Detecting denial-of-service hardware Trojans in DRAM-based memory systems

Detecting denial-of-service hardware Trojans in DRAM-based memory systems

Heba Salem
School of Informatics
The University of Edinburgh
Edinburgh, United Kingdom
s1573838@ed.ac.uk

Nigel Topham
School of Informatics
The University of Edinburgh
Edinburgh, United Kingdom
nigel.topham@ed.ac.uk

***Abstract*—DRAM latencies are inherently variable, potentially allowing a denial-of-service hardware Trojan (DoS HT) to degrade memory performance without becoming immediately obvious. This paper addresses the challenge of detecting a DoS HT that may have been inserted into a DRAM-based memory system, without requiring detailed internal knowledge of the DRAM device. We present a real-time machine-learning based DoS HT detection technique based on a low-cost hardware monitor at the interface to the DRAM controller, coupled with periodic software based analysis of monitoring output.**

***Index Terms*—DRAM, Hardware security, Hardware Trojans, Denial of service**

I. Introduction

The ubiquity of DRAM devices, and the influence their performance has in determining overall system performance, makes them a potential target for denial of service hardware Trojan (DoS HT) attacks. A DoS attack may typically extend the latency of a DRAM access to reduce system performance. The latency of an uninfected DRAM-based memory system will itself exhibit significant variability from one access to another. From the security perspective, this variability could be seen as an inherent vulnerability in DRAMs as the impact of a latency-targeting DoS HTs may be subtle and difficult to identify, allowing the HT to hide in plain sight.

To our knowledge, this is the first work that explores the challenges of detecting DRAM-based DoS HTs, and the first to propose a machine-learning based approach for detecting such HTs. Additionally, our DoS HT detection approach is based on DRAM characteristics that are observable at the interface to the DRAM controller, eliminating the requirement for knowledge about the specific physical and internal characteristics of a given DRAM; an issue associated with many of the published DRAM latency characterization techniques that mostly target DRAM performance optimization. Moreover, DRAM characteristics are increasingly dynamic, requiring a dynamic approach to DRAM latency analysis. In this paper, a hypothesis regarding the vulnerability of DRAMs To DoS HTs is introduced and tested using experimental simulations, a run-time machine-learning based DRAM latency analysis technique is then proposed and its feasibility and effectiveness in detecting such HTs is demonstrated.

Section II motivates the proposed technique, and in section III its foundations are explained. The experimental setup is presented in section IV. The implementation of the technique in real-life systems and in detecting DoS HTs is presented in sections Vand VI.

II. Motivation and related work

DRAM-based memory systems have highly variable latencies, resulting from variability in the number of steps needed to perform each read or write access and by the use of multi-rate refresh adopted in most DRAMs to account for the variable retention time (VRT) phenomenon.

The unpredictability of DRAM latency not only affects performance modeling, but also presents a vulnerability that could be exploited by security attackers to mask their attacks on DRAMs. A hardware Trojan generally comprises a set of malicious modifications to the implementation of a sub-system with the intention of leaking secret information, degrading the performance of the system, or causing a denial of service [1]. Several publications have explored the possibility of injecting HTs into memory systems; in [2] a HT that maliciously redirect a page addresses in DRAMs and could also bypass memory protection was introduced. HTs that could be inserted into the GDS-II files of an SoC by a malicious foundry in order to leak data or cause faults in DRAMs were proposed in [3]. HTs that can perform post-deployment data tampering in SRAMs and that can leak sensitive data out of NVM cells were introduced in [4] and [5], respectively. However, denial of service HTs have not been explored extensively in the context of memories and DRAMs.

Such HTs could degrade system performance by extending the latency of some or all types of DRAM access. However, their effects would be masked and made apparently invisible if the resulting extended latencies nevertheless remained within the limits of the worst case latency of the DRAM system. Consequently, any effort to counter and mitigate such HTs would need to be based on a comprehensive analysis of the attributes of unpredictability in DRAM latency. Several publications have attempted to characterize and model the highly variable latency of DRAMs, albeit for the primary purpose of reducing latency [6] [7] or for detecting DRAM errors and failures [8] [9] [10] [11]. None, however, focused on characterizing DRAM latency for the purpose of detecting DoS HTs or developed a universal and device-agnostic DRAM latency characterization and modelling technique that could be used with all types of DRAMS. Such techniques must be capable of inferring the presence of DoS HTs by analysing their impact on the latency of DRAM systems while accounting for the unpredictability of these systems. The focus of this paper is in developing a technique that recognizes the effect of DRAM unpredictability on its latency and is able to detect the presence of a DoS HT through even quite modest impacts on latency. In addition to the run-time detection of DoS HTs, the work presented in this paper could also pave the way to detecting other types of attacks on DRAMs that have a direct or indirect effect on its latency, such as environmentally-induced fault injection attacks.

