A GPU Covert Channel Based on Workload Manipulation Through the CuPy Library

Gianluca De Lucia[1,2,*,†], Alessio Merlo[1,*,†] and Luca Caviglione[3,†]

[1]CASD - School of Advanced Defense Studies, Rome, Italy
[2]University of Turin, Turin, Italy
[3]Institute for Applied Mathematics and Information Technologies, Genova, Italy

Abstract

The ubiquitous diffusion of Graphics Processing Units (GPUs) accounts for new threats and offensive templates. An emerging attack scheme leverages covert channels, i.e., parasitic communication paths hidden within legitimate flows of data or digital objects. In this perspective, this work explores the feasibility of creating covert channels through the manipulation of the workload offered to the GPU. Specifically, we propose to take advantage of the widespread `CuPy` open source library to encode information in the evolution of the most relevant GPU performance metrics. Results demonstrate the feasibility of the approach and also allow to propose some prime countermeasures to prevent unwanted data leakages and to make modern GPUs more secure.

Keywords

GPU Security, Covert Channel, Cupy, AI

1. Introduction

With the increasing demands for computationally intensive applications, Graphic Processing Units (GPUs) are becoming critical assets to protect. Unfortunately, the most complex ecosystems could be plagued by a wide array of security concerns, such as the lack of documentation in open source tools, difficulties in deallocating data in a secure manner, and subtle vulnerabilities arising from the adoption of virtualization schemes [1]. Even if improving the security posture of high-performance applications is now a major research topic, many issues prevent proceeding quickly or applying pre-existing security schemes over the GPU functional blueprint. For instance, the presence of multiple memory areas with different access rights poses challenges in securing the entire software toolchain, and standard taint analysis techniques may require a non-negligible re-engineering effort [1, 2]. Another important aspect regards the data-intensive nature of GPU-based applications, which often require processing sensitive information or supporting mission-critical decisions. As a significant use case, GPUs are the basis of deep learning and machine learning frameworks and thus make them a sensitive target for inferring behaviors or leaking data [3].

An important class of security threats takes advantage of the multitude of possible side-channel offensive templates or *covert channels* that can be built due to the broad attack surface characterizing the most complex and heterogeneous GPU-based architectures (see, e.g., [1] and [4] and the references therein). In this vein, a relevant corpus of works has emerged to mitigate the impact of covert channels that can leverage the CUDA framework [5] or how they can be combined with other vulnerabilities [6]. Moreover, many research works now focus on how deep learning techniques can be used to "understand" the complex interplay of GPU hardware/software layers to implement effective offensive schemes exploiting a side channel [7, 8, 4, 9, 10].

Since covert channels are increasingly deployed by real-world malware [11] and also demonstrated their effectiveness to elude security frameworks of many hardware/virtualized ecosystems [12, 13],

Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), February 03-8, 2025, Bologna, IT
*Corresponding author.
†These authors contributed equally. The first author developed the software.
gianluca.delucia@stud.unicasd.it (G. De Lucia); alessio.merlo@unicasd.it (A. Merlo); luca.caviglione@cnr.it (L. Caviglione)
0000-0001-7912-2083 (G. De Lucia); 0000-0002-2272-2376 (A. Merlo); 0000-0001-6466-3354 (L. Caviglione)

in this work, we further investigate how GPU covert communications could be simply established by leveraging a popular open source framework. In essence, we take advantage of the `CuPy` library[1] for GPU-accelerated environments to create a controlled load of operations that can alter some behavior of the GPU with a global visibility scope. Specifically, `CuPy`-induced alterations of GPU metrics, such as utilization, power consumption, and temperature, are used as the carrier to encode and transmit information between two different execution kernels. Such a proof-of-concept implementation[2] can be used as the basis to conduct tests for demonstrating how simple mechanisms can endow an attacker with the ability to remain "beneath the radar" of traditional monitoring tools. At the same time, investigating the feasibility of a novel covert communication scheme may open up the development of specific countermeasures and contribute to a deeper understanding of GPU-related security vulnerabilities and their potential implications.

