This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2020.2968260, IEEE Design and Test

Hardware-based Real-time Workload Forensics

Yunjie Zhang, Liwei Zhou and Yiorgos Makris
Electrical and Computer Engineering Department, The University of Texas at Dallas, Richardson, TX 75080, USA
E-mail:{yxz153430, lxz100320, gxm112130}@utdallas.edu

I. Introduction

As our reliance on network services and online applications increases, sensitive data are inevitably exposed to the threat of cyberattacks. Malicious software is designed to bypass security policies and compromise defense mechanisms, in order to launch Denial-of-Service (DOS) attacks or steal private data by taking advantage of vulnerabilities in system design. Accordingly, methods that are able to monitor program execution and identify suspicious behaviors are of great value. To this end, workload forensics collects and analyzes information to identify or reconstruct the behavior of a program executed in the past. Such solutions strengthen hardware security based on their ability to detect and track certain behaviors, including both malicious and benign ones. A successful workload forensics system should require as few physical and temporal resources as possible. Collecting and analyzing hardware-based features requires tracing CPU behaviors, which leads to increased design complexity. Moreover, the time spent in the analysis stage should, ideally, be limited to the extent that defense mechanisms can take immediate action against a possible malicious behavior before it finishes execution.

Current forensics solution can be further categorized into software-based and hardware-based methods. For the former type, various data-centric software analysis methods have become standard tools for forensic investigation in industry. For instance, EnCase ensures data integrity and enables data recovery by creating disk images [1]. However, because of the growing capacity of electronic devices and data volumes, these methods exhibit limitations in processing capability. Alternatively, instead of analyzing the data footprint left in storage media, program-centric methods focus on analysis of program behavior based on primitives such as system calls and system events. Software-based methods of this type generally employ statistical analysis and machine learning techniques to perform intrusion detection and workload forensics. Nevertheless, software-based methods could themselves be the target of a software attack. For example, sensitive variables used by forensics programs are stored in a memory area which should not be accessible by other programs, yet recently-developed attacks, such as Spectre, could allow attackers to compromise this barrier [2].

On the other hand, software execution traces cannot be hidden from the hardware as software has to be executed on hardware. Based on this premise, hardware-based forensics solutions have been developed. For instance, performance counter-based methods that monitor the frequency of a variety of system events and executed instructions have been proven to be effective in malware detection [3]. However, performance counter-based mechanisms that monitor execution of all instructions and system events lead to a relatively high data logging rate. Alternatively, forensics methods that perform analysis with compacted data collected from a complete process profile have also been proposed. While such methods require a drastically lower logging rate, they cannot respond promptly to a running intrusion until a malicious process finishes its execution.

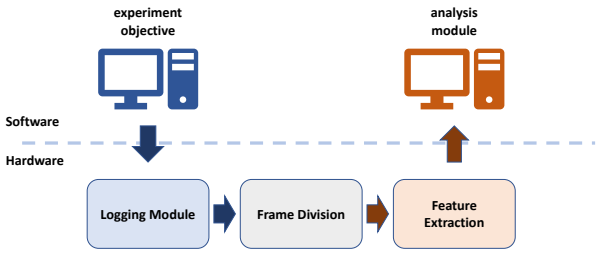

Figure 1: Overview of the proposed system architecture

In order to address the weaknesses mentioned above, we propose a forensics methodology which relies on compact information exclusively collected from hardware to identify a process, but which also enables real-time identification at any point during execution of a program, without requiring knowledge of process creation, switching or termination timestamps. To achieve this, we introduce a novel approach that utilizes system mode switching as a flag to divide a process into separate frames. Descriptive features can be extracted from each frame and further processed through machine learning algorithms to realize real-time process identification. In addition, hardware-based forensics requires process identification at the circuit level, without information from software execution or operating system status. This leads to a semantic gap problem, which requires bridging between hardware-based information, such as data stored in registers, and high-level software behavior. Earlier work resolves this problem by utilizing architectural conventions. For example, the CR3 register of an x86 machine that stores the base address of the page table of a process works as a proxy for process ID [4]. Any change in the value of this register corresponds to a crucial system event, such as process creation, switching and termination.

