Real-Time Anomalous Branch Behavior Inference with a GPU-inspired Engine for Machine Learning Models

Hyunyoung Oh[1], Hayoon Yi[1], Hyeokjun Choe[1], Yeongpil Cho[2], Sungroh Yoon[1] and Yunheung Paek[1]
[1]ECE and ISRC, Seoul National University, [2]Soongsil University
{hyoh, hyyi}@sor.snu.ac.kr, genesis1104@snu.ac.kr, ypcho@ssu.ac.kr, {sryoon,ypaek}@snu.ac.kr

***Abstract*—Attacks on embedded devices are likely to occur any time in unexpected manners. Thus, the defense systems based on fixed sets of rules will easily be subverted by such unexpected, unknown attacks. Learning-based anomaly detection may potentially prevent new unknown zero-day attacks by leveraging the capability of machine learning (ML) to learn the intricate true nature of software hidden within raw information. This paper introduces our work to develop an MPSoC, called *RTAD*, which can efficiently support in hardware various ML models that run to detect anomalous behaviors on embedded devices in a real-time fashion, and thus enable the devices to counteract the anomalies in the field. In the IoT era, the importance of security for embedded devices cannot be exaggerated because they will become an enticing target for adversaries as they are being integrated into everyday life to provide users with various services. The above-mentioned potential of learning-based detection is believed to benefit those deployed devices under attacks occurring any time during their field operations in unexpected manners. We hereby assume that ML models are trained with runtime branch information as their data features since a sequence of branches serves as a record of control flow transfers during program execution. In fact, there have been numerous ML studies that examine various types of branches in order to infer (or detect) anomaly in branch behaviors that may be induced by diverse attacks that can cause deviant control flow in software. Our goal of real-time anomalous branch behavior inference poses two challenges to our development of RTAD. Firstly, RTAD must collect and transfer in a timely fashion a sequence of branches as the input to the ML model. Secondly, RTAD must be able to promptly process the delivered branch data with the ML model. To tackle these challenges, we have implemented in RTAD two core components: an input generation module and a GPU-inspired ML processing engine. According to our experiments, RTAD enables various ML models to infer anomaly instantly after the victim program behaves aberrantly as the result of attacks being injected into the system.**

***Index Terms*—anomaly detection, machine learning, embedded system, runtime security**

I. Introduction

With the resurgence of machine learning (ML) in recent years, renewed interest is given to applying ML to solve diverse computer security problems where rule-based or deterministic algorithms have shown inherent limitations. The main attraction of applying ML to security is its capability to learn from data a model representing the behavior of a system or program which would otherwise be needed to be arduously developed by hand. Furthermore, ML also grants the possibility for the model to unravel and learn the intricate nature of a program which is hidden within raw information and thus impossible in practice to be unriddled by a set of man-made rules. One of the primary security applications leveraging these strengths of ML is *anomaly detection* [1]–[8], whose goal is to recognize aberrant executions caused by attacks, misconfigurations, bugs and eccentric usage patterns. With its capability to define normal states from given normal data, ML has been considered a natural fit for anomaly detection by many studies where a normal model is commonly constructed to compare against the current runtime behavior and discover any discrepancies characterizing abnormal behaviors. To detect such anomalies with unusual behaviors, ML models take as input a set of feature values representing the current runtime behavior of a program and test whether the input is normal or abnormal. This test procedure is usually referred to as *inference*. The merit of this learning-based anomaly detection over conventional rule-based security solutions is its independence from attack signatures which might be easily modified by attackers to dodge detection [9], [10]. Furthermore, this attractive attribute of ML could potentially help to proactively prevent new and unknown zero-day attacks.

In the IoT era, the importance of security for embedded devices cannot be exaggerated because they will become an enticing target for adversaries as they are being integrated into everyday life, thus storing and processing personal information to provide users with various services. The aforementioned potential strength of learning-based anomaly detection solutions is believed to benefit embedded devices in that attacks on these devices tend to occur any time during their field operations in unexpected manners, and thus the conventional defense systems based on fixed sets of rules will easily be subverted by such unexpected, unknown attacks. We also believe that such solutions would more benefit embedded devices if they could infer anomaly in a *real-time* fashion because for a certain device deployed in the IoT environment, inference speed is just as equally important as accuracy to its successful mission. For instance, upon receiving a report of anomaly in the system, a mission-critical device (e.g., unmanned vehicle) may be able to counteract anomalies quickly in the field and continue its mission without interruption. To this end of our real-time detection (or inference), we have developed a multiprocessor SoC (MPSoC), called *RTAD*, which is designed to support in hardware the online inference task of a variety of ML models that have been trained with records of normal runtime behavior of programs. We assume in our work the branch information as the records used in training ML models since it is widely regarded that a sequence of branches (i.e., control flow transfers) serves as a record of program behaviors at runtime. In fact, there have been numerous ML studies that examine various types of branches, ranging from those with specific purposes (e.g., system calls) to general ones, in order to infer anomaly in branch behaviors that may be induced by diverse attacks, such as control flow hijacking or data only attacks, that can cause deviant control flow in software.