Summing up, the contributions of this work are: *i*) understanding whether shared and unharmful information can be used to encode a proper alphabet to leak data; *ii*) demonstrate the feasibility of implementing a proof-of-concept covert channel by using the `CuPy` library; *iii*) provide a preliminary performance evaluation assessment.

The remainder of the paper is structured as follows. Section 2 introduces the attack model, Section 3 shows the design of the workload-based covert channel, and Section 4 discusses its implementation details. Section 5 presents numerical results obtained through laboratory tests, while Section 6 concludes the article and outlines possible future research directions.

2. Background and Attack Model

A covert channel is a hidden communication path that allows two endpoints to secretly exchange data or bypass blockages, security policies, or execution enclaves [14, 15]. To this end, the secret sender and receiver must agree on a shared resource, which constitutes the carrier. Possible examples of carriers are an unused protocol field, data used to pad a software object, or physical behavior such as the core(s) temperature that make up a CPU [16]. With the increasing complexity of modern hardware/software ecosystems, covert channels now represent a significant vulnerability as they can evade traditional security controls such as firewalls, access policies, or data monitoring systems [17, 18]. Among the various implemented threat models, covert channels are primarily used to exfiltrate sensitive data, such as cryptographic keys or small chunks of high-value information. For instance, they can bypass the security of hypervisors and establish cross-virtual-machine communication paths [19] or circumvent isolation approaches deployed in cloud infrastructures [13].

In general, evaluating the performance of a covert channel involves three tightly-coupled basic properties [20]: *i*) the *steganographic bandwidth*, which quantifies the volume of information transmitted per unit of time; *ii*) the *robustness*, which refers to the ability of the channel to maintain effective communication despite noise, (unwanted) system variations, or external interference; *iii*) the *undetectability* assessing how the channel blends into the expected behavior of the abused host/device.

To create channels targeting the same host, leveraging shared resources commonly used by users and system processes, e.g., CPU caches, GPU metrics, or the system memory, is a viable idea. Accordingly, such quantities can be directly manipulated or influenced to encode and transmit information between two processes.

Figure 1 depicts the general attack model. Specifically, we consider two execution kernels that want to communicate, i.e., they represent the secret sender and the secret receiver. Without loss of generality, the secret endpoints could be co-located or implemented within two GPU processes. To effectively implement such an attack scheme, the sender and receiver must agree on an alphabet/encoding. A possible approach leverages the workload offered to the various GPU cores, which accounts for the visible alteration of shared resources. For instance, producing a burden of operations will cause an increase in the GPU temperature, leading to a globally observable behavior. Hence, the sender could

[1] CuPy homepage: https://cupy.dev
[2] GitHub repository: https://github.com/gigernau/Cupy-Covert-Channel

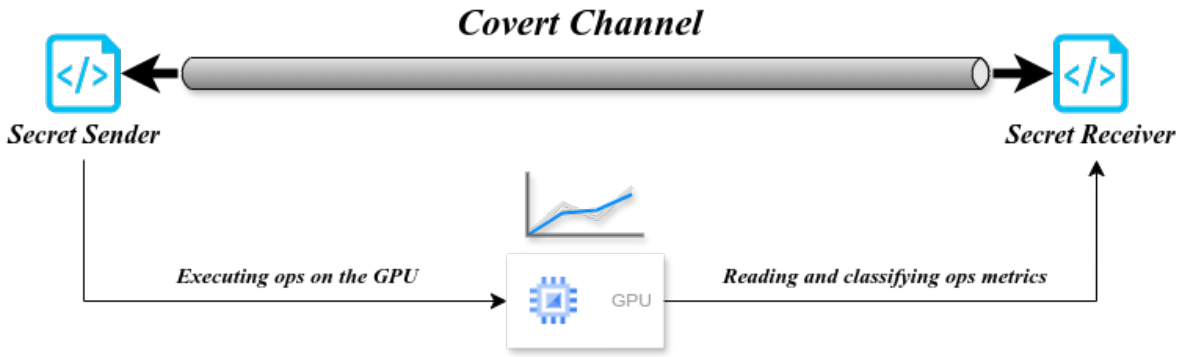

Figure 1: General attack model considered in this work for creating a covert channel in a GPU.

encode the binary value `1` by applying a load to increase the temperature beyond a given threshold. In contrast, the binary value `0` could be encoded by not performing operations. The receiver may then infer a message by inspecting the temporal evolution of the targeted shared metric of the GPU, i.e., the temperature.

