Program Counter as an Integrated Circuit Metrics for Secured Program Identification

Kofi Appiah, Xiaojun Zhai, Shoaib Ehsan,
Wah M Cheung, Huosheng Hu, Dongbing Gu,
and Klaus McDonald-Maier
School of Computer Science & Electronic Engineering
University of Essex, Colchester, UK
{*xzhai,kappiah,sehsan,wmcheu,hhu,dgu,kdm*}@*essex.ac.uk*

Gareth Howells
School of Engineering and Digital Arts
University of Kent
Canterbury
UK
W.G.J.Howells@kent.ac.uk

***Abstract*—Integrated Circuit Metrics is mainly concerned with the extraction of measurable properties or features of a given hardware device, capable of uniquely identifying the system's behaviour. This paper presents features that can be extracted from software executing on a device and identify the very software in execution. The main contribution of this paper is in two folds. The ability to extract features whiles the software is in execution as well as correctly identifying the software to divulge any tampering or malicious exploitation. Our aim is to use program counter values generated during program execution to train a *k-means* algorithm optimized for vector quantization, and later use the system to associate program counter values with an application.**

***Keywords*-Program Counter; SOM; Program Identification; ICmetrics;**

I. Introduction

Integrated Circuit metrics (ICmetrics) technology is based on the fact that electronic devices, depending on user inputs and uses, would normally function under unique conditions. Based on internal and external sensors, they sense different environmental condition, trigger the execution of different software, perform different tasks and even interact differently with different users. Similar to (*biometrics*) that are closely linked or can be used to characterized any particular human being, be it Iris, DNA or fingerprint; ICmetrics is the device equivalent for characterizing an electronic device.

This paper is focused on the extraction of measurable ICmetrics features capable of identifying a program in execution. The extracted features are robust enough for classification purposes. An interesting characteristic feature of the ICmetric used in this paper is the ability to extra while the application or software is still in execution. The feature is easy to extract and yet hard for an intruder to re-generate.

The paper is organised as follows. Section II is a review of previous classification algorithms and other related ICmetrics work. This is followed by the approach we have used in extracting measurable features in section III. Section IV describes the hardware setup used for experimental purposes. Details of all experiments conducted on the chosen algorithm with empirical and preliminary results are presented in section V, with conclusions and proposed future work in section VI.

II. Related Work

Papoutsis et. al. [1] present a novel technique for normalising sets of features, highly multi-modal for the purposes of generating encryption key based on ICmetrics. The work also demonstrates how cluster-finding techniques can be used with ICmetric to generate a secured encryption system. In [2] Kovalchuk et. al. demonstrate the use of ICmetrics in securing the transmission of sensitive medical data between patients and medical staff. The paper further discusses the advantages and challenges in using ICmetric as encryption key generator. A challenge exposed by the paper is the inability to use traditional pattern recognition techniques to classify ICmetric features.

The use of *PC* demonstrates that whiles some address spaces are only used by a particular application, several other applications share address spaces; making it difficult to distinguish between them. The aim of this paper is to:

- Use robust traditional pattern recognition techniques as proposed in [2] to classify *PC* values,
- Based on the classification technique used, be able to distinguish between programs that share the same *PC* address spaces and
- Use only *PC* sequence and frequency analysis as a strong ICmetrics candidate to dynamically identify an application.

Effectively, addressing most of the major challenges envisaged by previous research work conducted with the use of *PC* values as ICmetrics.

Changes in program execution may not be completely random [3], but rather falls into repeating behaviour. Huffmire and Sherwood [3] present a model for the automatic identification of program behaviour. They defined a phase as an interval (in time space) within a program that have similar behaviour. A phase is identified by analysing the execution history of the program. Rather than classifying data resides in memory (offline processing), Kasetty et. al

[4] introduced a framework for time series classification that can be implemented efficiently, updated in constant time and space, and deal with high data arrival rates. Their algorithm uses Symbolic Aggregate Approximation to summarize data in minimal space and yet captures the local behaviour in time space.

III. Our Approach

Our aim is to design a system with dynamic classification techniques capable of identifying an application or software as it executes on the hardware. Using the Program Counter (PC) as ICmetric feature, the system should be able to distinguish between applications and software that share address spaces. We also demonstrate the use of traditional pattern recognition techniques as our classification tool, a drawback in the work presented in [2]. We begin our algorithmic design with sample features and associated characteristics of interest, and further refine those features to achieve the aim of this work. The following sections describe the components of our system into details.

