The First Concept and Real-world Deployment of a GPU-based Thermal Covert Channel: Attack and Countermeasures

Jeferson González-Gómez*†, Kevin Cordero-Zuñiga*, Lars Bauer†, Jörg Henkel†
* *Instituto Tecnológico de Costa Rica (TEC)*
† *Karlsruhe Institute of Technology (KIT), Chair for Embedded Systems (CES)*
{jeferson.gonzalez,lars.bauer,henkel}@kit.edu, kevinscorzu@estudiantec.cr

***Abstract*—Thermal covert channel (TCC) attacks have been studied as a threat to CPU-based systems over recent years. In this paper, we propose a new type of TCC attack that for the first time leverages the Graphics Processing Unit (GPU) of a system to create a stealthy communication channel between two malicious applications. We evaluate our new attack on two different real-world platforms: a GPU-dedicated general computing platform and a GPU-integrated embedded platform. Our results are the first to show that a GPU-based thermal covert channel attack is possible. From our experiments, we obtain a transmission rate of up to 8.75 bps with a very low error rate of less than 2 % for a 12-bit packet size, which is comparable to CPU-based TCCs in the state of the art. Moreover, we show how existing state-of-the-art countermeasures for TCCs need to be extended to tackle the new GPU-based attack at the cost of added overhead. To reduce this overhead, we propose our own DVFS-based countermeasure which mitigates the attack, while causing $2\times$ less performance loss than the state-of-the-art countermeasure on a set of compute-intensive GPU benchmark applications.**

***Index Terms*—security, thermal covert channel, GPU, attack, DFT, DVFS, countermeasure**

I. Introduction

In cyber-security, a covert channel attack is a means of communication between two applications or processes which are not allowed to communicate in a specific system [1]. Thermal covert channel (TCC) attacks leverage the temperature of a compromised component to establish a stealthy channel between these malicious applications.

As a security threat, TCCs have been studied in several fields, ranging from multi/many-core [2]–[4] and cloud FPGA environments [5] to embedded systems [6].

In a TCC, an attacker application (typically denoted as *transmitter* or *malware*) has infiltrated a secure zone (e.g., a *Trusted Execution Environment* or TEE). From there, the transmitter encodes sensitive binary data as temperature variations by performing intensive computing on its CPU when transmitting a binary 1 or leaving the CPU in an idle state when transmitting a 0. In a non-secure zone, a second malicious application (typically denoted as *receiver* or *spy*) reads the thermal sensors of its core, where the temperature variation produced by the transmitter is noticeable. By doing so, the receiver can interpret temperature variations as 1s or 0s, hence establishing a communication channel between the attackers.

This work is partially funded by the Deutscher Akademischer Austauschdienst (DAAD) and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project Number 146371743 - TRR 89 Invasive Computing.

In recent years, GPU computing has risen as an efficient way to accelerate computation in several domains such as high performance, cloud computing, and machine learning (ML) applications due to its highly parallel nature. With the surge of GPU-based solutions in these diverse domains, providing security to the GPU has become more and more critical, especially when it is used on sensitive data [7]. Given the recent utilization of GPUs in trusted environments [8]–[11], we propose to explore the possibility of creating a GPU-based TCC attack, which has not been proposed so far and constitutes a new attack vector for which no countermeasures have been analyzed so far.

Although GPU and CPU-based attacks share a similar functioning principle, several challenges in the attack implementation and countermeasure effects make the distinction relevant.

Challenges in the attack implementation: On one hand, GPUs are comprised of hundreds or thousands of small cores, with typically one thermal sensor for the whole device. This means that raising the temperature to a noticeable degree involves a highly parallel computational effort which has to be high enough to heat the device, but also not too long so that the device can cool down in a short time. This fact affects the packet size, as temperature accumulations affect the error rates of the channel as the number of consecutive transmitted bits increases. This is not the case for CPU-based attacks, as each physical core has its thermal sensor that is leveraged individually. As consequence, CPU-based TCCs normally reach transmission rates of up 45 bps with very high frequencies of up to 500 Hz for 64-bit packets on a single core [4], [12]. We discuss more GPU-specific TCC challenges and solutions to overcome them in Section III-B.