III. Proposal background

In addition to the VRT phenomenon that led to the development of multi-rate refresh, which in turn added additional variability to DRAM latency, DRAMs have operational variability that is defined by their state and access patterns. The three stages of accessing the prevalent DDRx DRAMs involve activating the requested memory row, precharging and closing the previously activated row, and finally performing the requested read/write transaction. These three stages, however, are not initiated for all transactions; a transaction that is destined to the same row (in the same bank) as the previous transaction doesn't need to spend time activating or precharging the required row (open-row access), whereas a transaction that targets a row that is neither precharged nor activated needs all three stages (conflict-row access). Additionally, the type of the current transaction and that of the preceding transaction (i.e. read or write) determine whether the bidirectional DDRx DRAM bus need to be switched or not, which induces an additional delay of up to 18 cycles [12].

To test the hypothesis that variability in DRAM latency could mask the presence of a DoS HT we simulated a DRAM-based system under a number of DoS HT scenarios, and examined the latencies obtained from each of these simulations along with the latency of a baseline HT-free simulation. We assumed an injected DoS HT may introduce delays at the point where transactions are forwarded from the arbiter to the scheduler (inside the DRAM controller) and/or at the point where they are released from the DRAM controller. Table I summarizes the actions performed by the simulated HTs. From the values of the obtained latencies shown in Figure 1, it is clear that most transactions have a latency of less than 300ns, regardless of whether a HT was inserted or not. Furthermore, the maximum baseline latency of 464.5 ns was exceeded only twice by each of HT B and HT C, whereas the HTs A, D, and E were found to never exceed that value in around 2 million transactions. This finding strongly supports our hypothesis that the actions of DoS HTs could be masked and obfuscated by the variability and unpredictability of DRAMs and that the upper bound baseline latency might never be exceeded when ”stealthy” HTs are injected into a DRAM system.

Our proposed technique is based on the notion that the latency of a specific DRAM access can be modeled as a discrete function of a small number of dynamic access features. Specifically, the latency of a DRAM device at a given point of time is directly influenced by its workload, the type of the transaction to be performed, and the attributes of preceding transactions.

TABLE I: The implemented ”DoS HT” modifications

HT	**Forward**	**Release**	**Read**	**Write**	**Delay**
HT A	✓	✓		✓	11.25 ns
HT B	✓	✓	✓		11.25 ns
HT C	✓	✓	✓	✓	11.25 ns
HT D	✓		✓	✓	5 ns
HT E	✓		✓	✓	1.25 ns

This approach allows a baseline distribution of DRAM latencies to be constructed as a machine-learning model for each combination of dynamic access features. These models can then be used to predict the expected DRAM latencies that should be experienced at run-time, allowing anomalous latency distributions to be detected.

A key aspect of this hypothesis is the influence of the workload on the latency. Internally, DRAMs contain per-bank request queues, and the dynamic depth of each queue naturally influences the latency of requests entering the queue. This relationship was verified by putting a simulated DRAM system under varying amounts of load, derived across a number of different benchmark applications. The results are presented in the heat map of Figure 2. The DRAM load, represented as the number of outstanding operations, is plotted against the latency experienced by the respective transaction. This shows the general trend that increasing load results in increased latency. However, it is also clear that load is not the only determinant of latency, as there is a noticeable overlap between the ranges of latencies for the different loading conditions – other latency-affecting parameters also need to be taken into account.

