraries, programs, and security certificates from Amazon. With the addition of a USB microphone and analog speakers connected via the Raspberry Pi's 3.5mm audio output port, a user can query the device in the same way as one would with a device such as an Amazon Echo and receive the same results.

C. Exercising the Alexa capabilities

Currently, there is no interface available to the public to facilitate automatic interaction with the Alexa Voice Service for practicing built-in capabilities. The majority of computation performed when making queries is handled at a remote server rather than the Alexa-Enabled device itself. Consequently, the system calls made on the device itself when handling different queries are very similar as the device mostly establishes a connection with Amazon's servers and offloads the query. Furthermore, in practical use, the device is not handling queries at all, but rather listening for a wake word. During this passive listening state, the Amazon Voice Service is still running and awaiting input. This means that even when the device is not actively handling a query, it still continually makes system calls as part of the service. As such, the pattern of behavior demonstrated while the device is idle is still useful in building a realistic model for anomaly detection. When gathering data for our experiments, we exercised a broad set of Alexa built-in capabilities from books, calendar, weather, music, and standard built-in skills categories. We also gathered data when Alexa was idle, both in a quiet environment as well as in an environment with ambient noise.

V. Experimental Results and comparisons

We start by describing the distribution of the system call features in §V-A for each of the 1) malware-free Alexa-Enabled devices that are *not* being queried; 2) malware-free Alexa-Enable devices that *are* being queried; and 3) malware-infected Alexa-Enabled devices. The results in §V-B show that Alexa has a behavioral signature that can be used to distinguish an OS running Alexa from one that is not. Finally, the experiments described in §V-C reveal the relationship between detection delay and detection accuracy, and demonstrate that the anomaly detector can detect malware execution accurately and efficiently. In this work, the detection accuracy is evaluated by two metrics: the *false alarm rate (FAR)* and the *true positive rate (TPR)*, which represent the fraction of benign samples that are falsely identified as anomalies, and the fraction of malicious samples that are correctly detected as anomalies, respectively.