Challenges in the countermeasures: On the other hand, Dynamic Voltage and Frequency Scaling (DVFS) techniques [13], which are considered the reference countermeasures to CPU TCCs in the state of the art, work by reducing the frequency of the CPU involved in the attack. However, due to the shared nature of the GPU, reducing the GPU frequency without any further consideration severely affects the execution of other GPU-dependent applications on the system. The same can be said of other GPU-based countermeasures (which we analyze

978-3-9819263-7-8/DATE23/© 2023 EDAA

and expand further in Section IV). Since the GPU is typically shared for the whole system, the performance loss on other applications when applying a countermeasure is much greater for GPU- than for CPU-based TCCs, as our evaluation shows.

In this paper, we investigate the construction of GPU-based TCCs in terms of their effectiveness and response to current countermeasures, as well as the considerations needed to port these countermeasures to the GPU domain. The main contributions of this paper are the following:

- We are the first to propose using the GPU to create a thermal covert channel (TCC) for heterogeneous CPU-GPU computing systems.
- We develop a real-world implementation of the new GPU-based TCC that evades existing detection techniques in the state of the art by exploiting a previously unconsidered computing resource.
- We analyze and expand current CPU-based countermeasures for TCCs to tackle the new GPU-based attack and measure their performance impact on other applications.
- We propose a new thermal-aware DVFS countermeasure to GPU-based TCCs that mitigates the attack while producing less performance loss than the state-of-the-art solution on a set of compute-intensive GPU benchmark applications.

II. Related Work

A. Thermal Covert Channels and Countermeasures

The feasibility of TCCs in multi-core processors was first shown by Masti *et al.* [2]. There, the authors achieved a data transmission rate of 1.33 bps with a bit error rate (BER) of 11 % on a CPU-based system. Since then, different works have increased the channel efficiency with new encoding schemes on the temperature signal, such as the Manchester [12] and Return-to-Zero encoding (RZE) [3]. Modulation [3] has also been introduced in TCCs as a way to improve the overall data rate, by shifting the frequency at which the signal is transmitted and hence avoiding low-frequency noise interference due to regular CPU activity. These techniques have shown stable and improved transmission rates of up to 20 bps with low BER and packet error rate (PER) in multi/many-core processors.

More recently, an effort has been put into combating the TCCs. One of the initial techniques to tackle thermal channels consists in generating thermal noise by doing extra processing. This approach was later improved with channel jamming with frequency-specific noise [14] aimed at the detected TCC frequency. Although the latter approach considerably increased the error rate of the channels, rendering them unusable, unproductive processing was still needed to produce the thermal noise. A more efficient solution was then proposed by leveraging the DVFS mechanism, normally used by thermal- and power management modules in many-core systems [4], [13], while also introducing detection techniques based on temperature and performance frequency spectrum analysis.

GPU-based TCCs have never been proposed before. TCC countermeasures, as they appear in the state of the art, deal with either analyzing the CPU behavior to detect the attack, or interfering with the CPU temperature to block the attack. *None of the current countermeasures has considered the GPU as a possible medium for a TCC*. Consequently, state-of-the-art countermeasures need to be further expanded to overcome the corresponding threat. Moreover, the effect of the extended GPU-based countermeasures on the performance of other applications has not been analyzed before this work. We extend the aforementioned state-of-the-art thermal noise [14] and DVFS [13] countermeasures to tackle GPU-based attacks and analyze their impact on other applications as part of our contribution.

B. Trusted Environment and GPU-based Covert Channels

Providing security and isolation to the GPU when working with sensitive data has become increasingly critical with the rise of big data and ML applications [7]. The problem of supporting a trusted execution environment (TEE) for the GPU was first stated by Volos *et al.* [8]. There, the authors showed the feasibility of GPU TEEs for commercial off-the-shelf NVIDIA GPUs, which supported the secure copying of data and kernel execution on the GPU. More recently, several other approaches have tackled the inclusion of the GPU in TEEs in diverse environments such as heterogeneous systems [11], IoT devices [9], and the cloud [10].

