Detecting Privacy Violations in Children's Apps Using HPCs

Author:

Suha Hussain

Queens High School for the Sciences at York College

Mentors:

Dr. Kanad Basu

New York University Tandon School of Engineering

Dr. Ujjwal Gupta

Intel Corporation

Dr. Ramesh Karri

New York University Tandon School of Engineering

Abstract

The recent surge in children's apps reflects the proliferation of new avenues for education and entertainment globally. However, more than half of all children's apps on the Android platform violate the Children's Online Privacy Protection Act (COPPA), indicating an invasion of user data privacy that threatens the safety of millions. Previous research on detecting COPPA violations rely upon an analysis of the system binary, which is neither scalable nor reliable. To overcome these challenges, hardware performance counters (HPCs) were utilized to detect COPPA violations. A novel dataset was established after the profiling of a number of COPPA-compliant and COPPA-violating Android apps. Based upon this dataset, two methods, a general COPPA violation detector and a series of specialized COPPA violation detectors, were formulated. The former detects the existence of any possible COPPA violation. Supervised learning algorithms were applied to the whole dataset and, to address HPC measurement constraints, to feature-reduced data. The latter detects the existence of a specific COPPA violation. Thus, several classifiers trained upon feature-reduced data were developed. In addition to yielding high accuracies and low misclassification rates, these classifiers are secure and efficient due to the nature of HPCs.

Contents

1 Introduction **4**

2 Related Work **5**

3 App Corpus **6**

4 Performance Profiling **7**

5 Data Labeling and Feature Selection **8**

6 General COPPA Violation Detection **9**

6.1 Background 9

6.2 Methodology 11

6.3 Results 11

7 Specialized COPPA Violation Detection **13**

7.1 Methodology 13

7.2 Results 14

8 Discussion **16**

9 Conclusion **16**

1 Introduction

According to the Federal Bureau of Investigation (FBI), over 1300 cybercrimes were committed against children in 2017 (1). Children heavily use mobile apps, presenting these malicious agents with a new attack vector. Children are often issued standard warnings, e.g., not speaking to strangers on the Internet. However, many apps engage in extensive data collection, enabling malicious agents to covertly use personal data and rendering standard warnings insufficient (2). Even if the apps are not malicious, data breaches and leaks pose a security threat. One notable case in this context is that of uKnow, a parental intelligence system, whose data breach in 2016 exposed sensitive data for over 1700 children, including photos and passwords (16).

The Children's Online Privacy Protection Act (COPPA) is one of the few comprehensive online privacy regulations in the USA (Commission). To protect children from extensive data collection, COPPA restricts the types of data gathered from children and grants parents more control. Child-directed apps are sent considerable information on COPPA upon entering the market. Still, over half of these apps violate COPPA, with many of them passing information to third party SDKs and advertising agencies without parental consent and bypassing the standard Android permissions system (17). **The Federal Trade Commission (FTC) analyzes these apps manually. This is inefficient, ineffective and a losing proposition considering the large number of apps, and the multitude of transmission vectors in every app and the obfuscation protocols used by the apps.**

To combat this issue, researchers conducted a dynamic analysis of network traffic to monitor COPPA compliance (17). They created an online repository and an automated testbed to determine the COPPA compliance of Android apps. Dynamic analysis compiles procedures used to transmit data and obscure transmissions making it rigid when the techniques are not known, incurring false negatives. Due to its specialized nature, consumers rely upon the continual updating of the repository, which leaves gaps between the app update and repository use timelines. A consumer does not check a repository prior to app installation. Children often use apps prior to informing their parents, by which time, their personal data is already disseminated. Non-COPPA compliant apps were able to deter potential lawsuits by claiming the app is not directed toward children, despite the FTC's encompassing definition of a "child-directed app" (8). **Thus, an average consumer of child-directed apps should have access to COPPA**