However, in order to support ML models for real-time anomalous branch behavior inference, there have been two challenging requirements to be addressed in the development of RTAD. First, to meet our development goal, RTAD is required to collect and transfer in a timely fashion a sequence of branches as the input to the ML model. This requirement is challenging due to the fact that as branches in the real code do occur fairly frequently, it will immensely slowdown the host system (up to 167% with software instrumentation [4]) to collect branch events and transfer a colossal amount of branch information to the ML model. Recent work [7], [8] gives a glimpse of promise in handling this requirement by employing hardware support, such as Intel Processor Tracing (PT), for

collecting runtime branch outcomes. However, though their hardware may facilitate an efficient collection of runtime branch data, it alone cannot suffice the first requirement for RTAD since it lacks a mechanism for a fast transfer of the collected data to ML models. In our work, RTAD has been implemented in hardware to fully meet the requirement. For this purpose, RTAD is equipped with a dedicated hardware module, called *input generation module* (IGM), which gathers runtime branch outcomes inside the CPU on the fly and quickly transforms them into vectors which are then fed as inputs to the ML model running for anomaly inference.

The other requirement for the development is that the ML model running on RTAD must be able to promptly compute and perform inference on the delivered branch data without significant delay. The natural approach for this would be to implement a high-performance accelerator engine for ML model computation. In order to help RTAD run diverse ML models in software, we have designed the engine to be programmable. As a prime candidate architecture for our programmable engine, we opted for a GPGPU due not only to its programmability, but also to its excellent parallel processing capability that would be instrumental to fully utilizing the high degree of parallelism inherent in most ML models for speed up. Capitalizing on the GPGPU's versatility to accept software instructions, RTAD would easily support various ML models with the same hardware engine. In our early design, we employed an open-core GPGPU *MIAOW* [11]. However, in the preliminary experiment, we have found that MIAOW is designed to be too general-purpose to yield optimal performance for certain special-purpose operations like our ML computations. More specifically in our original implementation, MIAOW was not fast enough to catch up with the speed of branch outcomes that IGM generates especially when a branch-heavy application was running, ending up with its internal input buffer being overflown. In order to tackle this performance problem, we could choose a straightforward strategy where we boost up the computing capability by adding more *compute units* (CUs) to the original implementation so that we can process in parallel more incoming branch outcomes. However, such a straightforward strategy to enhance performance was not a plausible option for our RTAD since we target embedded devices that are mostly subject to severe resource constraints.

Alternatively, we adopted a different approach where we build a variant of the MIAOW architecture, called *ML-MIAOW*, customized for ML operations by transforming the excessive GPGPU-style hardware into more compact application-specific one. ML-MIAOW is inspired by the strength of GPGPU in terms of programmability and parallel processing in a sense that it basically works as a GPGPU like MIAOW except its datapath optimized for ML operations. We have built ML-MIAOW by eliminating logic elements unnecessary for ML operations while maintaining the core datapaths needed for software programmability. Our experiments reveal that ML-MIAOW attains 5x performance-per-area, meaning that its area is just about 1/5 of that of MIAOW, yet achieving the same performance. Since ML-MIAOW and MIAOW both have virtually the same core circuits like pipeline stages and ALUs, ML-MIAOW can also support the same runtime environments as MIAOW, thus facilitating accommodation of existing ML models designed to run on a GPGPU.

To ease the deployment of RTAD in SoC-based embedded devices today, we endeavor to comply with state-of-the-art design rules of SoC. Our hardware IPs are placed outside and connected to the host CPU core to build the target SoC. RTAD is basically an MPSoC combining two heterogeneous processing elements: the CPU and *ML processing unit* (MLPU). We choose an ARM processor as our CPU since ARM has been a dominant player in the embedded CPU market for years. MLPU consists of two core modules, IGM and *ML computing module* (MCM). Our ML-MIAOW is the main component of MCM and control logics for operating ML-MIAOW are also included within MCM. To evaluate RTAD, we have deployed several ML models on an FPGA-based prototype and found that thanks to RTAD's support, they can infer an attack from branch data as early as within just about 24μs after the attack initiates an attempt to divert the branch behavior of a victim process running on an ARM device, yet attaining a performance improvement of 2.75x on average over the original MIAOW as an inference engine.