Although TCCs on GPUs have not been studied before, other types of covert channels that leverage the GPU as an attack source have been proposed in recent years. The construction of cross-component timing-based covert channels on heterogeneous CPU-GPU systems was first introduced by Naghibijouybari *et al.* [15]. More recently, several works have proposed exploiting GPU components, such as cache [16], translation lookaside buffer (TLB) [17] and network-on-chip (NoC) micro-architecture [18], to create contention-based covert communication channels.

In summary, the surge of recent work around GPU TEEs and GPU-based attacks highlights the relevance of previously unconsidered resources and their impact on the security of modern computing systems.

III. GPU-based Thermal Covert Channel Attacks

A. Threat Model and Assumptions

Similar to other works in the state-of-the-art TCCs, our proposed GPU-based TCC assumes two attacker applications with specific characteristics. The *transmitter* attacker resides in a secure zone and has access to sensitive information that cannot be communicated through conventional means. This secure environment is commonly supported by commercial technologies, such as ARM TrustZone [19] and Intel soft-guard extension (SGX) [20]. As mentioned in Section II-B, there is a current trend and a real need to include the GPU within TEEs to provide GPU acceleration to heavy-processing algorithms that deal with private information [8]–[11]. In this zone, the transmitter has access to the GPU and requires no extra privileges to execute code in it as part of its program, i.e., the transmitter has been designed to utilize the proper API functions to run parallel kernels in the GPU. Details about our implementation of the attackers are discussed in Section III-B.

The *receiver* attacker, on the other hand, executes in a non-secure zone where it can perform any type of unsupervised I/O operation. The receiver can read the GPU thermal sensor in user space without the need for any privilege escalation, as commercial vendors already provide this feature either through a tool (e.g., the NVIDIA System Management Interface [21]) or directly from the file system through already provided drivers [22], which is often the case for embedded devices.

B. GPU-based Thermal Covert Channel Attack

1) High-level description of the attack: In our new GPU-based TCC, the transmitter application encodes the binary data by increasing or decreasing the temperature signal of the GPU. By doing high parallel processing on the GPU for a certain amount of time t_{up}, the transmitter can raise the temperature, hence encoding a bit value of 1. When the attacker needs to transmit a bit value of 0, it sleeps the GPU for an amount of time t_{down}, allowing the temperature to decrease due to the cooling system (e.g., the fan). The receiver then reads the thermal sensor on the GPU and proceeds to decode the information.

2) Transmitter: The first step in the design of the transmitter is the processing mechanism to encode the binary data as temperature fluctuations.

Sending a bit value of 1 requires some sort of processing on the GPU to raise its temperature. Since GPUs are comprised of hundreds or thousands of small cores, raising the temperature of the whole device requires a high enough parallel load, which is not the case for CPU-based attacks as each physical core in a CPU comes with its thermal sensor. To perform this high parallel processing, we chose to implement a parallel multi-threaded busy waiting kernel. In this kernel, each thread is constantly querying the state of a timer, which controls the period of a binary 1, hence heating the GPU. The number of parallel threads for each computation is extracted empirically offline for the target platform according to the number of computations it requires to produce the minimum valid temperature change at the desired channel transmission rate.

When encoding a bit value of 0 as temperature, we sleep the application for an amount of time equal to half of the period of the transmission data rate. Note that during the cooling phase, other (background) applications might execute on the GPU. This would add some noise to the channel, which is out of the control of the attacker and it is a challenge for GPU-based attacks. The interference that background GPU applications could produce on the channel is an additional design factor for the channel.

To overcome this challenge, we form small packets (e.g., from 8 to 16 bits) comprised of a header (used to identify the beginning of a packet and its sender) and data bits encoded through an error correction code (i.e., Hamming [23]) which helps to improve the reception when in noisy environments. This constitutes another difference over CPU-based TCCs, where the packet size can be extended while keeping low error rates. We experimentally demonstrate the need for a small packet size in Section V-A. Moreover, to avoid the effect of temperature accumulation which may affect the actual reception of packets, we utilize the return-to-zero encoding (RZE) which is commonly used in TCCs exactly for this particular reason [3]. For our experiments, we additionally employ on-off-keying (OOK), as a modulation mechanism.