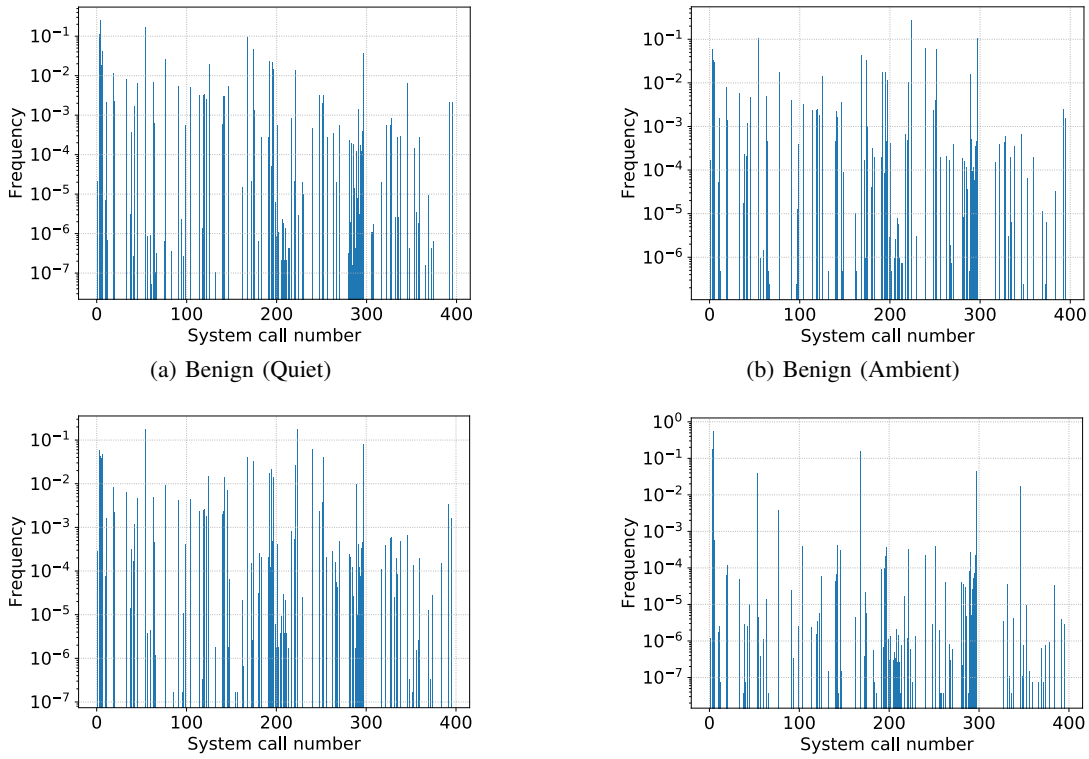
A. Statistics of system calls

Table I shows the discrepancy between the system calls observed when the Alexa-Pi device is idle and placed in a quiet room, and those observed when: i) the malware-free device is placed in a noisy room (*i.e.,* ambient); ii) the malware-free device is being queried; and iii) malware infects the device [1]. Furthermore, Fig. 1 shows the histograms of system calls observed for these four scenarios. We can see from the table and the histogram figure that the behavior of Alexa-enable devices in a noisy room is distinct from its behavior in a quite place. Fig. 1 also demonstrates the system call distributional difference among the four scenarios.

Ambient	pwrite64, fdatasync, getrandom, rt_sigreturn, tgkill, mremap
Query	pwrite64, getrandom, fdatasync, tgkill, rt_sigreturn, mremap, sched_getparam, sched_getscheduler, fallocate, readlink
Infected	getrandom, fcntl, kcmp, timerfd_create, tgkill, waitid, setxattr, capget, rmdir, rt_sigreturn, sigaltstack, timer_create, flock, msync, ppoll, timer_delete, timer_settime

TABLE I: System calls that do not appear when the Alexa-Pi device is idle but do appear in infected (malicious) traces, benign traces in an environment with ambient noise, and benign traces when the device is being queried.

[1]The MD5 hashes of the malware sample that we use in this work are: 217f26322d3166c9ce9595710874ceff; 5b4a1226bfe9a557b73cf4138330b6c7; and 218c8518121efec889b58c335adf5203. The malware samples were obtained from *VirusShare* (https://virusshare.com/).



(a) Benign (Quiet)

(b) Benign (Ambient)

(c) Benign (Query)

(d) Infected

Fig. 1: System call distribution.

The system calls that occur in the infected traces that are not in the benign traces of the idle device being queried, or being placed in a noisy environment are shown in Table II. Seventeen system calls are absent from the idle traces but are present in the infected traces as shown in the first row of Table I. Also, the number of system calls in the infected traces that are not in the benign traces is reduced to fourteen by exercising Alexa's features (*e.g.,* querying Alexa) as shown in Table I.

Infected	fcntl, kcmp, timerfd_create, waitid, setxattr, capget, rmdir, sigaltstack, timer_create, flock, msync, ppoll, timer_delete, timer_settime

TABLE II: System calls in malicious traces and are not in any of the benign traces (queried, ambient, or idle.)

To study the characteristics of voice controlled devices under various scenarios, we focused on the most frequently invoked system calls of the OS, their corresponding occurrence frequency (*i.e.,* PDF) and the cumulative frequency (*i.e.,* CDF) when: i) the device is idle in a quiet environment (Table III); ii) the device is being queried (Table IV); iii) the device is placed in a noisy environment (Table V); and iv) the device is infected by malware samples (Table VI).

If a malware sample is relatively passive, its pattern of malicious activity may also hide in the system calls that are not triggered frequently. Therefore, we also show the list of least frequent system calls of the infected traces:

`setuid32`	`socketpair`
`sigaltstack`	`rename`
`timer_create`	`flock`
`msync`	`ppoll`
`timer_delete`	`timer_settime`

Each has an occurrence frequency of at most 4×10^{-8}.