Given that the DRAM latency varies as a consequence of the variability in the number of read/write steps, we identified a set of external parameters that influence the latency of a given access for a specific DRAM state at run-time. These parameters, which we refer to as features of the DRAM latency model, are unified across all DRAM systems and hence are not specific to DRAM architecture or implementation. These features are:

- Dynamic DRAM load, defined as the number of outstanding transactions
- Type of transaction (read or write)
- Type of preceding transaction (read or write)
- The row (and the bank) to which the transaction is destined
- The row (and the bank) to which the preceding transaction was destined

Collecting data related to these features at run-time allowed us to identify the loading condition and the state of operation of the DRAM (i.e. open-row, closed-row, or conflict-row). Additionally, it allowed us to determine whether the direction of the internal DRAM bus would be switched by the

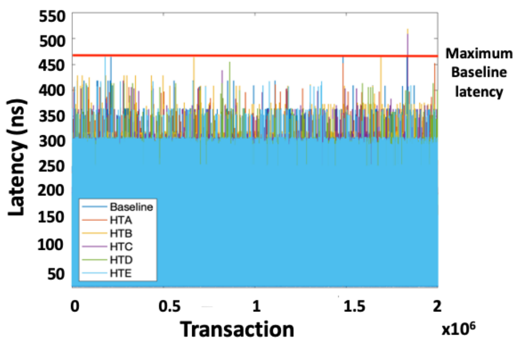

Fig. 1: Memory access latencies for HT-free and DoS HT DRAM simulations

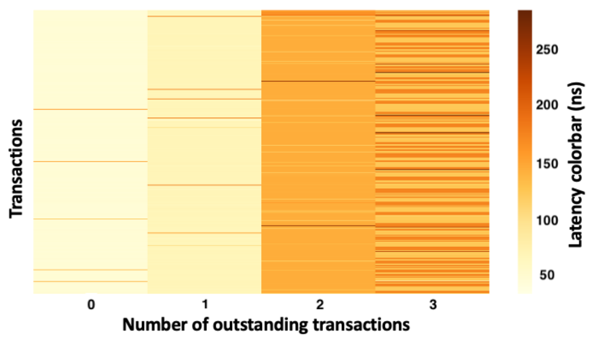

Fig. 2: Heat map showing the range and distribution of latencies for different DRAM loadings

transaction. All of these features directly affect the latency of the next access. In this context, it is worth noting that in this work the focus was on the DRAM access patterns that result in best case latency (open-row access) and worst-case latency (conflict-row access) and hence the third type of access, closed-row access, was not exclusively addressed. Nevertheless, the framework could with slight adjustment be extended to address and analyze closed-row access by simply adding the bank number to the list of features.

We use the following terminology to describe concisely the feature set for any transaction (or set of transactions) received by the DRAM system: T_cT_pRL. Where T_c is the type of the current transaction, T_p the type of the previous transaction, R is the row situation, and L for the number of outstanding transactions at the DRAM interface. Therefore, a read transaction that came after another read transaction when the load was three, and where both transactions targeted the same row, is denoted RRS3. Whereas, a write transaction after a read transaction to different rows and with zero load is denoted WRD0.

IV. Methodology and Experiments

A. Experimental setup

The experimental setup for reporting the results in this paper was based on the integration of two simulators; the gem5 CPU simulator [13] and the DRAMSys memory simulator, which simulates a DDR3 DRAM and its controller [14] [15] [16]. We selected our workload as a subset of applications from the PolyBench benchmark suite [17]. Table II summarizes the details of the experimental setup.

TABLE II: The experimental setup

Simulation	Specifications
System simulation	Gem 5 simulator
	System emulation(SE) mode
	O3 CPU (out-of-order CPU)
	a range of frequencies from 1 GHz to 8 GHz[1]
	32KB L1 cache and 8KB L2 cache
DRAM simulation	DRAMSys simulator
	gem5 coupling mode
	DDR3 standard

A small number of minor modifications were made to DRAMSys to enable the extraction of transaction traces containing the start and end times for each transaction (observed at the DRAM controller), as well as the row and bank number and transaction types. With minimal knowledge of the DRAM dimensions a hardware DRAM transaction monitor could infer the row and bank numbers, and a cycle counter could be used to create timestamps for the start and end time of each transaction. Traces of 20 million DRAM transactions were extracted from the benchmarks and used for offline training of our machine-learning model of DRAM latency.