3) Receiver: The receiver application is modeled as a three-stage process, as seen in Fig. 1. As discussed in Section III-B1, the transmitted packet is encoded as the temperature variations on the GPU thermal sensor. Then, as input, the receiver module takes periodic samples from the thermal sensor of the GPU. As a general consideration, the chosen sampling frequency needs to be set to at least double the channel frequency, as per the Nyquist sampling theorem.

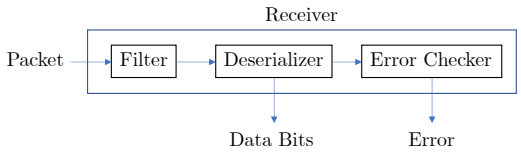

Fig. 1. Receiver model for the time-based attack.

The first stage in the design of the receiver module is a filter, formally described in Eq. (1). This filter smoothens the discrete temperature signal from the thermal sensor and removes the variations due to the sensor's precision δ, which may affect the accuracy of reception. If the magnitude of the difference between the current and the previous temperature sample is greater than δ, the output of the filter is assigned to the current sample. Otherwise, the output is assigned to the previous output sample. In the case of modulated attacks, the output of this filter is then used as an input to a second band-pass filter that eliminates the non-relevant frequency components.

$$y(k) = \begin{cases} x(k), & |x(k) - y(k-1)| > \delta \\ y(k-1), & |x(k) - y(k-1)| \leq \delta \end{cases} \quad (1)$$

The second stage of the receiver module is the de-serializer, depicted as pseudo-code in Alg. 1. This module continuously performs a comparison between the current and the previous measurement of the GPU temperature. When it detects an increase bigger than δ, it appends a bit value of 1 to the packet (line 8). If this difference is not detected within the transmission period (i.e, the inverse of the transmission rate), a *timeout* signal occurs on a parallel timer thread (not shown in Alg. 1). When this happens (line 11), the receiver interprets the sent bit as a 0, appends it to the packet (shown in line 12) and then restarts the timer thread. This process is repeated until the packet is successfully de-serialized. After de-serialization, the header of a packet is checked. Packets composed with a wrong header are discarded. Once a packet with a correct header has been de-serialized, we decode the data bits by using the Hamming ECC, which reports whether there were errors in the received packed.

By using a header of at least two bits (one to indicate the beginning of the packet and one to identify the sender), both malicious applications can act as transmitter and receiver in an acknowledge-based protocol. By doing so, the communication

becomes more robust, since the receiver can ask the transmitter to re-send an erroneous packet.

Algorithm 1 Receiver de-serializer

Input: $currentGPUTemp$, $timeout$, $packetBits$
Output: $packet$, $resetTimer$

```
1: packet ← 0
2: N ← 0
3: prevTemp ← currentGPUTemp
4: while N ≠ packetBits do
5:     nextTemp ← currentGPUTemp
6:     deltaTemp ← nextTemp − prevTemp
7:     if deltaTemp > δ then
8:         packet ← (packet << 1) | 1
9:         N ← N + 1
10:        resetTimer ← 1
11:    else if timeout then
12:        packet ← (packet << 1)
13:        N ← N + 1
14:        resetTimer ← 1
15:    end if
16:    prevTemp ← nextTemp
17:    resetTimer ← 0
18: end while
```

IV. Countermeasures

A. GPU Thermal Noise

In the context of CPUs, the baseband thermal noise technique comes from the execution of background workloads on the CPUs that are suspected to be involved in the channel. When moving to a GPU domain, as previously motivated in Section I, this naïve approach requires further considerations. The first one is the computing required to raise the temperature for the GPU. As discussed, GPUs consist of hundreds or thousands of smaller cores, which need to be heated up. This means that the otherwise innocuous noise translates into a high computing workload on the GPU. This noise will be present as constant background affecting the power of the system and the performance of all other GPU-accelerated applications. The latter would not be the case for CPUs, as only the cores involved in the attack are affected. To improve efficiency, approaches in the state of the art for CPU-based TCCs have suggested frequency-specific (or *narrow-band*) noise [14], as discussed on Section II. By applying background processing at a specific rate, the computational power and performance impact is reduced over constant noise. Similar to the baseband noise, the GPU-based narrow-band noise requires a highly parallel computational workload, but it can be executed at a much higher rate. However, since the GPU is still a shared resource for all GPU-accelerated applications, the contention for it creates performance degradation on all other GPU application.