It is noteworthy that the benign ambient system-call pattern presented in Table V is more similar to the pattern in the benign query shown in Table IV than the benign idle system-calls shown in Table III. The presence of system-calls such as `write` in both the Query and Ambient traces, but surprisingly not in the Idle traces suggests that the device has very similar behavior in the presence of ambient sound in the room prior and after the activation with the wake word, *"Alexa"*.

B. Experimental results of detecting Alexa operation

We demonstrate that the behavioral signature of our Alexa-enabled device is not only characterized by its underlying hardware and OS, but is also characterized by Alexa's operation. Before showing the capability of

System call	PDF	CDF
ioctl	0.2803	0.2803
rt_sigaction	0.0831	0.3634
close	0.0733	0.4367
read	0.0579	0.4946
getrusage	0.0457	0.5402
mmap2	0.0420	0.5822
poll	0.0399	0.6221
stat64	0.0396	0.6617
mprotect	0.0355	0.6972
recvmsg	0.0354	0.7327

TABLE III: System call distribution of benign traces when the device is idle (top ten system calls).

System call	PDF	CDF
ioctl	0.1771	0.1771
gettid	0.1735	0.3505
recvmsg	0.0793	0.4299
futex	0.0617	0.4915
read	0.0579	0.5494
close	0.0469	0.5963
write	0.0417	0.6380
epoll_wait	0.0412	0.6791
poll	0.0400	0.7191
open	0.0385	0.7576

TABLE IV: System call distribution of benign traces when the device is queried (top ten system calls).

System call	PDF	CDF
gettid	0.2746	0.2746
ioctl	0.1066	0.3811
recvmsg	0.1055	0.4867
futex	0.0637	0.5504
epoll_wait	0.0597	0.6101
read	0.0581	0.6682
poll	0.0441	0.7124
rt_sigaction	0.0331	0.7455
write	0.0322	0.7777
close	0.0305	0.8082

TABLE V: System call distribution of benign traces when the device is subjected to ambient noise (top ten system calls).

System call	PDF	CDF
write	0.5548	0.5548
read	0.1769	0.7318
poll	0.1587	0.8905
recvmsg	0.0433	0.9338
ioctl	0.0391	0.9729
epoll_pwait	0.0175	0.9905
getrusage	0.0039	0.9943
open	0.0006	0.9949
close	0.0006	0.9955
_newselect	0.0004	0.9959

TABLE VI: System call distribution of infected traces (top ten system calls).

an anomaly detector to learn the normal operation of Alexa, the first step is to show that the Alexa client operation is distinguishable from the other programs that are executing on the OS of the device. To achieve this goal, we designed a proof-of-concept experiment to test whether or not an anomaly detector can successfully distinguish a device with the Alexa client present from the same device without the presence of Alexa.

A one-class SVM based anomaly detector was trained by a trace, containing 2.02×10^7 system calls, generated by a Raspberry-Pi device that was not running the Alexa client. Then, we tested if the anomaly detector could differentiate a sequence of system calls generated by Alexa-enabled device from a sequence of system calls generated by the device without Alexa. Fig. 2 shows the resulting ROC curve with 2-gram features and an observation window of 1000 system calls. We can see that Alexa's presence can be detected 100% of the time without triggering any false alarms when the delay of the detection is $D = 100000$ system calls.