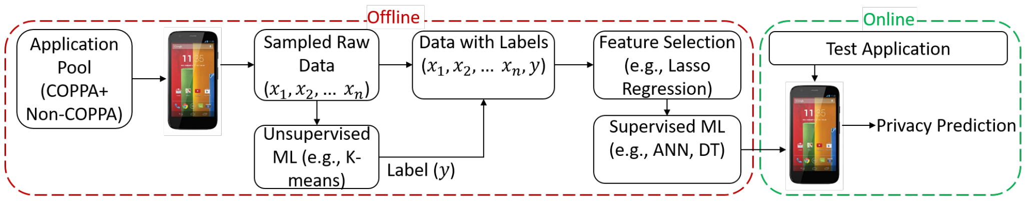

Offline
Application Pool (COPPA+ Non-COPPA)
Sampled Raw Data $(x_1, x_2, \dots x_n)$
Data with Labels $(x_1, x_2, \dots x_n, y)$
Feature Selection (e.g., Lasso Regression)
Unsupervised ML (e.g., K-means)
Label (y)
Supervised ML (e.g., ANN, DT)
Online
Test Application
Privacy Prediction

Figure 1: An overview of the COPPA violation detection approach

compliance monitoring tool that is easy to use and works in real-time.

To overcome these issues of scalability and reliability, this work, unlike prior research, presents machine learning-based detection methods that rely upon hardware performance data, or data generated from hardware performance counters (HPCs). HPCs are registers incorporated in modern processors for performance calculations. They have gained popularity as the basis for malware detection systems for several reasons, including its high granularity, low resource consumption, and high resistance to tampering. Profiling applications using HPCs provides a more generic infrastructure, independent of OS. Leveraging HPC data means that the methods are not only efficient and secure, but are highly adaptable to unknown threats as they avert the issue of unknown traffic obfuscation tools entirely, further augmenting their adaptability and effectiveness. The two major detection methods are general COPPA violation detection and specialized COPPA violation detection. The former indicates the existence of any COPPA violation, while the latter targets specific COPPA violations, including accessing permission protected resources or transmitting location data. An overview of the proposed approach can be found in Figure 1.

2 Related Work

Hardware performance counters (HPCs) are registers found in all modern day computers, which measure specific architectural and microarchitectural events, including the number of branches taken, number of instructions retired, and number of L1 cache misses. HPCs were initially developed for debugging and compiler optimization. However, over time, they have been used for a myriad of functions, including auto tuning (21), performance optimization (4; 3; 11), high performance computing (22; 15), OS support (13; 14), power analysis (19) and malware detection (20; 7). Intel processors are equipped with a special functional unit for HPC measurements, known as Performance Monitoring Unit (PMU) (Das et al.). Intel Ivy-Bridge and Broadwell processors can monitor 468 and 519 hardware events, respectively (19). However, only four to six HPC parameters can be monitored simultaneously; this is typical of most smartphone processors. Application of HPCs for any form of privacy analysis was first shown by researchers at Worcester Polytechnic Institute(10). The authors demonstrated that a malicious application can infer the privately accessed websites in an user device by monitoring the HPCs. Their supervised learning algorithms were successful in classifying 30 top Alexa sites and 10 whistleblowing portals.

The Children's Online Privacy Protection Act (COPPA) dictates how online services can handle Personal Identifiable Information (PII) collected from children under the age of 13 and how parents are informed of information distribution. COPPA's definition of PII includes personal information (name, email-address and phone numbers), geographical locations, audio and visual recordings as well as persistent identifiers like IMEI, MAC addresses, Android ID and Android advertising ID. Any PII data collected from children cannot be used for behavioral profiling or device tracking and can only be collected contingent upon parental permission (18). As stated previously, the issue of COPPA compliance enforcement was addressed by researchers at the International Computer Science Institute (17). Their research revealed that more than half of all children's apps on the Android platform violated COPPA. They established a dataset of COPPA compliant and COPPA violating apps in addition to proposing a specialized automated testbed reliant upon an unadaptable dynamic analysis that enables false negatives. Not only are their methods ineffective, but they are inefficient and not scalable.

3 App Corpus

The creation of an app corpus utilized Python, the Google Play Store, and COPPA dataset from the International Computer Science Institute (17). The app corpus contains the information of and COPPA compliance status of 100 different Android apps.