As we mentioned above, previous hardware-based methods incur high logging overhead. Inspired by the successful paradigm of utilizing Translation Lookaside Buffer (TLB) miss profiles in malware detection [5], we focus on such TLB features for our forensic analysis rather than using all system events. Moreover, besides realizing a single-frame process identification solution, we also introduce a majority voting strategy to improve process accuracy by combining results generated by multiple neighboring frames. Our method is evaluated using Spike, an open-source RISC-V simulator, running the MiBench testbench suite on a Linux operating system. Experimental results show that our system achieves overall accuracy of 98.9% with single-frame identifiers and 99.7% when majority voting strategy is applied. data-processing latency and frame run-time are also evaluated to demonstrate real-time feasibility of our solution.

II. Architecture Overview

As shown in Figure 1, our architecture comprises four main modules: logging module, frame division module, feature extraction module and analysis module. The hardware-based logging module collects TLB-related data from directly from the hardware. To enable real-time process identification functionality, we design a frame-division mechanism to split the instruction flow into frames. The feature extraction module continuously collects, then, descriptive features from each frame. The software-based analysis module which uses a machine learning-based strategy is built in a separate secure environment. Below, we introduce these components in detail to illustrate how real-time process identification is realized.

A. Logging Mechanism

To reduce logging overhead, we rely on only profiling instructions which cause TLB misses rather than the complete instructions flow. In modern computer architecture, a TLB is a cache which stores the recently-used translations of virtual to physical addresses. This translation mechanism helps flexible programming of application software without requiring knowledge of the specific memory hierarchy design of a CPU. Prior research shows that the average TLB miss rate is about 0.01-1%[6], which implies that profiling TLB misses induces a lower logging overhead to our forensics system, as compared with performance counter-based methods that monitor and log all changes of CPU status at the instruction level. In modern CPU designs, based on the cached content, a TLB can be further divided into two parts: instruction TLB (iTLB) and data TLB (dTLB). In this work, we only focus on instruction flow-related information, so we profile exclusively iTLB misses and disregard dTLB misses. Additionally, our analysis module only considers user-space instructions and disregards system-mode instructions, as the former normally better reflect information related to program behavior. To locate switching between user-mode and system-mode, we leverage the operating system convention that, in 64-bit Linux OS, virtual addresses lower than 0x0000 8000 0000 0000 are regarded as user space.

Our analysis module uses features extracted from frame-related data that occurs at any point during execution of a process, which means that knowledge of process creation, switching and termination timestamps is not required. However, for the purpose of training and testing our machine learning model, features need to labeled with a corresponding process ID. To this end, similarly to the previously mentioned CR3 register in X86 architectures, we use a RISC-V conventional register, $sptbr$, which holds the physical page number of the root page table and address space identifier to relate hardware information with program behavior. This naturally provides a solution to this problem as any change in $sptbr$ corresponds to a context switch in the OS. The logging module is able to label each frame with its process ID by logging related instructions and its corresponding $sptbr$ value. This establishes a semantic connection between hardware-level instructions and the workload to be reconstructed. The logged data can be divided into three parts: instructions that cause an iTLB miss, values seen by the $sptbr$ register and program counter values which can be used to distinguish user-space instructions from kernel-space instructions.

B. Frame Division

Our real-time workload forensic analysis is performed using features extracted at the granularity of a single frame. A uniform size is used for all frames to simplify the frame construction process. Here size refers to the maximum number of instructions contained in a frame. In Figure 2, to better explain our frame division strategy, we introduce the concept of "gap sequence", which encompasses user-level instructions that cause iTLB misses between two mode switches. The instructions of a gap sequence are pushed into the current frame if the whole gap sequence can fit into the remaining space of the frame. Otherwise, the current frame is passed to the analysis module and a new frame is generated. Another special scenario is when a gap sequence is too large to fit into a single frame. In this case, its instructions are used to construct as many frames as possible, until all instructions in this gap sequence have been handled. This frame creation strategy does not require any extra information about system events or any additional logging capability.