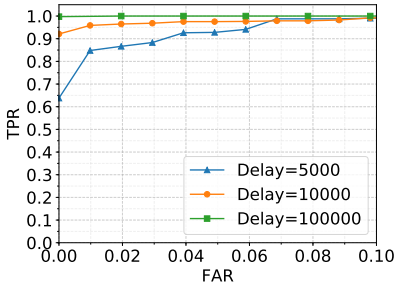
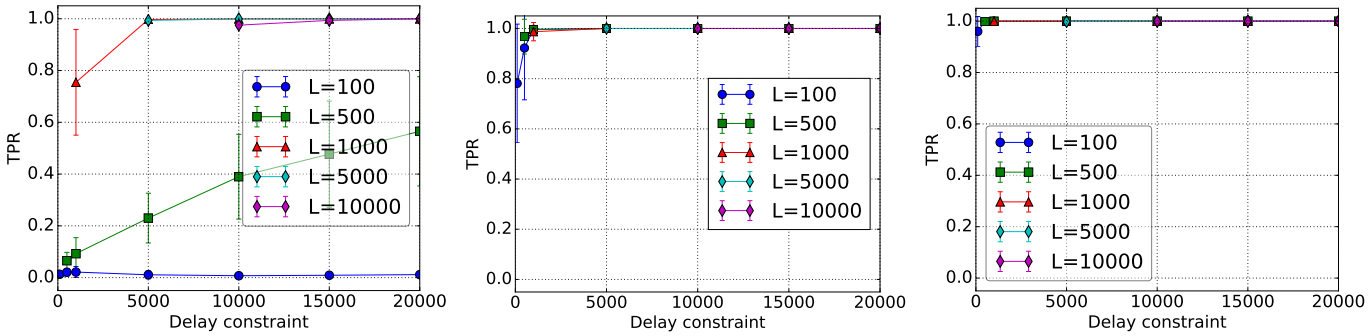


Fig. 2: ROC curve of Alexa operation detection ($L = 1000$, 2-gram, $m = 3$).

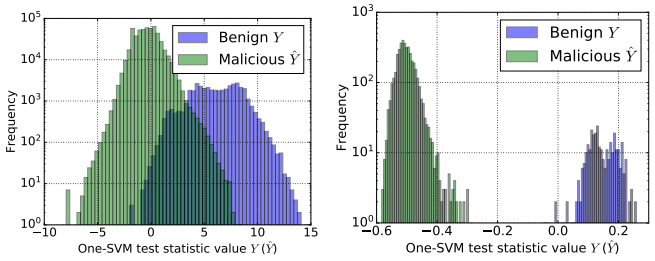
C. Experimental results of on-line malware detection

This section investigates the influence of the observation window size L, the n-gram, and the detection delay constraint $D_u = L + (i-1)S$, where i denotes the index of the maximum allowed detection observation window index and S denotes the stride size, on the detection accuracy. In the experiment, we consider every system call trace of length 2×10^4 as a single sample. In order to decide whether a trace is abnormal or not, the anomaly detector extracts a sequence of test statistics $\mathbf{y} = [y_1, ..., y_T]$ based on n-gram feature vectors $\mathbf{x}_1, ..., \mathbf{x}_T$ observed in a sliding window of L with stride $S = L/2$ on the trace. It then performs the CUSUM test or SR procedure over the sequence of test statistics. An earlier detection of malware execution is desirable since we want to minimize the damage caused by malware. However, raising an alarm immediately after spotting any suspicious activities may cause too many false alarms. It is of great practical importance and interest to find an operating point that can balance the trade-off between the accuracy of detection and the detection delay.



(a) 1-gram feature (b) 2-gram feature (c) 3-gram feature

Fig. 3: TPR obtained at FAR=0 vs. detection delay constraint D_u (CUSUM test).



(a) $L = 100$, $\frac{\mu_Y - \mu_{\hat{Y}}}{\sigma_Y} = 2.92$. (b) L=$1 * 10^4$, $\frac{\mu_Y - \mu_{\hat{Y}}}{\sigma_Y}$=12.45.

Fig. 4: Histograms of the one-class SVM's test statistics when $L = 100$ or $L = 10000$ (using 2-gram features).