II. Related Work

To our knowledge, this is the first work that builds a complete MPSoC on an ARM device to efficiently support real-time anomalous branch behavior inference. RTAD has several distinctive merits over previous work. Firstly, RTAD is able to support many different ML models whereas others support fixed models. Thus in the RTAD SoC, users may realize and deploy several models at their disposal, trying diverse types of branches as data features. Secondly, our system can be applied to existing software environments established for today's ARM system since in our RTAD SoC, an ARM processor can be integrated with other hardware IPs for anomaly detection. As stated earlier, the goal of our work is to provide a system that could efficiently support anomalous branch behavior inference and therefore we consider the numerous work in this area focusing on developing ML models to be orthogonal to our work. The studies closely related to RTAD are those that took into consideration the performance of inference and its data collection process. Ozsoy et al. [12] proposed *malware-aware processors* (MAPs). Their work was motivated by the results shown by Demme et al. [13]. MAPs have a hardware-based real-time detector that differentiates malware from legitimate programs. Rahmatian et al. [14] proposed a host-based intrusion detection solution that detects malicious software in embedded systems. To characterize the benign program behavior, they implemented an FSM to model the possible system call sequences occurred during the program execution. To extract the system call sequence, they modified the internal microarchitecture of a SPARC3 Leon processor. Das et al. [5] proposed GuardOL, a hardware architecture to perform real-time malware detection. They have modified the x86 internal architecture to extract system calls and to construct features necessary for GuardOL's ML algorithm. Although all these hardware-based studies attain their goals with remarkably low performance overhead, their solutions cannot be applied directly to real embedded systems running on ARM, unlike RTAD.

In the latest work [7], [8], ML models can work with branch data collected from the Intel PT. However, as their focus was in developing a model that works well with branch data, they have only employed hardware to efficiently collect branch information. RTAD considers the real-time branch behavior inference problem and designs hardware modules to augment such branch collection hardware support for this end. Duarte et al. [15] discussed a general approach where the MIAOW architecture can be trimmed for specific applications by eliminating unnecessary hardware, but they did not specifically consider anomaly detection or other security applications in their work. In particular, they discussed optimizations for a single fixed application. On the other hand, in our work, we consider simultaneous trimming for multiple applications by merging the minimum required logics of several different ML models.

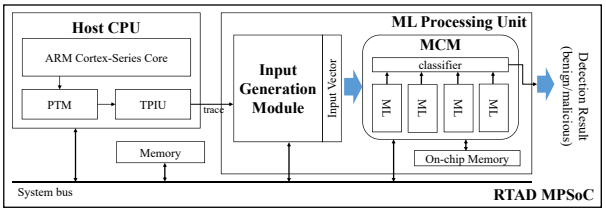

Fig. 1. RTAD architectural overview

Furthermore, we eliminate additional unnecessary logics by analyzing code coverage as will be discussed in Section III.

III. RTAD Architecture

Fig. 1 shows the overall architecture of RTAD where an off-core MLPU is integrated together with the host CPU and other peripheral IPs. As shown in the figure, the host CPU of the RTAD architecture is an ARM Cortex-series processor which is the de-facto standard processor deployed in commercial smart devices these days. The host CPU and MLPU are connected with a shared main memory via the ARM NIC-301 bus, a standard AMBA3 AXI interconnect. It is noteworthy that we have tried to design all modules in accordance with the standard protocols and specifications of the current, up-to-date ARM-based MPSoCs.

In our system, we have designed IGM to receive the branch information of a running program inside the CPU through the ARM CoreSight PTM and TPIU, as being inspired by previous studies for different purposes [16], [17]. PTM provides similar support in collecting runtime branch outcomes as Intel PT employed in recent work [7], [8]. However, as stated in Section I, PTM alone does not fulfill the performance needs of RTAD and therefore we design IGM to augment the support provided by PTM. Upon receiving information through PTM and TPIU, IGM refines it to generate input vectors that are given as input to ML models running in MCM. MCM takes the outputs of IGM and makes transactions conforming to the ML-MIAOW input protocol. In the subsequent subsections, we give detailed descriptions of the hardware modules in RTAD.

A. Input Generation Module

IGM overview: As illustrated in Fig. 2, IGM receives the ARM CoreSight PTM traces as input and generates the input vector after decoding the branch address that is generated during execution of the target application. PTM is the key module of CoreSight that captures diverse types of debug information generated by programs running inside the ARM CPU, such as branch target addresses, exceptions, instruction set mode changes (ARM/THUMB) and current process IDs. After compression, the generated trace stream is routed to TPIU, and finally forwarded to the off-chip pins to provide the external peripheral modules with the runtime branch information of host programs. In the current implementation (Fig. 1), the output signals of TPIU are directly routed to the on-chip ports of MLPU instead of the off-chip pins so that we can utilize the CoreSight modules within the RTAD MPSoC. To activate the functionalities of PTM and TPIU, we have also built a device driver running on the Linux kernel.