A. Signature Extraction

This section is mainly concerned with the interpretation of the raw PC values in a form that can uniquely identify any particular program or application. Depending on the platform and size of memory used, the PC values can be significantly huge. A typical execution of the angle conversion program [5] on the platform described in section IV can have PC values ranging from 134218632 to 134223590.

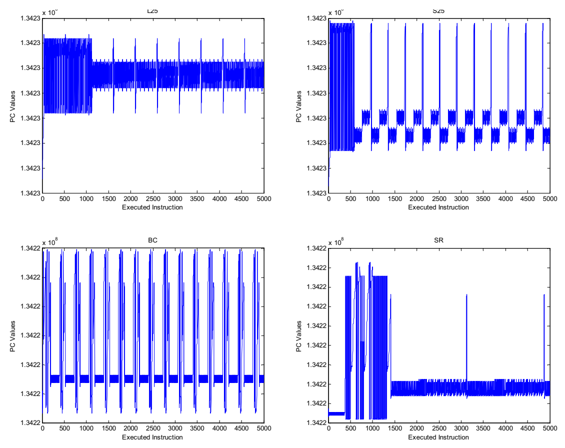

Figure 1. Sample output of Program Counter values from 4 different applications, each with $5,000$ samples. The variances for the applications are 607 for L25, $1.0e4$ for S25, $2.3e4$ for BC and $4.2e4$ for SR

As depicted in figure 1, each of the four applications have different access to memory (PC values) in terms of time and space. Thus, even when applications share or have overlapping PC values, the similarity might not occur at the same time during execution. We normalize the PC values into a set of continuous values ranging from zero to one (i.e $0 \leq \mathcal{K} \leq 1$), where $\mathcal{K}$ is the normalised representation for the PC value. Given $x_1, \ldots, x_k$ as the set of PC values $\mathcal{K}^i$ is expressed as

$$\mathcal{K}^i = \frac{x_i}{max(x_i)} \forall i \in (1 \ldots k) \quad (1)$$

where $max(x_i)$ is the maximum PC value.

B. Training Sub-System

After successfully representing the PC values in a form usable as our ICmetric, the next step is to design a system capable of distinguishing between different applications. At the minimum, the system should be able to infer to a degree of certainty that the extracted PC signature has previously been observed and belongs to a particular application. We refer to that as *True Positive*. When the signature has not been previously observed, the system is expected to throw a *True Negative*; thus an unknown signature. We also want to minimize the *False Negative* rate; situations where known signatures would be flagged as unknown, as well as *False Positive*, the assignment of unknown signatures to a known application.

A fundamental pattern recognition and clustering process, in which intrinsic inter- and intra-pattern relationships among the stimuli and responses are learnt without the presence of a potentially biased or subjective external inuence is presented in Kohonen's Self Organizing Map (SOM) [6], and would be adopted in this work as the basis for our classifier. We utilize the $k - means$ nature of the SOM, to partition the extracted PC signatures into a user-specified number of clusters, k.

Now that the clustering algorithm has been identified and defined, we will have to tune our ICmetric data (PC signatures), to conform with the required algorithmic inputs. The algorithm operates on a set of $N-$dimensional vectors for both inputs and network weights; where N is fixed for any particular network. The task here is to

1) Partition the PC signatures into N dimensional vectors,
2) The size of the vectors N, should be the same for all applications and
3) The vector should not over-fit or under-fit any application.

The size N of the vector, if set too large will over-fit the application that it represents. Similarly, if it is set too small, it will under-fit the application. Thus choosing the right size of N is very important. Again, if it is set too large it would require storing large volume of data before analysis can be conducted to infer the application in execution. That clearly undermines the purpose of this work. The best value of N, is the minimum number of PC signature values that represent one phase of the executing program. A phase is identified by analysing the execution history of the program. The value of N in this work has been set to 1000, after empirically examining the test data. Results of using other values will be discussed in section V.