Our experiment performs 10-fold cross validation, namely we partition the benign data to ten shares. In each iteration, one share of the benign data is used for testing and the rest nine shares are used to train the anomaly detector. In the experiment, we want to know the variation of the detection rate versus a tolerable *delay constraint* D_u: the anomaly detector keeps making new observations if $D < D_u$, and declares anomaly once the change-point detection's test statistic (W_i for CUSUM, C_i for SR) is greater than a certain threshold; however as soon as the delay $D > D_u$, it stops taking new observations and makes a final decision. In the experiment, we observe that the absolute difference between the sample mean of benign samples' one-SVM test statistics and the sample mean of malicious samples' one-SVM test statistics tends to be larger for the case with a large L than the case with a small L, as shown in Fig. 4. Hence, we use $m = 3$ for $L = 1 \times 10^2, 5 \times 10^2, 1 \times 10^3, 5 \times 10^3$, and $m = 12$ for $L = 1 \times 10^4$ in the experiment.

Fig. 3 presents the relationship between the detection delay constraint D_u and the achievable TPR (averaged over 10-fold cross validation) for CUSUM test obtained at zero FAR for the observation window size taking values in 1×10^2, 5×10^2, 1×10^3, 5×10^3,1×10^4 with n-gram features ($n \in \{1, 2, 3\}$). Note we also implement the SR procedure and obtain very similar results as Fig. 3. From Fig. 3, we can see that, in general, the detection accuracy improves when we have a larger delay constraint. Using 1-grams, the detector achieves a TPR of over 99.9% with $L = 1000$ when the delay is greater and equal to 1×10^4. Comparing Fig. 3a with Fig. 3b and Fig. 3c, we can see the detection accuracy is significantly improved with 2-gram or 3-gram features for $L = 100, 500, 1000$. With 2-gram, the detector achieves 100% detection rate for all L with a delay of 1×10^4 system calls.

Fig. 5 shows the distribution of detection delay D of CUSUM test if we do not place any upper constraint D_u on the delay with $L = 500$ and 2-gram features. This histogram is obtained with zero false alarms and 100% true positive. We can see that the delay mainly lies in the range from 1000 to 2250. This experiment shows that the anomaly detector is able to identify anomalies within a very short time limit and also obtain a very high detection accuracy. Table VII presents the average detection delay D at zero FAR of the CUSUM test and SR procedure for various choices of L, and we can see that the difference of the average detection delay between the CUSUM and SR procedure is very small. Note that our anomaly detector only uses n-grams observed in the benign training data, and we do not use any "malicious" n-grams that only occur during the execution of malware. Masquerade attacks, which mimic the operations of the normal systems, try to perform malicious activities using only the system calls that occur frequently in the kernel of a malware-free device. Our promising experiment results demonstrate the potential capability in detecting such stealthy masquerade attacks.

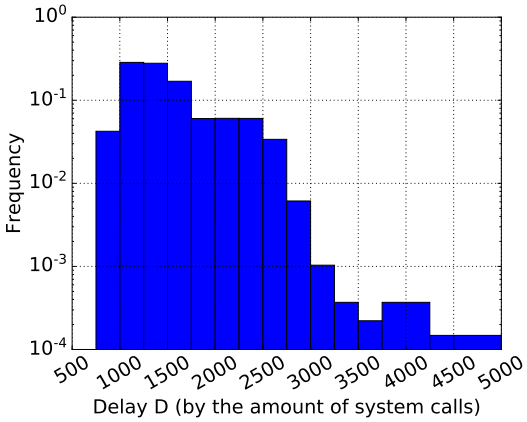


Fig. 5: Distribution of delayed system calls D of CUSUM with 2-gram at TPR=100%, FAR=0% (L=500).

L		100	500	1000	5000	10000
1-gram	CUSUM μ_D	9079	9746	2914	7052	10160
	SR μ_D	9074	9640	2925	7063	10020
2-gram	CUSUM μ_D	1213	1392	2076	5514	10000
	SR μ_D	1219	1410	2094	5514	10000

TABLE VII: Compare the average detection delay μ_D obtained of the CUSUM test and the SR procedure for various observation window size L.