Trace analyzer: The main submodule in IGM is the *trace analyzer* (TA) that receives the trace stream through a 32-bit port and decodes it to extract branch target addresses. Because the trace stream is constructed of multiple packets of one or more bytes of data, decoding for each packet must be done sequentially in bytes. TA has four *TA units* responsible for each byte decoding. Since the incoming 32-bit input can be decoded into four branch addresses in the worst case, we install the *parallel-to-serial converter* (P2S) between TA and

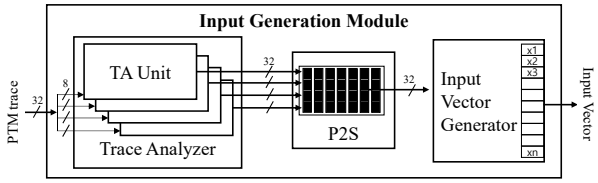

Fig. 2. Block diagram of IGM

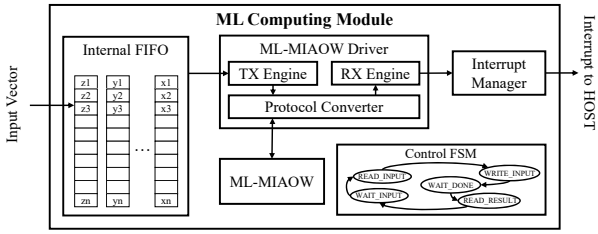

Fig. 3. Block diagram of MCM

input vector generator (IVG). IVG transforms a sequence of branch addresses into an input vector format suitable for use in the inference process of MCM. IVG is largely divided into two sub-blocks: the *address mapper* and *vector encoder* (VE). The address mapper lets only the relevant branch addresses be passed by filtering out the addresses not existing within a lookup table. Users can configure the table to select branches related to their ML models, such as system calls or critical API function calls which are used in many previous anomaly detection algorithms [4], [5]. The filtered address values are transferred in real time to VE as input and then converted into vector format following a conversion table that can be configured to match the need of target ML models.

B. ML Computing Module

MCM overview: The code running on the host CPU manages memory on both the host and peripheral, and also launches *kernels* which are functions that can be executed in parallel on the peripheral. Before executing a kernel on ML-MIAOW, all the data used by the kernel must be transferred from the host memory to the peripheral memory. After execution, the data produced by the kernel most likely needs to be transferred back to the host memory. Then, the host CPU continues operations with the copied results. For data transmission, in the base hardware architecture of ML-MIAOW, it has an AXI bus interface through which bus masters can deliver data to ML-MIAOW. When the data is delivered via the interface, ML-MIAOW stores the data in its internal memory. ML-MIAOW then uses the stored data for its operation. In order to fully utilize RTAD modules, a hardware component is necessary to quickly convert the output of IGM to the input protocol of ML-MIAOW. In this regard, we design MCM as shown in Fig. 3.

The *control FSM* contains configuration registers and controls the operation of the *ML-MIAOW driver*. The *TX engine* and *RX engine* are responsible for sending data to ML-MIAOW and getting data from ML-MIAOW, respectively. The *protocol converter* is used to convert the TX/RX data to the protocol required by ML-MIAOW. The *interrupt manager* is responsible for generating an interrupt to the host CPU. In the initial `WAIT_INPUT` state, MCM waits for the output of IGM to come. When the input vector arrives at the *internal FIFO*, the FSM state transits to the `READ_INPUT` state. The vector value is temporarily stored in the internal FIFO, and the TX engine reads the stored value. Then, the FSM state is changed to the `WRITE_INPUT` and the TX engine makes write transactions to drive input to ML-MIAOW. At the same time, control registers for each CU such as starting addresses

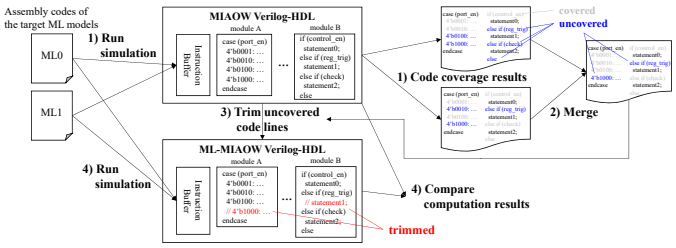

Fig. 4. Trimming MIAOW into ML-MIAOW

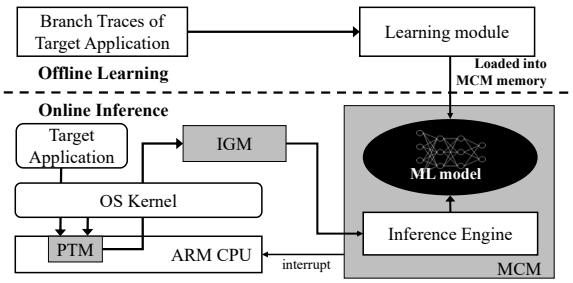

Fig. 5. RTAD anomaly detection procedure

of register files and local memory are also set. Then, the TX engine triggers ML-MIAOW to start computing for the inference. The FSM state then transits to the `WAIT_DONE` state and waits for the ML computation to end. When the computation is finished, the RX engine reads the results from ML-MIAOW after transiting to the `READ_RESULT`. If the results indicate the existence of an anomaly, the interrupt manager fires an interrupt to the host CPU.