B. DVFS

Dynamically changing the voltage and frequency levels of the CPU has been shown before as a way to jam CPU-based TCCs as an improvement over noise-based solutions. In the reference state-of-the-art approach for CPU-based TCCs [13], authors suggest a periodic DVFS policy that switches the frequency of the compromised CPU from the highest level to a lower level every channel period T. To establish the frequency values, they employ a down up rate β, computed as the ratio of the time the CPU stays at a low frequency (t_{down}) and the time that it stays at a high frequency (t_{up}). Although this state-of-the-art approach seems to successfully block the attack with a slight performance degradation on other applications that run on the same core, when translating this effect to GPU, the performance loss is much bigger (as it is discussed in Section V-B).

Because of this, we here propose a temperature-aware DVFS policy that seeks to block the attack, while reducing the performance loss on other GPU applications. Our policy is depicted as pseudo-code in Alg. 2. In summary, our policy sets a target temperature defined as the base temperature under normal utilization. Then, while a covert channel is active, we read the current temperature value. If the temperature is higher than the target temperature (adding the sensor's precision δ), we scale down the GPU frequency (line 3) by setting it to the next available lower level (defined by ΔF). On the other hand, if the temperature is lower than the averaged value, we increase the frequency, again by ΔF (line 5). To ensure a correct functionally, we delimit the frequencies to a valid range (line 7). Finally, we wait for $period$ ms to start the process again. In our experiments, we employ an empiric acting $period$ of 5 ms to repeat the process, since it yields to fast reaction to a possible attack. Because of the frequency oscillations caused by constantly increasing and decreasing the frequency, the GPU temperature also oscillates, which affects the expected behavior of the attacker and hence mitigates the attack.

Algorithm 2 Proposed temperature-based DVFS Policy

Input: $targetTemp$, $isActive$, $currentTemp$, $MaxFreq$, $MinFreq$, δ, ΔF
Output: $currFreq$

```
1: while isActive do
2:     if currentTemp > targetTemp + δ then
3:         currFreq ← currFreq − ΔF
4:     else
5:         currFreq ← currFreq + ΔF
6:     end if
7:     currFreq ← clip(currFreq, MinFreq, MaxFreq)
8:     sleep(period)
9: end while
```

V. Evaluation

To evaluate the new GPU-based TCC, we perform a series of experiments on two real-world platforms: a GPU-dedicated general-computing and a GPU-integrated embedded platform. The characteristics of these platforms are shown in Table I. Using these platforms, we proceed to evaluate our new attack, as well as the expanded countermeasures introduced in Section IV.

A. Effectiveness of the New GPU-based Attack

To verify the feasibility and effectiveness of our new GPU-based attack, we launched attacks with different packet sizes and encoding mechanisms, for both platforms described in Table I. In this setup, we did not limit the GPU to any application. This means that the underlying OS and applications use it at any time if needed. In fact, hardware-accelerated GPU scheduling is enabled for graphical interface display in the OS.

TABLE I
CHARACTERISTICS OF THE EVALUATION PLATFORMS.