For the specific case of the proof-of-concept implementation proposed in this work, covert communication is made possible since the secret sender performs operations directly mapped to binary values that will influence the visible behavior of the GPU. Such operations allow for building a shared alphabet. The receiver infers the binary digit to decode the secret value by classifying a specific GPU trait observed through a given time window. In other words, a covert communication (see Figure 1) is made possible by two disjoint/distinct GPU processes, where the first one (i.e., the sender) executes operations to force specific metrics on the GPU, while the second one (i.e., the receiver) reads and classifies the metric, understanding the hidden operation.

Therefore, in this work, we consider a malicious threat actor who wants to implement the general attack scheme of Figure 1 through GPU cores and uses as the carrier for secret data GPU metrics such as temperature, power consumption, and percentage of usage. As detailed later, the first phase of this attack chain should consider some form of reconnaissance, i.e., the attacker "sniffs" metrics to understand the action-reaction relationship between a load of operations and a global GPU behavior.

3. Covert Channel Design

As hinted, to design the covert channel, we referenced the `CuPy` library since it allows us to produce burdens of operations straightforwardly to indirectly influence the behavior of a shared resource. This choice is motivated by `CuPy` providing a simple Python interface towards CUDA and operating directly on the GPU cores. Establishing a covert communication path involves the transmission of a binary digit composing a secret message. To this aim, each bit should be mapped against one or more `CuPy` operations, leading to a specific signature in global GPU metrics that the receiver can infer. In more detail, in this work, we focus on encoding the information by altering the following GPU metrics: temperature [°C], power usage [watt], GPU usage [%], memory usage [Mb], cache usage [%], and the frequency of the video clocks [MHz].

To design a suitable "secret alphabet", we performed several tests to understand possible `CuPy`-operations-to-metric mappings. As an example, Figure 2 showcases how the GPU metrics vary when a burden of operations is created by a process executing the `cupy.dot` function, which is responsible for performing a dot product of two arrays. Instead, Figure 3 reports a similar investigation for the `cupy.mean`, which computes the arithmetic mean over a sequence. As it can be noticed, only `cupy.dot` leads to some visible alterations, which can be used to encode information. When using `cupy.dot`, the selected metrics seem insensitive to the load. Despite the used `CuPy` functions, the usage of resources should be coherent with the undetectability of the channel, for instance, to not cause delays or lags at a system-wide level that can reveal that the covert communication attempt is ongoing.

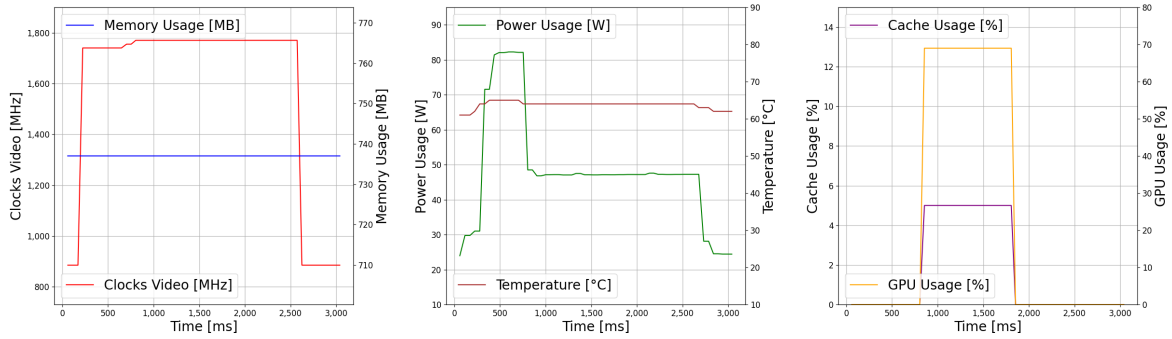

Figure 2: Patterns within the selected metrics when the `cupy.dot` function is invoked.