ML-MIAOW: At the center of our MCM lies ML-MIAOW, optimized from the open source MIAOW processor which is available in the RTL form and prototyped in FPGA. MIAOW is basically a GPGPU implementing a subset of AMD's Southern Islands ISA. MIAOW supports the OpenCL programming model widely used for general heterogeneous parallel computing. Our ML-MIAOW naturally inherits this characteristic of the original MIAOW while having significant performance merit over MIAOW. As we mentioned earlier in Section I and II, we trimmed unnecessary logics from the original MIAOW to improve performance-per-area. This area saving can bring not only power efficiency but also more computation power by increasing the number of CUs without demanding more space. The trimming process, depicted in Figure 4, is as follows:

1) Run dynamic simulations for the target ML models with turning on the option for code coverage indicating which lines of the MIAOW HDL code are actually hit during simulation (e.g., conditional branches or items within case statements).
2) Merge code coverage results of each simulation.
3) Identify uncovered code lines, which would represent circuits not required for computing the ML models, and trim them out.
4) Verify whether the trimmed code operates correctly by comparing its computation results with those from the original MIAOW.

This process allows us to efficiently and thoroughly trim MIAOW and leave only the circuits needed for computing the target ML models, greatly improving performance-per-area. We employ Cadence Incisive Enterprise Simulator as our dynamic simulator and ICCR for merging and analyzing the coverage results.

C. Anomaly Detection with RTAD SoC

As stated in Section I, RTAD is an MPSoC designed to provide support for ML models performing inference on runtime branch behavior. RTAD can help to collect data for training models by running the target application in advance and extracting the branch traces generated by the target application for various inputs using IGM. A model would then be able to learn from the collected traces, effectively modeling the expected branch behavior of normal program execution. Once learning is finished, the model is employed by the inference engine running on MCM to infer attacks on the target application. When the target application is loaded by the OS kernel, the corresponding model is also loaded into MCM's memory. The inference engine uses the model to detect the existence of an anomaly by monitoring the actual behavior exhibited while the target application is running. For inference, the branch traces emitted from PTM are transformed into input vectors encoded form by IGM. The transformed trace is then delivered to MCM and the ML-MIAOW driver sends a start command to ML-MIAOW. Upon receiving this start command, ML-MIAOW executes the inference engine code. At this time, ML-MIAOW has in its local memory the model of the target program. Based on this model, the inference engine code judges the existence of an anomaly based on the received branch sequence. If the model discerns the probability of the given branch sequence to be unlikely, the inference engine recognizes it as an anomaly. In this case, MCM is notified of the anomaly and then the host CPU is informed through an interrupt signal. We depict the overall procedure of anomaly detection supported by RTAD in Fig. 5.

Threat model and assumptions: When RTAD is deployed for anomaly detection, we assume that the OS kernel, which configures the hardware modules, is uncompromised. Therefore, we assume that the adversaries cannot directly tamper with the configuration of RTAD. We also rule out direct physical attacks that compromise the underlying CPU and the RTAD hardware modules. In practice, we also adopt any assumptions made by the anomaly detection ML models that are deployed in RTAD such as assuming that the OS and CPU cooperate to forbid a memory page from being both writable and executable simultaneously by enforcing the W$\oplus$X security protection rule where under such assumption, adversaries cannot directly run their code by modifying the code region of the target program.

IV. Experimental Results

To evaluate our approach, we have implemented an RTAD prototype on the Xilinx ZC706 evaluation board. This development board contains the Zynq XC7Z045 FFG900 -2 platform which is equipped with a dual-core ARM Cortex-A9 processor, ARM NIC-301 AXI interconnect, an FPGA chip, 1GB DDR3 SDRAM and other peripherals. We have built the host system with the A9 processor and deployed Xilinx ARM Linux kernel 4.9 as the host OS. Also, the two CoreSight modules, PTM and TPIU, in the Cortex-A9 processor are enabled to extract branch traces from the CPU. The RTAD modules are developed in Verilog HDL to be mapped on the FPGA. Mainly due to the speed limit of FPGA, RTAD modules are configured to operate at 125 MHz except for ML-MIAOW which can satisfy timing constraints only when the clock frequency set to 50 MHz. The CPU clock is lowered to 250 MHz to emulate the performance ratio between the host and the coprocessors in most AP systems [18].