The Google Play Store was scraped for apps to test. Android enables free apps to be downloaded without requiring a password on default settings, making free apps more likely to be downloaded and used by children prior to parental examination. In order to ensure that all of these apps are governed by COPPA, only apps under Google

Plays Designed for Families (DFF) program were utilized. These apps voluntarily chose to indicate that the whole or a percentage of their target audience are children. To be a part of this program, they received guidance on COPPA compliance from Google and affirmed that these standards were met. In addition to extracting the application IDs for profiling, the COPPA compliance status of each app must be determined. The previously mentioned repository on COPPA compliance was utilized to label each app with a 0 or a 1 to indicate the presence of a COPPA violation. These are the labels utilized in a general COPPA violation detector. Instead of indicating an overall violation of COPPA, a specialized detector indicates the presence of a specific COPPA-violating behavior, which can be found from the same dataset. These behaviors, or features, are enumerated in (Commission). Note that each of these behaviors are observed without the consumer providing explicit permission. The corpus is composed of 28 apps that violate COPPA.

4 Performance Profiling

A Moto-G smartphone running Android 8.0 with kernel 3.18.31-perf-g708ac5e was employed in conjunction with Android's $simpleperf$ and Monkey tool to generate HPC data for every app contained within the app corpus.

Android is based on a modified version of the Linux kernel; this kernel wraps the HPCs into perf events as well as provides hardware independent software events and tracepoint events. Android's $simpleperf$ tool obtains raw event counter information from apps. The events are limited to hardware cache events, hardware events, and raw CPU PMU events. Android's UI/Application Exerciser Monkey (the ”Monkey”) from Android's development SDK automates the usage and testing of apps. Monkey executes a psuedorandom stream of user input actions into each app, similar to the process of fuzzing. The stream of actions is generated from a random seed, making it reproducible.

Monkey's effectiveness can be questioned, since Monkey only injects random user actions without examining visual input, its usage might result in sub-optimal execution path coverage. Previous research has determined that Monkey matched or exceeded human coverage 61% of the time (17). However, this experiment compared Monkey's coverage to that of adult humans. On the other hand, these methods are designed for children, who are more likely to have a coverage similar to the psuedorandom injections of Monkey than adults.

As modern CPUs only have a small amount of registers for HPCs, prior methods had to select only that number of events for profiling. Since Monkey's user input is reproducible, each app ran multiple times for each configuration with the same Monkey seed. The events list was split into sets of six and sweeped through, resolving the issue of hardware limits. Moreover, each counter was collected 100% of the time to prevent hardware multiplexing from biasing the data. Five events were measured at a time for nine rounds and three events for the 10th round, generating a total of 48 HPC parameters. Table 1 displays information on the dataset. Further details on the data can be found at https://github.com/suhacker1/hpc-a.

Table 1: Frequency of COPPA parameters transmitted by COPPA-violating apps

Parameter	Transmission(%)	Parameter	Transmission(%)
Protected Resources	8.11	Name	0.00
Location	13.51	Phone	5.41
Email	0.00	IMEI	10.81
Advertising ID	72.97	Android ID	66.22
Device Description	67.57	Wi-Fi MAC Address	2.70
Google Services ID	0.00	Serial Number	4.05
SIM Serial Number	0.00		

5 Data Labeling and Feature Selection

Every app that violates COPPA will contain phases where it does and does not violate COPPA. In order to obtain more veracious data, after profiling each application and sampling every 100 ms, k-means clustering was applied. This algorithm separated the

COPPA-violating and COPPA-compliant phases of each app, enabling more accurate discernment and classification for the detectors.

k-means clustering is an unsupervised learning algorithm that organizes data into a number (k) clusters. Essentially, given some data $(x_1, x_2, x_3, ..., x_n)$, the goal is to partition the data into a series of clusters or sets that can be represented by $(K_1, K_2, K_3, ..., K_n)$ where K_n is the kth cluster. The procedure is as follows:

1. Assign every data point a cluster with the closest mean. Distance is defined as the least squared Euclidean distance.
2. Calculate the new means to be the centroids (centers) of the data points in each cluster.
3. Repeat steps 1 and 2 until convergence.