PC values are grouped into sets of 1000 data points before converted into signatures. To enable continuous ICmetric analysis, the system presented here requires just 1000 PC values at a time to infer its corresponding application. Because the system is based on SOM – and a variant of the $k - means$ algorithm, the value of k should also be set. The value of k is set depending on the total number of software or applications under investigation, and the number of distinct phases in any particular application. At minimum, the value of k should be equal to the number of applications under investigation; thus when each of the application has no more than one repetitive phase. During the training phase, the entire PC values are divided into parts each of size 1000. Each part is then converted into the corresponding signature, $\vec{\mathcal{K}}^i$ for the normalised signatures and $\vec{\mathcal{B}}^i$ for the binary signatures. Typically, an application which generated 3000 PC values during execution will have three different normalised signatures ($\vec{\mathcal{K}}^1, \vec{\mathcal{K}}^2, \vec{\mathcal{K}}^3$) for training, each vector with size 1000.

During training, the vectors are presented individually to the network in a random order. Euclidean distances between the input vector $\vec{\mathcal{K}}^i$ and all the network neurons are computed and the neuron with the minimum distance is chosen as the winning neuron. The winning neuron is updated to reflect the input as described in equation 2. The same training data is used until there is little or no changes in the network values; thus when the network converges.

$$\Delta w_k = \eta(\mathcal{K}^i - w_k), \tag{2}$$

where $\eta \in (0, 1)$ is the learning rate and w_k is the winning neuron.

C. Recall and Identification

After training the k neurons with the input data, the next step is to associate each of the network neurons with an application. This is accomplished in this work by using Vector Quantization[7]. Its purpose is to reduce the cardinality of the representation space, in particular when the input data is real-valued. In this work, we use Vector Quantization to assign labels to the trained neurons (or cluster centroids) in the network as follows:

- Assign labels to all the input training data. The label is the identifier for the application from which the training data has been extracted from.
- Find the neuron (centroid) in the network with the minimum distance to the labelled input data.
- For each input data $\vec{\mathcal{K}}^i$ maintain the application label $\mathcal{A}_i$, the corresponding neuron w_j and the distance measured. The distance is maintained as a tie breaker for applications that share similar address space.

After estimating the corresponding neuron for each input, the steps involved in labelling the network neurons are as follows:

1) For each network neuron w_j, estimate the number of applications $\mathcal{A}_i$ (where $i \in 1 \ldots T$ for T different applications) that are associated with the neuron w_j.
2) If only one application $\mathcal{A}_k$ is associated with a neuron w_j, and the number of data points exceed 10% of the total number of application data points, the neuron is exclusively assigned to $\mathcal{A}_k$.
3) $\forall \mathcal{A}_i$ with more than 10% of its data points associated with w_j, create a codebook with an entry for the neuron w_j, and the corresponding applications $\mathcal{A}_i^j$, each with its distance range – thus minimum distance and maximum distance to the neuron.

Condition 2 is used to identify odd and unusual network neurons. Thus if less than 10% of a particular application's data points are associated with a neuron, the chances are that they are less frequent characteristics of that application. Condition 3 is very important when two or more applications share the same address space. If that is the case, then the distance measures for different applications would be different and it is used as the tie-breaker. The importance of this is demonstrated in section V.

IV. Hardware Setting

For this study, we have used the Keil MCB-STM32F200 evaluation board based on the STMicroelectronics STM32F207IG series of ARM Cotex - M3 processor. The platform has a 120MHz processor with 1MB Flash and 128KB RAM on-chip memory. This provides the attached host computer with unique stream of trace data in the form of executing profiling and code coverage. The PC values are first saved into appropriate files and then converted from their native Hexadecimal (Hex) into Decimal (Dec) for further analysis. The PC value extraction from the saved files and conversion from Hex to Dec is achieved with the use of MathWorld MATLAB. It should be pointed out that the data extraction is intrusive, and yet has little or no effect on the PC values, as the complete profile of the program execution is saved.