B. DRAM transaction features and categories

In order to assess the impact of each combination of transaction features on the resultant latency, the memory traces obtained from simulation were sorted, placing transactions with identical features in the same group. There are four combinations of transaction type for the current and preceding transaction (RR, RW, WR, and WW), and two combinations of the current and preceding rows (same or different), resulting in eight categories that are uniform across all systems. The last feature, the load, is an orthogonal feature that determines the total number of categories based on the maximum number of outstanding transactions occurring for each of the previous eight categories. Although, our simulated system occasionally had transactions occurring when there were up to six outstanding transactions, this occurred only rarely; hence, and without loss of generality, we limited our analysis to a maximum of three outstanding transactions. This resulted in a total of 32 transaction categories, and naturally some categories had more transactions than others. By analysing the range of latencies for each of the different categories it was found that the latency distribution for each of the categories had its own distinctive distribution with one large peak and one small peak. The large peak occurred at the nominal latency for that category, whereas the small peak occurred at a latency resulting from instances where a transaction of that category coincided with an internal

[1]High-frequency simulations were ran to force the system to generate main memory transactions at a higher rate for the purpose of training the machine-learning model.

refresh cycle. This lengthens the latency as the transaction has to wait for the refresh to be performed before it is processed.

The aggregated latency distribution of transactions from all categories (i.e. unsorted) would contain many individual peaks at different latency values, making for difficult analysis. However, by sorting the transactions into categories, defined in terms of fundamental features of a transaction, we see a collection of distributions each showing the distinctive and recognizable properties shared by transactions of the same category.

C. The machine-learning algorithm

Predicting the latency of each individual transaction is not our purpose; instead we are interested in knowing whether the DRAM system exhibits latencies that are nominal or unusual. This requires real-time statistical analysis of measured DRAM latencies and an efficient method by which to compare them against the expected baseline latency. We have seen that DRAM latency is not a fixed predictable value, even for transactions of the same category. However, DRAM latency can be represented efficiently as a discrete probability distribution, and in real time such a distribution can be measured by capturing a histogram of latencies. We therefore selected a machine learning approach through which we could create a model of the actual latency distribution for each category of access. For this we use Kernel Density Estimation (KDE), a non-parametric probability density estimation that has the ability to provide probability distribution approximations for data-sets with multiple peaks and non-defined distributions – as we see in the probability distributions for DRAM latencies. We used the KDE implementation provided by the Scikit-learn tool [18]. A top-hat kernel was used and the KDE algorithm bandwidth was set to 0.5 so that a reasonable estimation can be obtained that is neither too detailed nor too simplified.

The latency data obtained from running the Polybench benchmark applications on the simulated system was divided into a training and evaluation data-sets. The number of DRAM transactions in each of the training and evaluation sets for each benchmark are shown in Table III

TABLE III: Number of transactions in the testing and evaluation data-sets

Benchmark	Training	Evaluation
atax	2,420,781	1,000,000
jacobi-2d	7,030,047	2,000,000
adi	5,129,039	2,571,703
mvt	1,101,975	1,498,752
3mm	1,259,717	289,503
bicg	83,773	216,635
doitgen	1,286,567	879,942
2mm	1,002,676	39,197

Transactions from the training data-set were partitioned into the 32 categories and each partition was used to train a distinct KDE model that would serve as a baseline latency model for that category. The evaluation data-sets were then used to construct evaluation KDE models for the purpose of comparing them with the baseline distributions obtained from the training data-sets. This comparison was first performed through visual inspection of the training and evaluation KDEs, which confirmed that the training KDEs could be used as reference for analysing and detecting anomalies in latency data obtained in real-time. For example, the KDEs of the training-data sets and those of the evaluation data-sets for the RR category are shown in Figures 3 and 4. From these figures, it can be seen that the KDEs of the evaluation and training data for the same categories are similar, and providing initial confidence that DRAM latency modeling could be made predictable, provided transactions are characterized appropriately.

Visual inspection is a subjective approach to comparing and analysing the obtained KDEs and that our main goal is building an automated and systematic DoS HT detection technique, the possibility of using statistical methods of similarity or distance measurement for the quantification of the similarity between KDEs was explored. One such technique is the Jensen–Shannon distance. This is the square-root of the Jensen–Shannon divergence, which is in turn based on the Kullback-Leibler divergence that measures the divergence of one probability distribution from another using the sum of probabilities in the former and the log of probabilities of both.