After applying k-means clustering, either LASSO or Ridge regression was applied as a form of feature selection. The Least Absolute Shrinkage and Selection Operator (LASSO/Lasso) or Ridge algorithms (12) were employed to obtain a subset of all the features that can be collected on the platform. The LASSO applies a ℓ_1-norm penalty of coefficients on the optimization objective of the regression problem as follows:

$$\hat{\mathbf{a}} =_{\mathbf{a}} \sum_{k=1}^{M} \left[(y_k - \hat{y}_k)^2 + \lambda ||\mathbf{a}||_1 \right]$$

where M is the total number of samples, y_k is the label for k^{th} sample, $\hat{y}_k$ is the prediction for the label of k^{th} sample using regression. The vector **a** are the coefficients of the regression model and λ is the regularization parameter. When λ increases, more coefficients become zero, due to higher penalty. Therefore, the Lasso algorithm makes irrelevant coefficients zero and enabling us to select the features of non-zero coefficients. The Ridge algorithm is a variant of Lasso that adjusts for multicollinearity.

6 General COPPA Violation Detection

6.1 Background

Machine learning refers to a set of techniques that automate the analysis of data. Supervised learning is a task in machine learning that learns to produce certain outputs given particular inputs. Supervised learning algorithms infer a representation or a function from labeled training data. Typically, x represents the input while y represents the output or target. Examples of supervised learning algorithms include K-Nearest Neighbors, decision trees, random forests, and neural networks.

K-Nearest Neighbors, also referred to as KNN or k-NN, is a supervised learning algorithm used for both classification and regression problems. Assuming the geometry of the data encodes similarity, it organizes the data points according to a specific distance metric (Euclidean distance is the most common). For each data point, a number of nearest neighbors are chosen. Majority vote is applied to these neighbors to finalize the classification of that data point. KNNs are considered to be a form of instance-based learning as it does not require a separate training phase. The algorithm can also be thought of as calculating distance for each data point, estimating the conditional probability for each class, and assigning the data point to the class with the highest probability. If P is the conditional probability, k is the class number, A is the entire set of data, x is the data point, i is the index of the data point, j is the true label, I is an indicator function that evaluates to 1 based upon truth, and y is the label, the probability estimation function can be represented as:

$$P = \frac{1}{K} \sum_{i \in A} I(y^i = j)$$

A decision tree is a non-parametric supervised learning model that can be used for both classification and regression problems. With classification, the decision tree infers decision rules in an if-then or else-then format from training data in order to predict labels. Strong in interpretability and transparency, decision trees are prone to bias and instability. A random forest is an ensemble learning technique that utilized a multitude of decision trees at training time and returns the mode of the classes outputted by the individual trees, mitigating, but not removing the flaws inherent in decision tree usage. This is a form of a weighted neighborhood scheme where the weights represent neighbors.

Neural networks are the foundation of deep learning; these pattern-recognition frameworks store parameters in units known as neurons that are organized in layers. Training data is fed through these layers and backpropagation is utilized to update the parameters. While being more computationally effective, neural networks are more expensive and less transparent. Specifically, a neural network applies a softmax function to a logits output, the product of multiple neuron operations. Following previously defined conventions, the neural network can represented as a function $N(x)$ where:

$$N(x) = \arg\max(exp(z(x_i))/\sum_{j=1}^{n} exp(z(x_j)))$$

assuming the argument of the maximum is the conditional probability and $z(x)$ is the logits output (9).

6.2 Methodology

Sci-kit Learn, Jupyter Notebook, and Python were used to implement this detection method. First, k-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF), and a Neural Network (Multilayer Perceptron) were applied to the dataset and their performances were measured. Next, LASSO was executed on the dataset as a feature reduction technique. The algorithms were also applied to this feature-reduced dataset and the performances were measured.

6.3 Results

TPR (sensitivity or recall) measures the proportion of COPPA violations that were correctly identified, while FPR measures the proportion of COPPA compliant behaviors that were incorrectly classified. If TP represents true positives and TN represents true negatives,the formulas for TPR is:

$$TPR = \frac{TP}{TP + TN}$$

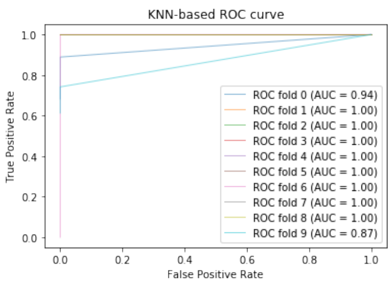
If FP represents false positives, the FPR formula is:

$$FPR = \frac{FP}{FP + TN}$$

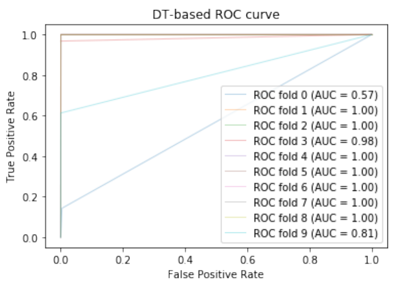
An ideal classifier should have 100% TPR and 0% FPR.