V. Experimental Results

To evaluate the performance of our system of identifying executing programs with their PC values, we have used seven different programs. Four of the seven programs have been taken from [5]; the automotive package of benchmark algorithms –MiBench. Two are taken from the Open Source Computer Vision Library (OpenCV [8]) and the last taken from [9]. All the seven programs have been implemented on the Keil MCBSTM32F200 evaluation board and their corresponding PC values extracted from the tracing log generated via the JTAG debugger. The programs taken from [5] are the Angle Conversion (AC) with 649141 PC values,

Prog	TP + TN(%)	FP(%)	FN(%)
AC	78.76	1.53	20.31
BC	96.89	0	2.67
CF	38.46	38.46	23.08
RN	100	0	0
SR	89.29	0	10.71
S25	100	0	0
L25	93.69	0	6.31
Tot	88.01	1.56	10.43

Table I
A TABLE SHOWING THE CLASSIFICATION SCORE FOR THE TEST PROGRAMS. THE VECTOR SIZE USED FOR THE NETWORK WEIGHT IS SET TO 1000.

the Bit Count (BC) with 448029 PC values, the Cubic Function (CF) with 51230 PC values and the Square Root (SR) with 279435 PC values.

The two programs chosen from [8] are the Sobel edge detector (S25) with 207671 PC values and the Laplacian edge detector (L25) with 222789 PC values. The last program is a random number generator (RN) with 54013 PC values. The PC values for all the seven programs have been divided into two equal parts for the training and test dataset. For example the training set for the AC program has 324570 PC values, which is further converted into 325 different feature vectors, each of size 1000. In table I, Prog is the program label and (TP + TN) represents all points that are correctly identified by the system. False Positive (FP) represents values that are incorrectly associated with wrong programs and False Negative (FN) are PC values which could not be associated with any program. Overall, the classifier can correctly identify programs with more than 85% certainty.

VI. CONCLUSION

We have presented a system of program identification with the use Program Counter as the ICmetrics data. The system uses an optimised k-means algorithm for the classification. We have also demonstrated the need to reduce the raw Program Counter values into normalised and binary forms. Analysis conductors has shown that the systems is capable of identifying an executing program with more than 85% certainty. The problem with the system is with the choice of the vector size. This can be resolved if another ICmetric can be used to determine when the execution changes from one phase to the other, this forms the basis of our future work.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the support of the UK Engineering and Physical Sciences Research Council under grant EP/K004638/1 and the EU Interreg IV A 2 Mers Seas Zeen Cross-border Cooperation Programme SYSIASS project: Autonomous and Intelligent Healthcare System (projects website http://www.sysiass.eu/).

REFERENCES

[1] E. Papoutsis, G. Howells, A. Hopkins and K. McDonald-Maier, *Integrating Multi-Modal Circuit Features within an Efficient Encryption System*, Third International Symposium on Information Assurance and Security, 2007.

[2] Y. Kovalchuk and K.D. McDonald-Maier and W.G.J. Howells, *Overview of ICmetrics Technology Security Infrastructure for Autonomous and Intelligent Healthcare System*, International Journal of u- and e-Service, Science and Technology, 2011.

[3] Huffmire, Ted and Sherwood, Tim, *Wavelet-based phase classification*, Proceedings of the 15th international conference on Parallel architectures and compilation techniques, 2006.

[4] Kasetty, Shashwati and Stafford, Candice and Walker, Gregory P. and Wang, Xiaoyue and Keogh, Eamonn, *Real-Time Classification of Streaming Sensor Data*, Proceedings of the 2008 20th IEEE International Conference on Tools with Artificial Intelligence - Volume 01, 2008.

[5] Guthaus, M.R. and Ringenberg, J.S. and Ernst, D. and Austin, T.M. and Mudge, T. and Brown, R.B.,*MiBench: A free, commercially representative embedded benchmark suite*, IEEE International Workshop on Workload Characterization, 2001.

[6] Yin, Hujun, *The Self-Organizing Maps: Background, Theories, Extensions and Applications*, Computational Intelligence: A Compendium,Studies in Computational Intelligence, vol. 115, 2008.

[7] Jegou, H. and Douze, M. and Schmid, C., *Product Quantization for Nearest Neighbor Search*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, 2011.

[8] OpenCV Developers Team, *Open Source Computer Vision Library*, March, http://opencv.org/, 2013.

[9] Y. Kovalchuk and W.G.J. Howells and H. Hu and D. Gu and K.D. McDonald-Maier, *A practical proposal for ensuring the provenance of hardware devices and their safe operation*, Proc. of Institute of Educational Technology System Safety Conference, 2012.