The Jensen–Shannon distance (J-S distance) was utilized for quantitatively measuring the similarity between the KDEs obtained from training and evaluation data-sets, or more precisely, the similarity between the KDEs of the baseline distribution and the real-time distribution, when deploying the framework in a live system. Figure 5 shows the values obtained for the J-S distance between the KDE of each training set and the KDE of the corresponding evaluation set for the 20 most populated categories. We observe that all J-S distances are in the range 0–0.4, with the majority being below 0.3. Since the basic J-S distance is a distance that could only take values between 0 and 1, where 0 is for identical data-sets and 1 is for completely distinct ones, the values shown in Figure 5 clearly indicate that the training and evaluation data for all categories have very similar attributes. These distances further demonstrate the feasibility of using the proposed framework in long-term analysis and inference of any abnormalities, shifts, or gradual and consistent change in DRAM latency. As the more discrepancies there are between the shapes of the training KDEs and the real-time KDEs, the larger the J-S distances between them are and the further these distances are from those obtained from the HT-free case.

V. The implementation

The implementation of our DRAM latency analysis framework is presented in outline in Figure 6. The first two steps are the recording and sorting of data observed at the DRAM controller interface. For reasons of practical efficiency and security the first two steps would be performed using hardware, either in a security wrapper [19] [20] or in the DRAM controller itself. For recording the data relevant to each transaction, a log of size $N + 1$ would be needed, where N is the maximum number of outstanding transactions (at the interface of the DRAM controller) allowed by the system

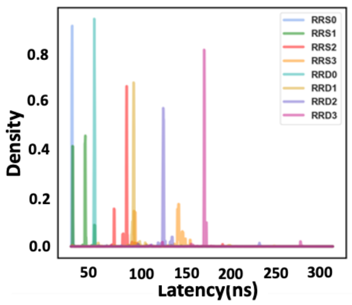

Fig. 3: Training KDEs of RR transactions

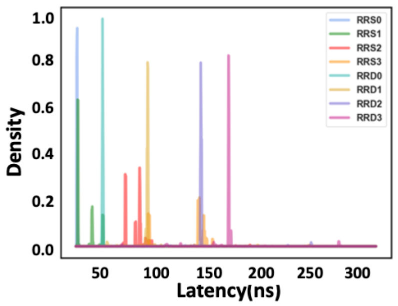

Fig. 4: Testing (evaluation) KDEs of RR transactions

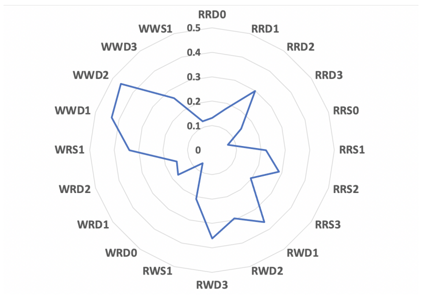

Fig. 5: The J-S distances between the training KDEs and the testing KDEs

at any given point of time. This logging mechanism shall allow the categorizing of transactions based on the values of the DRAM latency features as, for instance, RRS0, RRS1, RRD0, RRD1, RWS0, RWS1, and so on. Each category has a dedicated collection of bins, each entry of which is a simple reference counter. The number of bins depends on the range of latency expected for that category and its granularity. The convention here is to avoid unnecessary hardware overhead by quantizing the latency values and in a number of sub-ranges that are within the overall expected latency range for each category. For example, a category with an expected latency range between 20ns and 40ns could have 10 bins where the first bin is used to count the number of occurrences of latency values between 20ns and 22 ns. Deciding on the number of bins needed for each category is a design choice that the SoC integrator/ IC designer would choose based on how accurate and detailed they want their analysis to be. The KDE modelling and testing carried out in this work was performed after quantizing each of the latency ranges, for each of the 32 different categories, into 1000 bins. This represented a good trade-off between histogram size and specificity for the latency ranges obtained from our simulations, which for most categories were relatively large. Additionally, we maintained extra bins for each category to count the occurrences of latency values that were beyond the expected latency bounds for the category (to account for any out-of-bounds DoS HT action).

The monitoring hardware effectively histograms the actual latencies observed for each of the 32 categories of DRAM access. These real-time histograms are processed periodically by software embedded within the operating system. The interval between software sampling of the hardware counters depends on the dynamic range of the counters, the expected frequency of DRAM accesses, and the time-wise resolution and responsiveness required of the overall DoS HT monitoring system. Each time the software-based module is run it creates a KDE model from the run-time latency data for each transaction category over the previous time interval. The J-S distance between the instantaneous KDE and the baseline KDE is then computed and used to assess the similarity between the current DRAM latency distribution and the baseline distribution obtained from pre-deployment data (or from previous trusted run-time data). Any significant discrepancy between the two KDEs may indicate a potential DoS HT attack.