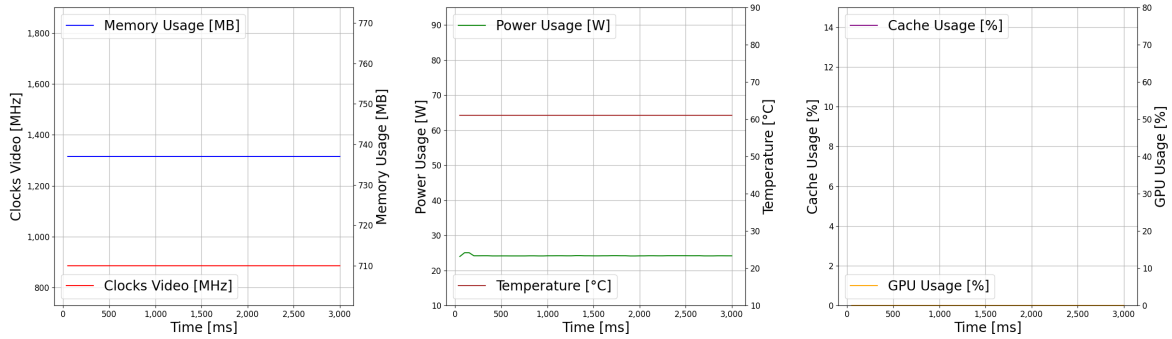

Figure 3: Patterns within the selected metrics when the `cupy.mean` function is invoked.

As a final remark, we point out that part of the design pipeline at the basis of this covert channel may resemble a side-channel attack. The proposed encoding/decoding process leverages a mapping between the `CuPy`-generated workload and the behavior of the metric (e.g., the power usage as depicted in Figure 2). However, our reference scenario is different from classic side-channel attacks since we are not assuming the presence of an external malicious entity, see, e.g., [21] and the references therein for the case of inferring data through power consumptions of GPUs. Instead, we consider two colluding GPU processes wanting to transfer information even without a direct software/hardware shared path [1].

4. Implementation and Testbed

We decoupled the transmission and receiving phases to prepare a proof-of-concept implementation of the proposed covert channel targeting GPU architectures. Specifically, the transmitting process (i.e., the secret sender) is responsible for encoding the hidden message within a variable amount of operations generated via functions of the `CuPy` library. As regards the receiving process, we implemented two main functional components. The first is wrapped around the system management interface offered by NVIDIA for collecting information on its hardware. Thus, we used the `nvidia-smi` command to collect all the metrics discussed in Section 3. The second is the decoding stage, which relies upon artificial intelligence tools to prevent the need to develop suitable decoding rules.

The adopted AI-based pipeline has been implemented by using the Random Convolutional Kernel

Transform method[3] to extract temporal patterns from the considered GPU metrics [22, 23]. To decode the binary digits `1` and `0`, we took advantage of the RidgeClassifierCV model[4], mainly owing to its automatic regularization capability via cross-validation. Although this approach may seem excessive for a proof-of-concept, it allowed us to quickly validate the possibility of decoding the metric patterns induced by `CuPy` operations. Replacing the AI-based decoding stage with lighter heuristics, possibly optimized for real-time processing, is part of our future research.

Implementing the overall testbed required to face several technical challenges. The main one deals with the synchronization between the secret sender and the covert receiver (i.e., the two processes in charge of implementing the covert communication approaches). In fact, without proper timing constraints, metric patterns from consecutive operations could overlap, thus causing a misalignment that prevents the decoding of the secret message. To fix this issue, controlled delays between the various `cupy.dot` operations were introduced to guarantee adequate temporal separation and prevent the need for more sophisticated synchronization schemes. Another difficulty we had to face was the "noise" generated by competing processes on the GPU. In this case, the volume of operations executed over the various execution cores could degrade the "envelope" conveying the information needed to implement the covert channel. As a prime workaround, experiments have been done in a partially-controlled environment. Nevertheless, non-deterministic variations in the behavior of the GPU required to repeat the execution of operations to obtain consistent averaged data, useful for both model training and analysis. Testing channels in more challenging environments or scenarios is part of our ongoing research.