A. Synthesis Results

Based on the aforementioned parameters, we synthesized our prototype onto the FPGA chip and quantified the logics

TABLE I
SYNTHESIZED RESULTS OF RTAD

RTAD Module	Submodule	FPGA			Design Compiler
		LUTs	FFs	BRAMs	Gate Counts
IGM	Trace Analyzer	11962	350	0	12375
	P2S	686	1074	0	14363
	Input Vector Generator	890	1067	0	10430
MCM	Internal FIFO	13	33	10	262
	ML-MIAOW Driver	489	265	0	5971
	Control FSM	1609	1698	0	16977
	Interrupt Manager	42	91	0	927
	ML-MIAOW (5 CUs)	183715	76375	140	1865989
Total		**199406**	**80953**	**150**	**1927294**

Gate counts are given as gate equivalents (1GE = area of 2-input NAND gate).

TABLE II
TRIMMING RESULT OF ML-MIAOW

	LUTs	FFs	Sum	Area
MIAOW [11]	180902	107001	287903	-
MIAOW2.0 [15]	97222	70499	167721	-42%
ML-MIAOW (ours)	36743	15275	52018	-82%

necessary for the RTAD architecture in terms of lookup tables for logic (LUTs), flip-flops (FFs) and block RAMs (BRAMs). The synthesis results are shown in Table I. Our MLPU occupies 91.2% (199,406/218,600) of total LUTs, 18.5% (80,953/437,200) of total FFs and 27.5% (150/545) of total BRAMs. ML-MIAOW executing the inference occupies the majority portion of the total used resources. Through the trimming method from Section III, we were able to deploy five trimmed CUs of ML-MIAOW, while only a single CU of the original MIAOW could be fitted into the same FPGA. To complement the result, we also estimated the gate count of MLPU using Synopsys Design Compiler. With a commercial 45nm process library, the total gate-count of the proposed modules is 1,927,294.

Table II shows the comparison of trimming results of MIAOW between MIAOW2.0 [15] and ours. Since MIAOW2.0 can only support one model at a time while ML-MIAOW can support various ML models, to make a fair comparison, we deploy one LSTM model which will be explained in the following subsection. The result shows that 82% of MIAOW is trimmed in ML-MIAOW while only 42% in MIAOW2.0. This difference is because that the trimming-tool of MIAOW2.0 analyzes the instructions of the target application and only trims unused codes in certain sub-blocks such as ALU or instruction decoder, while we try to find every unnecessary code line across all sub-blocks. As a result, ML-MIAOW has 3.2x more performance-per-area over MIAOW2.0.

B. Performance Analysis

To evaluate the performance overhead RTAD modules incur on the host CPU, we ran SPEC CINT2006 benchmarks with the reference test input workloads. The results are drawn in Fig. 6 where *Baseline* represents the base execution time of benchmarks, and *RTAD* presents the execution time of RTAD. Additionally, three software-based implementations—*SW_SYS*, *SW_FUNC* and *SW_ALL* are compared together. The *strace* utility is used for gathering the system call traces in SW_SYS. For SW_FUNC and SW_ALL, to collect function calls and general branches respectively, we inserted additional instructions to the binary for dumping branch information. RTAD introduces an overhead of 0.052% (geometric mean) while the software-based mechanisms show an overhead of 0.6%, 10.7% and 43.4% in order. Since MLPU has no feedback signal to the CPU that interferes with the processor critical paths, MLPU has no effect on the CPU performance. Note that the performance overhead is mainly attributed to the enabled ARM PTM interface but negligible.

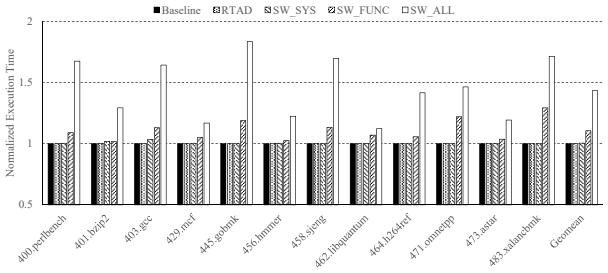

Fig. 6. Performance overhead of RTAD

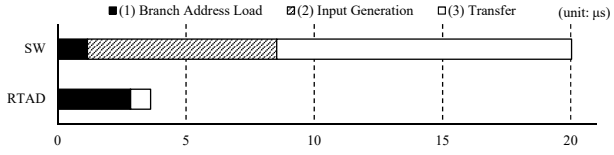

Fig. 7. Data transfer latency of RTAD

For successful real-time inference, what matters is not merely the speed at which MCM processes data, but how quickly the data can be transferred to MCM. When designed in pure software, the host would (1) read the gathered branch address by the instrumented code, (2) refine it into the input vector form and (3) transfer the data to the peripheral memory of MCM. Only upon completion of the latter operation, data will be available to MCM for processing. In our RTAD, (1) IGM decodes the branch address from PTM trace, (2) generates the input vector and then (3) transfer the data by directly driving the input signal of ML-MIAOW. We measured each latency between the software implementation (denoted by SW) and RTAD as shown in Fig. 7.