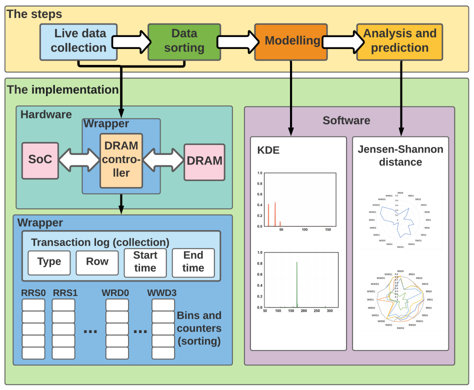

Fig. 6: The implementation of the proposed technique in real-life systems

VI. Detection of DoS HTs

For an analysis of the DRAM latency behaviour when the DRAM is infected with a DoS HT, the data obtained from simulating the HTs in table I was used in training KDE models and comparing those KDEs with the baseline ones (obtained from HT-free simulations) using the Jensen-Shannon distance. Before discussing the results of this comparison, it is fundamental to explain how those HTs were introduced to our experimental setup. In particular, an integer multiplier was used to vary the value of the "notification delay" variable that is used in DRAMSys when calculating the time needed to forward transactions from the arbiter to scheduler and the time needed to release transactions from the DRAM controller. Doing so added delays to DRAM transactions, extending their latencies. These additional delays served as a proxy for expected DoS HT behavior. Figure 7 shows the J-S distance between the HT-KDEs and baseline-training KDEs for the 20 most populated categories. The J-S distances between the evaluation data with inserted HS proxy behavior and the baseline allow each type of HT to be identified by its selective impact on each category of transaction. For example, the J-S distances obtained from HT C are the ones that most divert from the baseline J-S distances, conforming with the fact that HT C is the one with the greatest latency-extending action that affects all types of transactions. Whereas, the J-S distances obtained from HT B, which targets only read transactions, don't show large discrepancies from the baseline case for categories concerned only with write transactions.

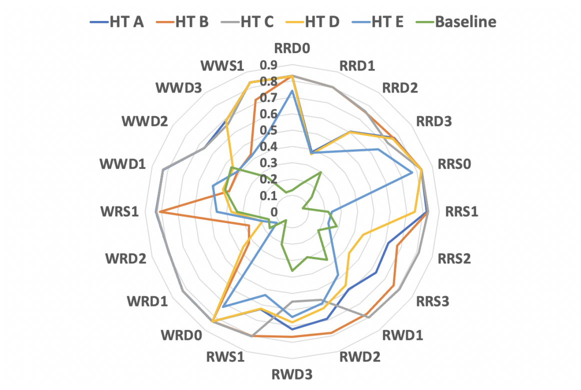

Fig. 7: The J-S distance between the training KDEs and the testing KDEs, in normal conditions and HT conditions

VII. Conclusion

The inherent variability in DRAM operation has been identified as a hindrance to proper characterization and analysis of DRAM latency. Several publications introduced techniques to address this issue, with the main aim of introducing more efficient refresh mechanisms or detecting DRAM failures. In our work, we focus on an unexplored aspect of the variability in DRAM latency and introduce a hypothesis on the possibility of using this variability in obfuscating attacks on DRAM, and specifically, denial-of-service hardware Trojan attacks. Through experimental verification we test this hypothesis and show that it is possible to hide the actions of DoS HTs in the unpredictability of DRAM latency, and consequently, we propose a Kernel Density Estimation based run-time technique to characterize the distribution of latencies partitioned into categories according to DRAM access patterns and loading conditions. This facilitates the potential detection of DoS HTs through continuous statistical comparison between a set of real-time distributions of DRAM latency and a set of baseline distributions. Basing our technique on externally observable features, such as the type of DRAM transaction and the workload, instead of internal factors such as the refresh mechanism and DRAM cell retention rate, allows it to be applied to all types of DRAMs including untrusted third party DRAMs. Our paper is to our knowledge, the first to investigate the possibility of using the variable DRAM latency in obfuscating DoS HT attacks, and to develop a generic framework to characterize this variable latency using machine-learning techniques.