The obtained dataset consists of 300 different runs of the following `CuPy` operations: `matmul`, `dot`, `linalg`, `sort`, `transpose`, `sum`, `sin`, `exp`, `log`, `rand`, `mean`, and `std`. For each run, we sampled values with a rate of 3 seconds, which allowed us to avoid overlaps and guaranteed a proper stabilization of the measured metrics. The resulting dataset has been split so that the 70% of the entries were used for the training phase, whereas the remaining 30% was used for tests.

The sender process has been implemented with a Python script and the `CuPy` library. In contrast, the receiver process exploits the `Scikit-learn` [5] and `Scikit-time` [6] frameworks. For this preliminary investigation, we did not further tune the model hyperameters, since they already provided satisfactory results. To perform tests, we used a host with Intel Core i7-9750H and 8 Gbyte of RAM and an NVIDIA RTX 2060 with 30 trace cores, 240 tensor cores, $1,920$ CUDA cores, 6 Gbyte of RAM, and a clock frequency in the $960 - 1,200$ MHz range. The host ran Linux with Ubuntu 24.04 LTS. The various `CuPy` operations were repeated several times on matrices of $4,000 \times 4,000$ elements for proper statistical relevance. This allowed the generation of workloads capable of altering the metric sensibly, i.e., to produce proper patterns.

5. Experimental Results

In this section, we present results obtained through our proof-of-concept implementation. First, we investigate the "best" metrics to implement covert communications, especially from the perspective of developing a suitable AI-based receiver. Then, we focus on analyzing the obtained covert channel. Lastly, we briefly introduce some countermeasures and mitigation techniques.

5.1. Feature Selection

GPU metrics serve as input features for training the classification model. To build an efficient classifier, feature selection was performed by using feature importance and permutation importance, which are

[3] Random Convolutional Kernel Transform reference documentation: https://www.sktime.net/en/latest/api_reference/auto_generated/sktime.transformations.panel.rocket.Rocket.html

[4] RidgeClassifierCV reference documentation: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeClassifierCV.html

[5] Scikit-learn reference documentation: https://scikit-learn.org/stable/

[6] Scikit-time reference documentation: https://www.sktime.net/en/latest/index.html

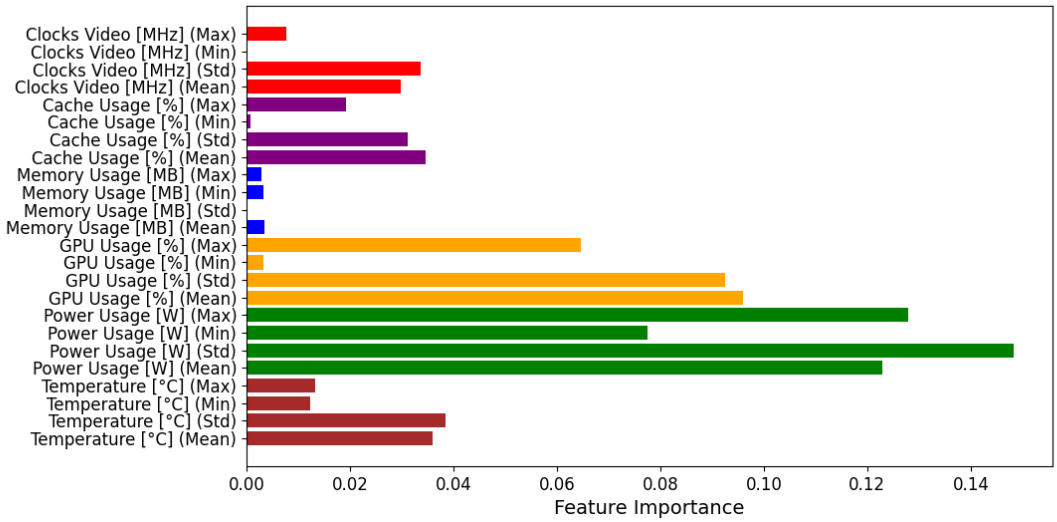

Figure 4: Feature Importance.