Characteristic	Platform	
	General computing (PC)	NVIDIA Jetson TX2
GPU	NVIDIA GeForce RTX2070	NVIDIA Pascal
GPU architecture	Turing	Pascal
GPU cores	2304	256
Max GPU frequency	1620 MHz	1302 MHz
Fan speed	100 %	100 %
Thermal sensor resolution (δ)	1 °C	0.5 °C
Sampling frequency	500 Hz	500 Hz
Operating System	Windows 11	Ubuntu 16.04

TABLE II
DESCRIPTION OF THE GPU-BASED ATTACK SETTINGS ON THE EFFECTIVENESS EXPERIMENT

Platform	Sent bits	Encoding	Transmission rates (bps)
PC Jetson TX2	24,064	RZE, RZE + OOK	0.68, 1.38 4.375, 8.75

In this experiment, we run attacks with different encoding and packet sizes, as summarized in Table II. We used an even split of RZE and OOK encoding. With our new attack we achieved maximum transmission rates of 1.38 and 8.75 bps for the general computing and embedded platform respectively. As seen in Table III, this error rate comes with very low BER and PER for small packets (e.g., less than 2 % for our recommended packet size of 12 bits). Note that as previously discussed, increasing the packet size (more than 16 bits in our case), decreases the quality of the channel due to temperature accumulations and background noise.

As a comparison, the first implementation of a CPU-based thermal covert channel [2] achieved a lower transmission rate of 1.33 bps, with a much higher BER of 11 %. This is a fair comparison, as our GPU-based attack is also the first implementation of a TCC of its kind.

B. Evaluation of Countermeasures

As a second evaluation, we perform an experiment where we submit our new GPU-based attack to the extended countermeasures discussed in Section IV. First, we evaluate the proposed extended countermeasures themselves in terms of their capacity to block the attack at the maximum transmission frequency. The evaluated countermeasures are baseband (BB) noise, narrowband (NB) noise [14], the DVFS approach from [13] using $\beta = 1$ (low β, 50 % duty cycle), $\beta = 9$ (suggested for most of their experiments), $\beta = 15.67$ (high β), and our thermal-aware DVFS technique (TAVFS). For the BB noise, we employed the Gaussian kernel from the Rodinia benchmark [24] as the generator application. For NB noise, we implemented a parallel Single-Precision A·X Plus Y (SAXPY) operation as the noise application due to its highly parallel nature and the fact that it can be toggled fast enough, as discussed in Section IV. As the minimum frequency, we employed a value of 300 MHz. The maximum frequency is the same value as shown in Table I. The error rate results for running the attack alongside the countermeasures on the both platforms are depicted in Fig. 2. As it can be seen from the figure, the channel gets severely degraded with the majority of the extended DVFS and noised-based techniques, as the PER rises as high as 98 % for the high β scenario, and around 78 % for our proposed countermeasure.

TABLE III
BER AND PER FOR DIFFERENT PACKET SIZES FOR THE GPU-BASED ATTACK ON THE EVALUATION PLATFORMS

Packet size (bits)	BER (%)		PER (%)	
	PC	Jetson TX2	PC	Jetson TX2
8	0.09	0.25	0.34	0.34
12	0.14	0.29	1.00	1.50
16	0.30	1.15	1.33	4.00
24	1.28	1.73	6.00	9.00
32	4.25	4.82	9.21	14.48

Although our proposed solution produces less PER and BER on the attacker compared to the extended state-of-the-art approaches, in both platforms all meaningful communication is nullified. Moreover, as we discuss further in this section, our temperature-aware DVFS approach trades off the error rates in favor of a reduced performance loss on other applications. Note that on the Jetson TX2 platform, for the low β scenario ($\beta = 1$), both BER and PER are lower that any other solution, resulting in about 50 % of the packets being unaffected by the countermeasure.

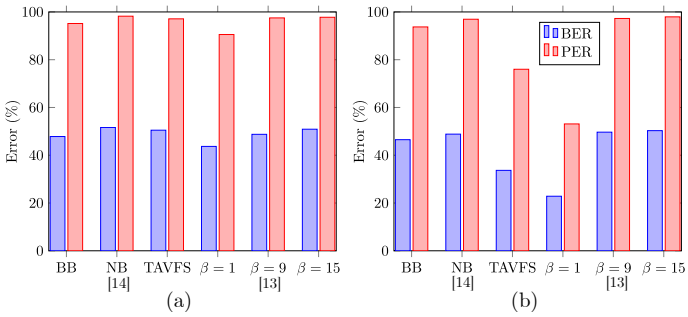

Fig. 2. Average BER and PER for the GPU-based attack under the extended countermeasures on the PC platform (a) and the Jetson TX2 board (b).