References

[1] R. S. Chakraborty, S. Narasimhan and S. Bhunia, "Hardware Trojan: Threats and emerging solutions," 2009 IEEE International High Level Design Validation and Test Workshop, 2009, pp. 166-171.

[2] B. Hopkins, J. Shield and C. North, "Redirecting DRAM memory pages: Examining the threat of system memory Hardware Trojans," 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2016, pp. 197-202.

[3] K.Nagarajan, A.De,M.N.I. Khan, and S. Ghosh, "Trapped:DRAM trojan designs for information leakage and fault injection attacks,"ArXiv,vol. abs/2001.00856, 2020.

[4] T. Hoque, X. Wang, A. Basak, R. Karam and S. Bhunia, "Hardware Trojan attacks in embedded memory," 2018 IEEE 36th VLSI Test Symposium (VTS), 2018, pp.1-6.

[5] M.N. Imtiaz Khan, K. Nagarajan and S. Ghosh, "Hardware Trojans in Emerging Non-Volatile Memories," 2019 Design, Automation & Test in Europe Conference Exhibition (DATE), 2019, pp. 396-401.

[6] K.K. Chang, A. Kashyap,H. Hassan, S. Ghose, K. Hsieh, D. Lee,T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding latency variation in modern DRAM chips: Experimental characterization, analysis,and optimization," in Proc. 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science (SIGMETRICS '16), New York, NY, USA: ACM, 2016, pp. 323–336.

[7] A. Das, H. Hassan and O. Mutlu, "VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency," 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), 2018, pp. 1-6.

[8] I. Giurgiu, J. Szabo, D. Wiesmann, and J. Bird, "Predicting DRAMreliability in the field with machine learning," in Proc. 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track, (Middleware '17). New York, NY, USA, ACM, 2017, p.15–21.

[9] E. Baseman et al., "Improving DRAM Fault Characterization through Machine Learning," 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), 2016, pp. 250-253.

[10] E. Baseman et al., "Physics-Informed Machine Learning for DRAM Error Modeling," 2018 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2018, pp. 1-6.

[11] L. Mukhanov, K. Tovletoglou, H. Vandierendonck, D. S. Nikolopoulos and G. Karakonstantis, "Workload-Aware DRAM Error Prediction using Machine Learning," 2019 IEEE International Symposium on Workload Characterization (IISWC), 2019, pp. 106-118.

[12] M. Hassan, "On the Off-Chip Memory Latency of Real-Time Systems: Is DDR DRAM Really the Best Option?," 2018 IEEE Real-Time Systems Symposium (RTSS), 2018, pp. 495-505.

[13] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu,J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell,M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator, "SIGARCH Comput. Archit. News, vol. 39, no. 2, Aug 2011, pp. 1–7.

[14] M. Jung, C. Weis, N. Wehn, and K. Chandrasekar, "TLM modelling of 3D stacked wide I/O DRAM subsystems: A virtual platform for memory controller design space exploration," in Proc. 2013 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '13), New York, NY, USA, ACM, 2013.

[15] M. Jung, C. Weis, and N. Wehn, "DRAMsys: A flexible DRAM subsystem design space exploration framework," IPSJ Transactions on System LSI Design Methodology, vol. 8, pp. 63–74, 2015.

[16] L. Steiner, M. Jung, F. S. Prado, K. Bykov, and N. Wehn, "DRAMsys 4.0: A fast and cycle-accurate Systemc/TLM-based DRAM simulator," Embedded Computer Systems: Architectures, Modeling, and Simulation, A. Orailoglu, M. Jung, and M. Reichenbach, eds., Springer International Publishing, 2020, pp. 110–126.

[17] L.-N. Pouchetet al., "Polybench: The polyhedral benchmark suite, "URL: http://www.cs.ucla.edu/pouchet/software/polybench, vol. 437, 2012.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O.Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, 2011, pp. 2825–2830.

[19] A. Basak, S. Bhunia, and S. Ray, "A flexible architecture for systematic implementation of SoC security policies," in 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2015, pp. 536–543.

[20] H. Salem and N. Topham, "Trustworthy computing on untrustworthy and Trojan-infected on-chip interconnects," 2021 IEEE European Test Symposium (ETS), 2021, pp. 1-2.