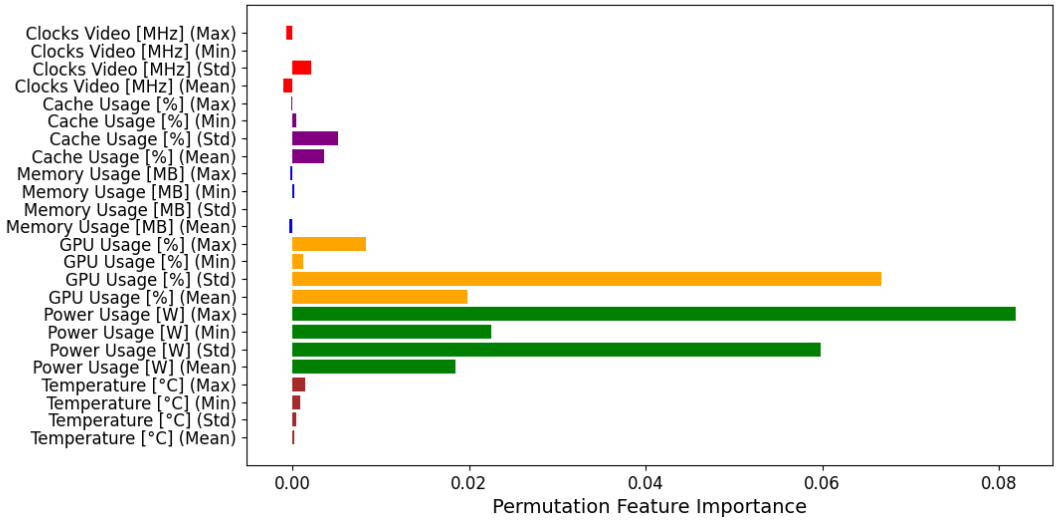

Figure 5: Permutation Importance.

defined as follows:

- **Feature Importance**: it quantifies the relative contribution of each GPU metric to the correct classification of GPU operations according to four statistical values: mean, standard deviation, minimum, and maximum. The relationship between values and GPU metrics is calculated through a Random Forest classifier that can handle high-dimensionality data and directly provide feature importance scores for each statistical value. The importance score of each GPU metric is calculated by averaging its statistical values. Figure 4 suggests that the most important are the mean power usage and the mean GPU usage.
- **Permutation Importance**: it evaluates the relevance of the metrics by measuring the accuracy drop when the feature values are shuffled randomly. Significant accuracy losses imply critical features for classification. According to Figure 5 power and GPU usage are the most relevant indicators for the classification task.