In SW, it shows an average latency of 20.0μs. Step (3) takes up the biggest part due to high overhead for copying the input vector into the ML-MIAOW memory, averaging at 11.5μs. Step (2) includes multiple data read/write transfers to calculate the input vector and takes 7.38μs. RTAD is measured to average at 3.62μs. Step (1) occupies the largest part. This is mainly because PTM does not send the packets until enough packets are buffered in the FIFO inside the ARM CPU. Step (2) benefits from IGM and requires only 2 cycles (16ns). The remainder is occupied by 0.78μs which is the successive write operations to the ML-MIAOW memory. As can be seen from the results, our work can drive MCM 16.4μs (4,100 cycles in processor frequency) earlier than SW, which would result in faster detection of anomalies.

C. Detection Speed of ML Models

We also evaluated the detection speed of RTAD, which is measured by the total time taken for our inference engine running on MCM to make a judgment on the normality of the behavior of a program immediately after the program executes a branch instruction. To test anomaly detection on our RTAD SoC, we chose two ML models from previous work [2], [8] which show competitive detection accuracy and implementation complexity.

- **Extreme Learning Machine**: The ELM model is more lightweight than a traditional *multi-layer perceptron* (MLP) while providing similar accuracy to MLP. The model in [2] was built upon branch data of system calls.
- **Long Short-Term Memory**: The LSTM model has achieved state-of-the-art results on modeling sequences of data in various fields of study. Researchers in [8] have used general types of branch to build the model.

To reflect real branch patterns, we used the SPEC CINT2006 benchmark suite for each ML model to learn. Moreover,

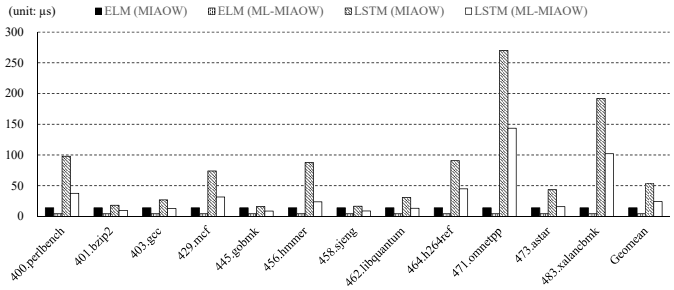

Fig. 8. Latencies of anomaly detection

we emulate attacks by randomly inserting legitimate branch data (i.e., branch addresses that can be observed during normal execution) in normal branch traces because inserting any random branch address would be trivial for detection. This resembles myriads of recent attacks that manipulate the program execution flow by exploiting software vulnerabilities. Fig. 8 shows the latencies taken for each model to detect anomaly after the target benchmarks behave aberrantly.

When the original MIAOW is employed as an inference engine, the ELM and LSTM models have latencies of 13.83μs and 53.16μs on average, respectively. After upgrading MIAOW to ML-MIAOW, the inference is accelerated so that the latencies reduce to 4.21μs and 23.98μs respectively and thus gaining 2.75x improvement on average. Note that the detection latencies of ELM are almost constant regardless of benchmarks while those of LSTM vary significantly. This is because the interval between occurrences of system calls is long enough to process one system call for anomaly inference before the next call comes. But in the case of LSTM where data inputs are branches which occur much more frequently than system calls, the latencies differ from each other because each benchmark has its own unique branch execution pattern. For example, when two branch instructions are executed consecutively in a short period of time, the following instruction would be buffered into the FIFO of MCM until it can be processed. Clearly, this buffering would increase the total detection latency, and in the worst case could cause a loss in branch information as the buffer would overflow and lose newly sent data when the processing speed of the model cannot match the delivery speed of runtime branch data for an extended time period. When MIAOW is employed as an inference engine, this overflow was occasionally observed in a benchmark of heavy branch pressure such as `471.omnetpp`. Fortunately, by upgrading to ML-MIAOW, buffer overflows rarely occur as the speed of processing branch data is fast enough to catch up with the rate of the generation of new branches.