C. Feature Extraction

Feature extraction is critical for our analysis module, as features are expected to reflect both order and content of workload execution. Herein, we consider as our features the raw instruction sequences contained in frames that cause iTLB misses. Typically, a 64-bit RISC-V instruction set includes more than 200 types of operators and operands, which would make the feature space overly large for hardware implementation of feature collection. Therefore, we focus exclusively on operators and categorize them into 18 types based on semantics given by the RISC-V design specification. For each frame, a feature vector is extracted and a list of vectors is collected from each process. The uniform size for all frames is a crucial parameter of our method indeed, since

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2020.2968260, IEEE Design and Test

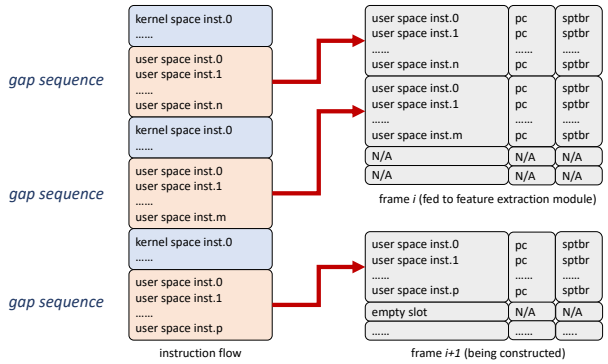

Figure 2: Frame construction in hardware

our analysis module utilizes machine learning, frame size will impact classification performance. A small-sized frame containing less information may have a negative impact on the overall process identification accuracy. On the other hand, a large-sized frame increases the complexity of the analysis module and requires more hardware resources for the logged data. Thus, an optimal unified size of all frames is sought experimentally in our work, in order to balance performance and overhead.

D. Analysis module

The analysis module consists of three main steps: frame identification, majority voting and outlier detection. A basic frame identifier is required as our workload forensics is performed at the granularity of a single frame. This fundamental identifier is designed to perform multi-class classification based on the features extracted from each frame, where each class corresponds to a single process. Similar classification problems can be found in Natural Language Process (NLP) tasks, such as machine translation and speech recognition. Recurrent Neural Networks (RNN) have been widely applied in NLP research areas, as they can efficiently handle temporal sequential input by storing previous status of neural networks. Two types of RNNs with gating units, namely Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs), have been developed to address these tasks. While there is no significant difference observed in the performance of these two models, GRUs-RNN may have a slight advantage in computational time [7]. Thus, the latter model is selected in our work to save computational resources during the training stage.

Moreover, in order to further improve the accuracy of our forensics methods beyond the abilities of a single frame classifier, a majority voting strategy has been employed to combine multiple frame identification results. This method improves the overall accuracy by suppressing sporadic frame prediction errors through the decisions of neighboring frames. In case of a tie, our method picks the earliest frame identification result.

E. Outlier Detection

Frames from unseen processes can be identified through outlier detection. We leverage the fact that the *softmax* output layer of our neural network model returns a vector of probabilities of frame classes. If a frame has never been seen in training, it is less likely to produce a dominating likelihood in any one of the target classes. This property of ambiguity is, then, utilized to perform outlier detection.