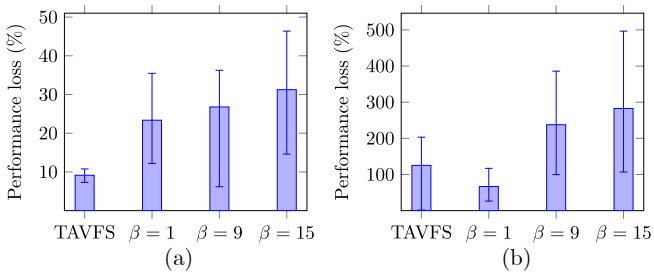

Fig. 3. Performance loss on benchmark applications due to DVFS countermeasures (our solution and the β-based from [13]) on the PC platform (a) and the Jetson TX2 board (b).

Finally, to evaluate the overhead of our new thermal-aware DVFS policy, we compare our solution against the reference DVFS countermeasure from the state of the art [13], as it has proven to be more efficient than the previous noise-based countermeasures. We evaluate our proposed DVFS technique

against the state-of-the-art DVFS approach for the β values mentioned above. To keep consistency with their work, we use the same metric (i.e., performance loss). To this end, we used four applications from the Rodinia benchmark as application set: *streamcluster*, *particlefilter*, *guassian*, and *myocyte*.

In the experiment, we measure the average execution time of the application set without any DVFS, and then we measure again applying each one of the evaluated approaches individually to compute the performance loss due to the countermeasure. We assume the attacker is present at all times ($\tau = 0$ in [13]). The results for the performance loss evaluation are depicted in Fig. 3. Here, the overall average performance loss on the benchmark applications on the PC platform is less than $10\,\%$ for our solution. In this platform, our DVFS policy outperforms the state-of-the-art approach by more than $3\times$ for the high β scenario and more than $2\times$ for the lowest β one.

An interesting outcome of this experiment is the performance loss on the embedded Jetson TX2 platform. Firstly, the performance loss from all the countermeasures increases drastically when compared to the PC platform, reaching almost $300\,\%$ for the high β scenario and around $70\,\%$ for the lowest β. Secondly, the lowest β scenario produces less performance loss than our solution is this platform. However, when analyzing the bit error rates depicted in Fig. 2 (b), the same countermeasure performs poorly compared to our proposed solution for the same platform (i.e., it produces much lower error rates on the attack). This means that our thermal-aware DVFS policy outperforms the state-of-the-art solution in this platform too when factoring both the error rates and the performance loss. Finally, since most of the attacks and countermeasures on TCCs found in state of the art have tackled mostly high-end multi-/many-core systems, we believe more attention should be put into disclosing new TCC embedded attacks, and (more importantly) proposing new countermeasures tailored for embedded devices. We see our new GPU-based attack and countermeasure as a starting point in this direction.

VI. Conclusion

In this paper, we have presented the first approach for a GPU-based thermal covert channel (TCC) attack. We analyzed and evaluated our attack on two different real-world computing platforms to show their general applicability. Our results show for the first time that a GPU-based TCC is feasible with high enough transmission rates and low enough bit and packet error rates that are comparable to existing CPU-based TCCs. Moreover, we have analyzed and extended existing countermeasures for CPU-based TCCs for our new GPU-based TCC attack. Consequently, we demonstrated how our implementation of the extended countermeasures successfully blocks the attack at the expense of added overhead in the system. Finally, we have proposed a new temperature-aware DVFS countermeasure that mitigates the attack while causing $2\times$ less performance loss than the state-of-the-art DVFS countermeasure on a set of benchmark applications.

References

[1] J. Millen, "20 years of covert channel modeling and analysis," in *Symposium on Security and Privacy*, May 1999, pp. 113–114.

[2] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms," in *USENIX Conference on Security Symposium (SEC)*, 2015, p. 865–880.

[3] Z. Long, X. Wang, Y. Jiang, G. Cui, L. Zhang, and T. Mak, "Improving the efficiency of thermal covert channels in multi-/many-core systems," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, April 2018, pp. 1459–1464.