A receiver operator characteristic or ROC curve plots the TPR against the FPR. The area under the curve (AUC) of a ROC curve is a measure of the classifier performance. A higher AUC indicates a better performance. Precision, also called the positive predictive value (PPV), measures the fraction of relevant instances among all of the data. The formula is:

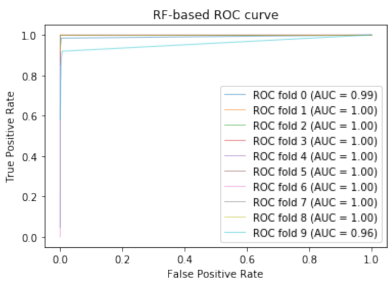
$$PPV = \frac{TP}{TP + FP}$$



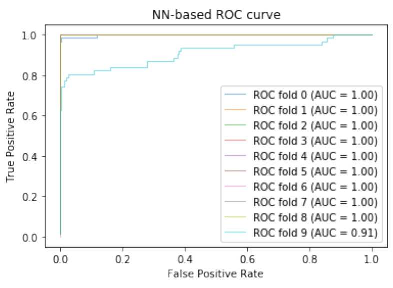
(a)



(b)



(c)



(d)

Figure 2: ROC curves with 48 HPCs for (a) KNN ,(b) DT, (c) RF, and (d) NN classifiers.

An ideal classifier should have a high PPV. Finally, accuracy indicates the percentage of samples that are correctly classified. A higher accuracy indicates better performance overall.

For every supervised learning algorithm, each of the above metrics were utilized to measure performance. For the initial dataset, the results of the KNN, DT, RF, and NN algorithms are shown in Table 2 with the ROC curves shown in Figure 2. The results for the feature-reduced dataset are shown in Table 3 with the ROC curves shown in Figure 3.

Table 2: Comparison of various classification schemes, represented in percentage.

Method	TPR	FPR	Precision	Recall	Accuracy
KNN	94.36	0.000035	99.99	94.36	99.94
RF	92.11	0.000034	99.99	92.11	99.91
DT	86.96	0.00032	99.99	86.96	99.84
NN	95.81	0.00037	99.99	95.81	99.92

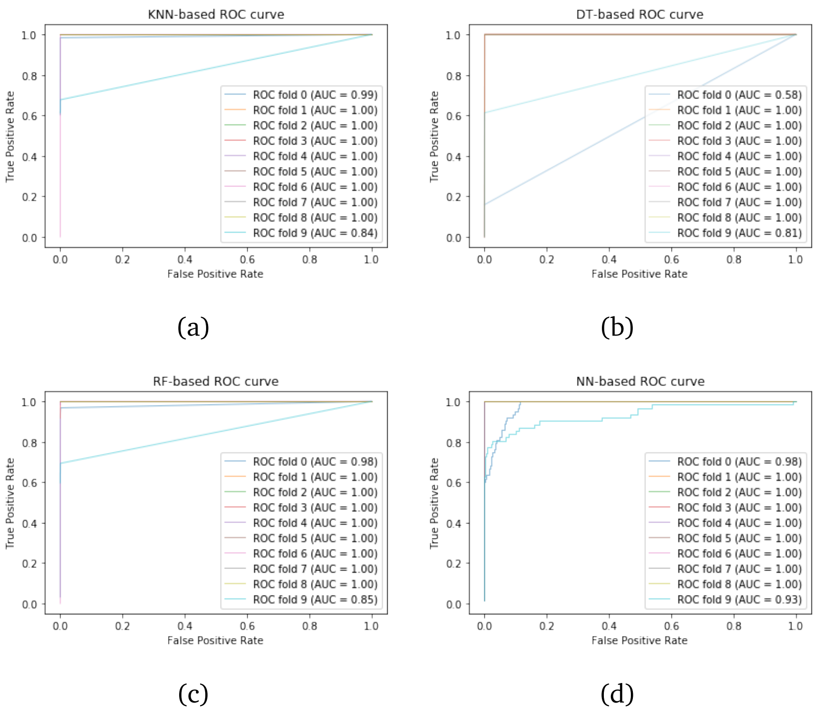


Figure 3: ROC curves with 4 HPCs for (a) KNN, (b) DT, (c) RF, and (d) NN classifiers.