VI. Conclusions

Intelligent virtual assistants, such as Alexa-enabled IoT devices, typically have specialized functionality which makes their ordinary behavior easy to learn. A semi-supervised anomaly detector, trained by ordinary system call sequences generated by the IoT device's kernel, can effectively identify malware activities without needing to have seen malware samples in training. In this paper, we demonstrate the efficacy of an on-line anomaly detector based on a one-class SVM combined with the CUSUM test or the SR procedure to detect malware infections on a voice controlled Alexa-enabled IoT device. Our initial experimental results show that the anomaly detector is capable of detecting the presence of malware samples studied in this paper with significantly high accuracy in a very short time, with only modest-sized training data.

Acknowledgment

The authors would like to thank Ms. Rachelle St. Fleur at Drexel University for helpful feedback to this paper.

References

[1] S. O. Blog, "IoT devices being increasingly used for ddos attacks," http://www.symantec.com/connect/blogs/iot-devices-being-increasingly-used-ddos-attacks, Sep 2016.

[2] Mirai-source-code. [Online]. Available: https://github.com/jgamblin/Mirai-Source-Code

[3] "The internet of stings," *The Economist*, October 2016.

[4] M. Kan, "Hackers create more IoT botnets with Mirai source code," *ITworld*, October 2016.

[5] T. Moffitt, "Source code for Mirai IoT malware released," *Webroot Threat Blog*, October 2016.

[6] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *COLT*, 1992, pp. 144–152.

[7] E. Stehle, K. Lynch, M. Shevertalov, C. Rorres, and S. Mancoridis, "Diagnosis of software failures using computational geometry," in *IEEE/ACM ASE*, Nov 2011, pp. 496–499.

[8] R. Canzanese, M. Kam, and S. Mancoridis, "Inoculation against malware infection using kernel-level software sensors," in *ACM ICAC*, 2011, pp. 101–110.

[9] ——, "Multi-channel change-point malware detection," in *IEEE SERE*, June 2013, pp. 70–79.

[10] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior," in *ESEC-FSE companion '07*, 2007, pp. 5–14.

[11] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *IEEE S&P*, May 2001, pp. 38–49.

[12] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *ACSAC*, Dec 2007, pp. 421–430.

[13] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

[14] N. Ye, X. Li, Q. Chen, S. M. Emran, and M. Xu, "Probabilistic techniques for intrusion detection based on computer audit data," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 31, no. 4, pp. 266–274, 2001.

[15] R. Canzanese, M. Kam, and S. Mancoridis, "Toward an automatic, online behavioral malware classification system," in *IEEE SASO*, 2013.

[16] R. Canzanese, S. Mancoridis, and M. Kam, "Run-time classification of malicious processes using system call analysis," in *MALWARE*, 2015.

[17] Webroot, "2018 Webroot threat report." [Online]. Available: https://www-cdn.webroot.com/9315/2354/6488/2018-Webroot-Threat-Report_US-ONLINE.pdf

[18] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. pp. 100–115, 1954.

[19] A. N. Shiryaev, "The problem of the most rapid detection of a disturbance in a stationary process," in *Soviet Math. Dokl*, vol. 2, no. 795-799, 1961.

[20] R. Canzanese, S. Mancoridis, and M. Kam, "System call-based detection of malicious processes," in *IEEE QRS*, Aug 2015, pp. 119–124.

[21] N. An, A. Duff, G. Naik, M. Faloutsos, S. Weber, and S. Mancoridis, "Behavioral anomaly detection of malware on home routers," in *2017 12th MALWARE*, Oct 2017, pp. 47–54.

[22] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[23] Raspberry-Pi-Foundation, "BCM2836," Oct. 2016. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2836/README.md

[24] WikiDevi, "Amazon Echo Dot (RS03QR)." [Online]. Available: https://wikidevi.com/wiki/Amazon_Echo_Dot_(RS03QR)

[25] MediaTek, "Mediatek MT8163V/A for tablets." [Online]. Available: https://www.mediatek.com/products/tablets/mt8163