III. Experimental Results

We evaluate our process identification method and its data logging overhead using Spike, an open-source RISC-V simulator. We configured Spike to work with the 64-bit RISC-V instruction set, we installed a 64-bit Linux kernel with the necessary applets and we used the MiBench testbench suite for our experiments. However, due to the on-going development of RISC-V compiler library, some MiBench applications cannot be properly compiled and were, therefore, excluded from our experiments. Binaries were executed with different arguments and in random order so as to eliminate any possible bias introduced by program execution order. As an open-source simulator, Spike provides us with great flexibility in implementing our data logging and feature extraction modules. Specifcially, we embedded an iTLB tracer in the MMU class of the Spike simulator to track TLB accesses and to log the TLB profile. Each time an iTLB miss occurs, another modified function is called to log the user-level instructions that cause iTLB misses, along with their corresponding program counter (PC) values, before they are executed. We also added a counter to record the total number of instructions executed while each frame is constructed, in order to calculate the logging rate. Data analysis and result evaluation is performed with TensorFlow 1.10.

A. Process Identification Performance

To evaluate the performance of frame-level process identification, a dataset containing iTLB miss profiles from a total of 71 processes has been logged for both training and testing. More specifically, a total of 59246 frames were generated by our frame division strategy with uniform frame size set to 30, and each frame was labeled with its corresponding process ID. For each process, 60% of the samples were randomly selected as training set while the rest were used as testing set. The results of single process identification are shown in Table I. As can be observed, the GRUs-RNN exhibits excellent process identification performance, reaching an overall accuracy of 98.9%. These results not only outperform previous post-execution methods [5] but can also be performed in real-time.

To explore the impact of frame size on identification accuracy, we conducted the same experiment with various frame sizes ranging from 16 to 34. Based on the results shown in Figure 3, which show that the overall process identification accuracy starts to increase after frame size exceeds 22, but stops improving once it reaches 30, we adopted the latter as our optimal frame size.

B. Majority Voting

Next, rather than relying on a single frame for identifying a process, we proceed to evaluate the effectiveness of a majority

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2020.2968260, IEEE Design and Test

Table I: Frame-level process identification results on RISC-V

application class	training samples	testing samples	accuracy
overall	34367	24879	98.9%
crc	2557	1705	99.6%
fft	3225	2150	99.2%
qsort	3125	2084	99.4%
toast	3624	2428	99.4%
search	3676	2450	99.4%
bitcnts	2784	1864	98.7%
untoast	2670	1782	99.4%
dijkstra	2538	1692	98.4%
patricia	2857	1904	99.3%
basicmath	2934	1956	96.0%
bf	3576	2384	100%
susan	3704	2480	98.0%

voting strategy, which makes decisions based on multiple successive frames. To this end, we use the collected in-order execution profile and applied this strategy for a range of different latency values. Latency here refers to the number of frames that we rely on to make a majority-vote decision. As shown in Figure 4, the overall accuracy improves significantly from 98.9% to 99.7%, when latency is set to 7.

C. Outlier Detection

In order to detect previously unseen processes, we utilize the probability estimations provided by the *softmax* layer of GRUs-RNN, which reflects the likelihood that a frame comes from certain known classes. What has been observed in the experiment is that frames from a seen process come with dominating probability values. However, for frames from previously unseen processes, no class exhibits a dominant likelihood. An example of likelihood distribution is shown in Figure 5 for each of above cases. It can be observed that predicted outputs of seen processes are more likely to fall into ranges from 0.7 to 1.0, while unseen processes are prone to be predicted with probabilities between 0.2 and 0.5. Based on these results, we can screen outlier processes by setting a minimum threshold for the highest probability given by the output *softmax* layer of GRUs-RNN. Any frame where no predicted likelihood exceeds this threshold can be regarded as belonging to an unknown process.