V. Conclusion

RTAD is an ARM-based MPSoC built to infer attack-induced branch behavior anomalies in a real-time manner. It has two heterogeneous processing elements, the ARM CPU and MLPU. According to our evaluation, RTAD imposes virtually no performance burden on the CPU for runtime detection of anomalies. We ascribe this result to the ARM CoreSight architecture, not to mention the combined effort of the two core modules of our MLPU (i.e., IGM and MCM). CoreSight PTM enables MLPU to receive a continuous stream of branch traces from the ARM CPU and by employing our GPU-inspired ML-MIAOW, MLPU efficiently processes the branch traces and runs ML models for anomaly detection. We have prototyped RTAD on an ARM-based FPGA board and demonstrated the effectiveness of our approach.

Acknowledgment

This work was partly supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00230, Development on Autonomous Trust Enhancement Technology of IoT Device and Study on Adaptive IoT Security Open Architecture based on Global Standardization [TrusThingz Project]), (No.2016-0-00078, Cloud based Security Intelligence Technology Development for the Customized Security Service Provisioning) and the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (NRF-2017R1A2A1A17069478) and NRF grant funded by the Korea government (MSIT) [2018R1A2B3001628].

References

[1] S. Forrest, S. Hofmeyr, and A. Somayaji. The evolution of system-call monitoring. In *2008 Annual Computer Security Applications Conference (ACSAC)*, pages 418–430, Dec 2008.

[2] G. Creech and J. Hu. A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns. *IEEE Transactions on Computers*, 63(4):807–819, April 2014.

[3] K. Xu, D. D. Yao, B. G. Ryder, and K. Tian. Probabilistic program modeling for high-precision anomaly classification. In *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th*, pages 497–511. IEEE, 2015.

[4] X. Shu, D. Yao, and N. Ramakrishnan. Unearthing stealthy program attacks buried in extremely long execution paths. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 401–413, New York, NY, USA, 2015. ACM.

[5] S. Das, Y. Liu, W. Zhang, and M. Chandramohan. Semantics-based online malware detection: Towards efficient real-time protection against malware. *IEEE Transactions on Information Forensics and Security*, 11(2):289–302, Feb 2016.

[6] K. Xu, K. Tian, D. Yao, and B. G. Ryder. A sharper sense of self: Probabilistic reasoning of program behaviors for anomaly detection with context sensitivity. In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*, pages 467–478. IEEE, 2016.

[7] L. Chen, S. Sultana, and R. Sahita. Henet: A deep learning approach on intel® processor trace for effective exploit detection. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 109–115. IEEE, 2018.

[8] H. Yi, G. Kim, J. Lee, S. Ahn, Y. Lee, S. Yoon, and Y. Paek. Extended abstract: Mimicry resilient program behavior modeling with lstm based branch models. *1st Deep Learning and Security Workshop (DLS 2018), arXiv preprint arXiv:1803.09171*, 2018.

[9] Z. Chiba, N. Abghour, K. Moussaid, A. El Omri, and M. Rida. A survey of intrusion detection systems for cloud computing environment. In *2016 International Conference on Engineering MIS (ICEMIS)*, pages 1–13, Sept 2016.

[10] P. S. Kenkre, A. Pai, and L. Colaco. Real time intrusion detection and prevention system. In *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014*, pages 405–411, Cham, 2015. Springer International Publishing.

[11] R. Balasubramanian, V. Gangadhar, Z. Guo, C. Ho, C. Joseph, J. Menon, and et al. Enabling gpgpu low-level hardware explorations with miaow: An open-source rtl implementation of a gpgpu. *ACM Trans. Archit. Code Optim.*, 12(2):21:21:1–21:21:25, June 2015.

[12] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev. Malware-aware processors: A framework for efficient online malware detection. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 651–661, Feb 2015.

[13] 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 '13, pages 559–570, New York, NY, USA, 2013. ACM.

[14] M. Rahmatian, H. Kooti, I. G. Harris, and E. Bozorgzadeh. Hardware-assisted detection of malicious software in embedded systems. *IEEE Embedded Systems Letters*, 4(4):94–97, Dec 2012.

[15] P. Duarte, P. Tomas, and G. Falcao. Scratch: An end-to-end application-aware soft-gpgpu architecture and trimming tool. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 165–177, New York, NY, USA, 2017. ACM.

[16] A. V. Fidalgo, M. G. Gericota, G. R. Alves, and J. M. Ferreira. Real-time fault injection using enhanced on-chip debug infrastructures. *Microprocess. Microsyst.*, 35(4):441–452, June 2011.

[17] Y. Lee, J. Lee, I. Heo, D. Hwang, and Y. Paek. Using coresight ptm to integrate cra monitoring ips in an arm-based soc. *ACM Trans. Des. Autom. Electron. Syst.*, 22(3):52:1–52:25, April 2017.

[18] L. Codrescu. Architecture of the hexagon 680 dsp for mobile imaging and computer vision. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–26, Aug 2015.