Table 3: Comparison of various classification schemes with 4 HPC values, represented in percentage.

Method	TPR	FPR	Precision	Recall	Accuracy
KNN	95.65	0.000086	99.99	95.65	99.94
RF	91.7	0.000051	99.99	91.7	99.91
DT	87.4	0.0002	99.99	87.4	99.84
NN	92.4	0.0001	99.99	92.4	99.91

7 Specialized COPPA Violation Detection

7.1 Methodology

A specialized COPPA violation detection follows a similar methodology to the general COPPA violation detection. However, the target data sets, or COPPA labels, now exhibit

COPPA Violation	HPC Parameters	KNN Accuracy
Serial	instructions raw-l1-dcache raw-load-retired branch-stores	99.06
Advertising ID	L1-icache-load-misses raw-l1-icache raw-bus-access raw-bus-cycles	99.10
Device Description	L1-icache-load-misses dTLB-store-misses branch-load-misses branch-misses	99.01

Table 4: Accuracy and parameters chosen for the first three specialized detectors

a singular COPPA violation as opposed to the general COPPA compliance status of the app. After splitting the COPPA dataset into its features, LASSO or Ridge was applied to the HPC data. Then, K-Nearest Neighbors functioned as the specialized COPPA violation detector for each feature.

7.2 Results

The chosen HPC parameters and accuracy for each detector is shown in Tables 3 and 4. Since apps that share name, Google Services ID, or SIM serial number were not present in the dataset (as shown in Table 1), those COPPA parameters did not undergo feature reduction and were not utilized to develop specialized detectors.

COPPA Violation	HPC Parameters	KNN Accuracy
Permissions	dtLB-store-misses branch-misses raw-exception-taken raw-pc-write-return	99.79
Location	L1-dcache-store-misses branch-load-misses branch-store-misses branch-misses	99.23
Phone	L1-dcache-stores L1-icache-loads Raw-mem-access raw-l1-icache	99.40
Email	raw-exception-return raw-l2-dcache-wb raw-bus-access raw-l2-dcache-refill	99.68
Android ID	L1-icache-load-misses raw-l1-icache raw-bus-access raw-bus-cycles branch-loads branch-load-misses	99.00
Wi-FI	L1-dcache-stores branch-store-misses raw-mem-access raw-L1-icache	99.35
IMEI	L1-dcache-stores L1-icache-loads Raw-mem-access raw-l1-icache	99.65

Table 5: Accuracy and parameters chosen for the last seven specialized detectors

8 Discussion

Both the general and specialized COPPA detection methods yielded high accuracies. Due to its usage of hardware performance counters, these detectors have zero hardware overhead and can function independent of the operating system. Several metrics were examined for each classifier, including the false positive rate and the accuracy. If this software were to be implemented in real-time, threat modeling must occur in order to prioritize certain metrics and certain objects. For example, in a specific case, it may be riskier for the location to be disseminated than the Advertising ID. Thus, the system developed for that specific case would be more likely to make use of the detector for location violation than for Advertising ID.

This software can be incorporated at multiple points within the architecture of a mobile phone; it is likely to function best as a component of a new ROM from Android or third party developers. Security and privacy teams at Google and other platform managers could utilize this software as the first of a two-phase detection system where one of the proposed machine learning detectors would signal a more thorough and transparent manual analysis.

For further research, to resolve the issue of sub-optimal execution path with the Monkey tool, a tool similar to Monkey based upon reinforcement learning would be able to generate more veracious results. Similar methods to the ones proposed in the paper can be adapted for other privacy guidelines, including GDPR, which was recently ratified in the European Union. These classifiers should also be optimized for robustness and resource consumption in addition to being tested within the various layers of a mobile phone.

9 Conclusion

Children's apps are required to abide by the guidelines established by COPPA. However, a lack of an effective and efficient analysis technique enables the proliferation of COPPA violations by impeding enforcement. Thus, a new method that applies machine learning to data generated from hardware performance counters was developed. Several machine learning classifiers based upon a novel dataset indicate the presence of either a general or specific COPPA violation. These alerts are accurate and maintain a low misclassification rate, which augments the efficiency resulting from utilizing HPCs.