Experiments have been conducted for multiple iterations to eliminate possible bias. Each time, a randomly selected 75% of processes are used as seen processes while the remaining 25% are kept as outliers (i.e., previously unseen) processes. Based on our experimental results,0.65 is selected as the optimal threshold for accepting the highest likelihood. Results from 5 random iterations with threshold 0.65 are summarized in Table II. In our study, outliers are defined as the positive class. Thereby, the False Oositive (FP) rate indicates seen process predicted as outliers, while the False Negative (FN) rate reflects outliers that are classified as seen processes. As can be observed from the shown results, our straightforward method of outlier detection achieves reasonably accurate results, with average FP and FN rates of 14.47% and 7.95% respectively. While better detection accuracy can be achieved by applying a separate binary classifier or with more advanced machine

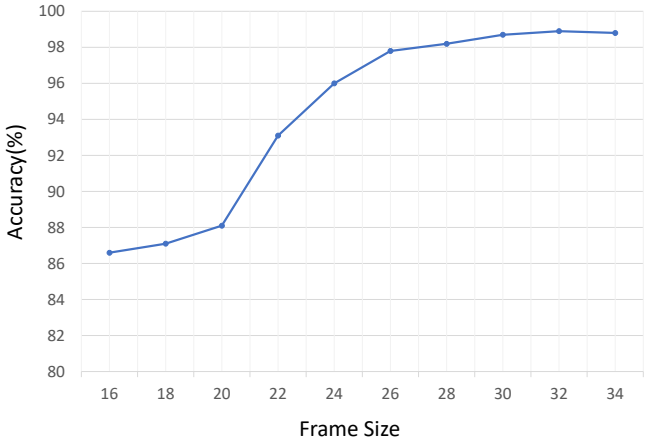

Figure 3: Process identification accuracy vs. frame size

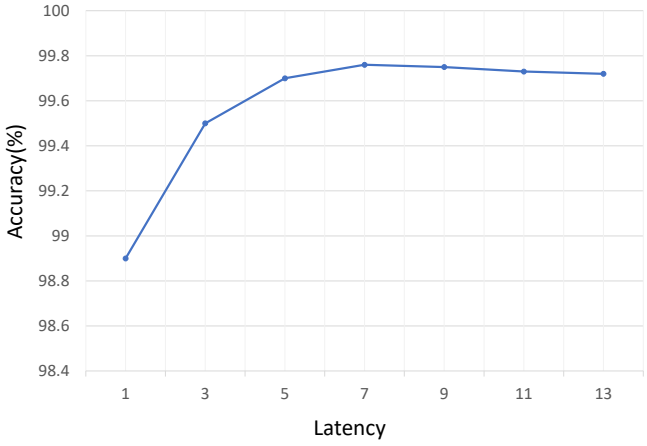

Figure 4: Process identification accuracy vs. latency

learning models, our method relies on the existing analysis model and does not require additional processing. This is particularly important as any added overhead could jeopardize our ability to perform this analysis in real-time, along with our future research direction of implementing the entire forensic system on chip.

D. Logging Rate

Logging rate refers to the amount of data that needs to be passed to the analysis module per unit time for processing.

Table II: Summary of FP/FN rates in outlier detection

test #	No. of seen frames	No. of outlier frames	FP rate	FN rate
average			14.47%	7.95%
test 1	19501	5378	15.74%	6.96%
test 2	18137	5146	14.55%	8.46%
test 3	18940	5939	11.26%	9.90%
test 4	17632	6261	18.38%	5.57%
test 5	18097	6782	12.41%	8.87%

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2020.2968260, IEEE Design and Test