[4] H. Huang, X. Wang, Y. Jiang, A. K. Singh, M. Yang, and L. Huang, "Detection of and countermeasure against thermal covert channel in many-core systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 2, pp. 252–265, 2022.

[5] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "C^{3}apsule: Cross-FPGA covert-channel attacks through power supply unit leakage," in *Symposium on Security and Privacy (SP)*, 2020, pp. 1728–1741.

[6] S. Chen, W. Xiong, Y. Xu, B. Li, and J. Szefer, "Thermal covert channels leveraging package-on-package DRAM," in *IEEE Int. Conf. on Trust, Security and Privacy In Computing and Com./IEEE Int. Conf. on Big Data Science and Eng. (TrustCom/BigDataSE)*, 2019, pp. 319–326.

[7] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh, "Heterogeneous isolated execution for commodity GPUs," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019, p. 455–468.

[8] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on GPUs," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Oct. 2018, pp. 681–696.

[9] R. Liu, L. Garcia, Z. Liu, B. Ou, and M. Srivastava, "SecDeep: Secure and performant on-device deep learning inference framework for mobile and IoT devices," in *International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2021, p. 67–79.

[10] T. Hunt, Z. Jia, V. Miller, A. Szekely, Y. Hu, C. J. Rossbach, and E. Witchel, "Telekine: Secure computing with cloud GPUs," in *USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2020.

[11] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, L. Zhao, F. Yuan, P. Li, Z. Wang, B. Zhao *et al.*, "Enabling privacy-preserving, compute-and data-intensive computing using heterogeneous trusted execution environment," *arXiv preprint arXiv:1904.04782*, 2019.

[12] D. B. Bartolini, P. Miedl, and L. Thiele, "On the capacity of thermal covert channels in multicores," in *European Conference on Computer Systems (EuroSys)*, 2016.

[13] H. Huang, X. Wang, Y. Jiang, A. K. Singh, M. Yang, and L. Huang, "On countermeasures against the thermal covert channel attacks targeting many-core systems," in *Design Automation Conference (DAC)*, 2020.

[14] J. Wang, X. Wang, Y. Jiang, A. K. Singh, L. Huang, and M. Yang, "Combating enhanced thermal covert channel in multi-/many-core systems with channel-aware jamming," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3276–3287, 2020.

[15] H. Naghibijouybari, K. N. Khasawneh, and N. Abu-Ghazaleh, "Constructing and characterizing covert channels on GPGPUs," in *International Symposium on Microarchitecture (MICRO)*, Oct. 2017.

[16] S. B. Dutta, H. Naghibijouybari, A. Gupta, N. Abu-Ghazaleh, A. Marquez, and K. Barker, "Spy in the GPU-box: Covert and side channel attacks on multi-GPU systems," 2022. [Online]. Available: https://arxiv.org/abs/2203.15981

[17] A. Nayak, P. B., V. Ganapathy, and A. Basu, *(Mis)Managed: A Novel TLB-Based Covert Channel on GPUs*, 2021, p. 872–885.

[18] J. Ahn, J. Kim, H. Kasan, L. Delshadtehrani, W. Song, A. Joshi, and J. Kim, "Network-on-chip microarchitecture-based covert channel in GPUs," in *International Symp. on Microarchitecture (MICRO)*, 2021.

[19] ARM, "Building a secure system using trustzone technology," https://documentation-service.arm.com/static/5f212796500e883ab8e74531, 2009, accessed: May 16, 2022.

[20] V. Costan and S. Devadas, "Intel SGX explained," Cryptology ePrint Archive, Report 2016/086, 2016, https://ia.cr/2016/086.

[21] NVIDIA, "Nvidia system management interface," 2022, accessed: May 20, 2022. [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface

[22] Nvidia, "NVIDIA Jetson TX2 NX thermal design guide," 2021, accessed: May 20, 2022. [Online]. Available: https://developer.nvidia.com/embedded/downloads

[23] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[24] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *International Symp. on Workload Characterization (IISWC)*, 2009.