References

[1] (2017). *2017 Internet Crime Report*. Federal Bureau of Investigation: Internet Crime Complaint Center. pages 4

[2] Broadhurst, R. (2017). Cybercrime: Thieves, swindlers, bandits and privateers in cyberspace. *Social Science Research Network*. pages 4

[3] Chen, D., Vachharajani, N., Hundt, R., Li, X., Eranian, S., Chen, W., and Zheng, W. (2013). Taming hardware event samples for precise and versatile feedback directed optimizations. *IEEE Transactions on Computers*, 62(2):376–389. pages 5

[4] Chen, D., Vachharajani, N., Hundt, R., Liao, S.-w., Ramasamy, V., Yuan, P., Chen, W., and Zheng, W. (2010). Taming hardware event samples for fdo compilation. In *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, pages 42–52. ACM. pages 5

[Commission] Commission, F. T. https://www.ftc.gov/enforcement/rules/rulemaking-regulatory-reform-proceedings/childrens-online-privacy-protection-rule. pages 4, 7

[Das et al.] Das, S., Werner, J., Antonakakis, M., Polychronakis, M., and Monrose, F. Sok: The challenges, pitfalls, and perils of using hardware performance counters for security. In *SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security*. pages 6

[7] Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., and Stolfo, S. (2013). On the feasibility of online malware detection with performance counters. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 559–570. ACM. pages 5

[8] Egelman, S. (2018). ”our children’s apps aren’t directed at children.”. pages 4

[9] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. pages 11

[10] Gulmezoglu, B., Zankl, A., Eisenbarth, T., and Sunar, B. (2017). Perfweb: How to violate web privacy with hardware performance events. In *European Symposium on Research in Computer Security*, pages 80–97. Springer. pages 6

[11] Gupta, U., Babu, M., Ayoub, R., Kishinevsky, M., Paterna, F., and Ogras, U. Y. (2018). Staff: online learning with stabilized adaptive forgetting factor and feature selection algorithm. In *Proceedings of the 55th Annual Design Automation Conference*, page 177. ACM. pages 5

[12] Hans, C. (2009). Bayesian lasso regression. *Biometrika*, 96(4):835–845. pages 9

[13] Lu, K., Zhou, X., Bergan, T., and Wang, X. (2014). Efficient deterministic multithreading without global barriers. In *ACM SIGPLAN Notices*, volume 49, pages 287–300. ACM. pages 5

[14] Lu, K., Zhou, X., Wang, X.-P., Bergan, T., and Chen, C. (2015). An efficient and flexible deterministic framework for multithreaded programs. *Journal of Computer Science and Technology*, 30(1):42–56. pages 5

[15] Mushtaq, H., Al-Ars, Z., and Bertels, K. (2012). Detlock: portable and efficient deterministic execution for shared memory multicore systems. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 721–730. IEEE. pages 5

[16] Ragan, S. (2016). uknowkids.com responds to data breach, says proprietary ip also exposed. pages 4

[17] Reyes, I. (2018). Won't somebody think of the children. page 6383. pages 4, 6, 7

[18] Reyes, I., Wijesekera, P., Reardon, J., Elazari Bar On, A., Razaghpanah, A., Vallina-Rodriguez, N., and Egelman, S. (2018). Examining coppa compliance at scale. pages 6

[19] Singh, B., Evtyushkin, D., Elwell, J., Riley, R., and Cervesato, I. (2017). On the detection of kernel-level rootkits using hardware performance counters. In *Proceedings of the ACM Asia Conference on Computer and Communications Security*, pages 483–493. pages 5, 6

[20] Wang, X. and Karri, R. (2013). Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–7. IEEE. pages 5

[21] Weaver, V. M. (2013). Linux perf_event features and overhead. In *The 2nd International Workshop on Performance Analysis of Workload Optimized Systems, FastPath*, volume 13. pages 5

[22] Zhou, X., Lu, K., Wang, X., and Li, X. (2012). Exploiting parallelism in deterministic shared memory multiprocessing. *Journal of Parallel and Distributed Computing*, 72(5):716–727. pages 5