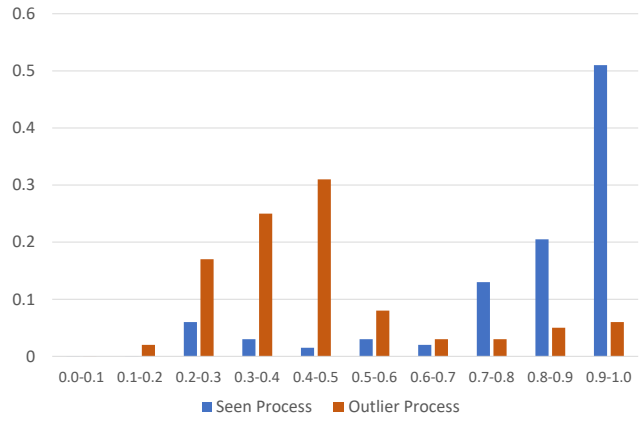

Figure 5: Probability distribution of top classes

This metric can be used to evaluate the overhead introduced by the logging and feature extraction modules. Higher logging rate requires higher processing capability and potentially a storage buffer if collected data cannot be processed at the rate of arrival. To compute the logging rate of our frame-level analysis, two critical values have been collected in our experiments, namely *frame generation rate* and *frame feature size*. The former indicates how many frames are generated per clock cycle and the later refers to the minimum number of bits required to represent features extracted from each frame. The frame feature size is the product of optimal frame size and bits needed to represent a, operator category, which are 30 and $\log_2 18$ respectively. Our experiments show that the average value of frame generation rate is 2.78×10^{-6}. The equation for estimating logging rate is shown below:

$$Logging\ Rate = Frequency \times Generation\ Rate \times Feature\ Size \times CPI \quad (1)$$

Cycles per instruction (CPI) is estimated to be 1 for modern CPUs. Assuming a 1.3Ghz clock frequency reported in a RISC-V prototype, our logging rate is estimated to be about 66.1 KB/sec, which outperforms the performance counter-based method [3].

E. Analysis Latency

The time required for analyzing the collected data should not exceed the time for constructing a frame, as our system performs continuous process identification. While our analysis module is implemented in software and runs on a separate system where the logged data is passed to and analyzed by, real-time processing is still feasible. Even without the usage of any custom hardware accelerator, such as a GPU, the time needed for analysis is much less than the frame construction time. In our experiment, the average time for constructing a frame is 0.277 ms, while the single frame identification delay and majority vote decision delay are 0.0679 ms and 0.00146 ms, respectively. Data transmission delay is estimated based on the results described in a survey [8] where less than 0.01 ms of one-way latency is introduced at a bandwidth of 2.1Gbps when TCP/IP Ethernet is used. As can be calculated, all delays combined are still much shorter than our frame construction time, which corroborates feasibility of real-time process identification.

IV. Conclusion

Our work explores feasibility of hardware-based real-time workload forensics. Unlike software-based approaches, the proposed method is immune to software attacks, as it does not involve data collected from the OS or software applications. Features extracted directly in hardware from instructions that cause iTLB misses are used to construct frames which are further analyzed through trained machine learning models to identify the running processes. Our experiments, which were conducted on Spike, a RISC-V ISA simulator, show an overall process identification accuracy of 99.7% when majority voting is employed on successive frames.

References

[1] S. Widup, *Computer forensics and digital investigation with EnCase Forensic v7*, 2014.
[2] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *arXiv:1801.01203*, 2018.
[3] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013, pp. 559–570.
[4] S. T. Jones, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau *et al.*, "Antfarm: Tracking processes in a virtual machine environment." in *USENIX Annual Technical Conference*, 2006, pp. 1–14.
[5] L. Zhou and Y. Makris, "Hardware-based workload forensics: Process reconstruction via tlb monitoring," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2016, pp. 167–172.
[6] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design*, 2007.
[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.
[8] S. Larsen and B. Lee, "Survey on system I/O hardware transactions and impact on latency, throughput, and other factors," in *Advances in Computers*, 2014, vol. 92, pp. 67–104.

Yunjie Zhang is pursing a Ph.D. in Electrical and Computer Engineering at The University of Texas at Dallas. His research interests include application of machine learning in workload forensics and malware detection. He is a student member of IEEE.

Liwei Zhou holds a Ph.D. in Electrical and Computer Engineering from The University of Texas at Dallas. His research interests include hardware security and reliable ASIC design.

Yiorgos Makris is a professor of Electrical and Computer Engineering at the Erik Jonsson School of Engineering & Computer Science at The University of Texas at Dallas. He holds a Ph.D. in Computer Engineering from the University of California, San Diego. He is a senior member of IEEE.