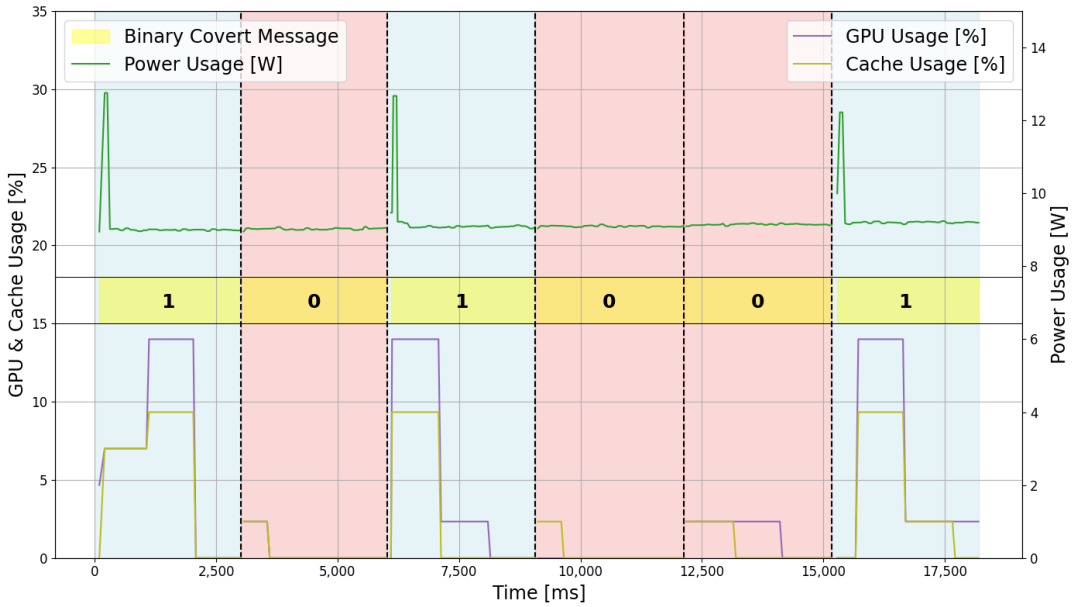

Figure 6: Temporal evolution for an instance of a covert communication.

Table 1 shows the accuracy of the classifier when used against the various CuPy operations. Specifically, the table reports the bit-to-operation mapping, which is the result of encoding a single binary digit with a specific CuPy function, e.g., `cupy.dot` to encode the digit `1`. As shown, the best encodings are those with an accuracy higher than 60%, denoted in bold in the table. Specifically, best results are achieved when the covert channel is built via operations producing a relevant burden on the GPU, e.g., the `cupy.dot` and the `cupy.matmul`. The table also reports "advanced" mappings, i.e., when different binary values are associated with different CuPy functions. For instance, an accuracy higher than 99% is achieved by encoding the value `1` with the `cupy.matmul` and the value `0` with the `cupy.sort`.

Single Operation Encoding		**Per-digit Encoding**	
Name	**Accuracy**	**Name**	**Accuracy**
`dot`	**68.71%**	`matmul, dot`	56.45%
`exp`	29.03%	`matmul, linalg`	81.45%
`linalg`	**90.32%**	`matmul, sort`	**99.84%**
`log`	29.68%	`matmul, transpose`	**99.19%**
`matmul`	**61.61%**	`dot, linalg`	82.26%
`mean`	22.90%	`dot, sort`	98.39%
`rand`	24.52%	`dot, transpose`	**99.68%**
`sin`	26.13%	`linalg, sort`	**99.84%**
`sort`	**99.68%**	`linalg, transpose`	**99.19%**
`std`	44.84%	`sort, transpose`	**99.84%**
`sum`	36.77%		
`transpose`	**61.94%**		

Table 1
Accuracy of single operation encodings and per-digit encodings.

5.2. Covert Channel Analysis

To showcase the behavior of the proposed hidden communication approach, Figure 6 depicts an example of a transmission cloaked within three different GPU metrics when the covert endpoints operate in a controlled environment. Specifically, the example considers the exfiltration of the `101001` message. Each digit has been encoded via suitable operations generated by the `cupy.sort` and `cupy.dot` functions

Encoding {1, 0}	Transmission Time [s]	BER [%]
matmul-sort	50.7	12.5
matmul-transpose	50.7	12.5
dot-transpose	52.3	50.0
linalg-sort	50.8	37.5
linalg-transpose	50.7	50.0
sort-transpose	**50.7**	**0.0**

Table 2
Performance of the covert channel with 8-bit long messages. Best results are in bold.

for the binary value 1 and 0, respectively. As shown, the cache, GPU, and power utilization GPU metrics suggest the presence of some encoded hidden data. The receiver exploits this behavior to decode the secret but leaves a visible signature that could be exploited to reveal the presence of covert communication. However, the various patterns will be less detectable when the attacker operates in a more realistic environment, i.e., when other processes compete for the GPU.

In this vein, we also performed a round of tests in a more realistic scenario. Specifically, during the covert communication, the host is loaded with user-defined operations, such as using Google Chrome to simulate a browsing session. Despite being simple, this configuration allowed us to conduct a prime assessment of the robustness of the channel. Evaluating the channel in more realistic settings is left as a future development. Table 2 provides results when the covert endpoints exfiltrate a 8-bit long message. As shown, the time needed to transmit the secret information (i.e., the steganographic bandwidth) is insensitive from the used CuPy functions, mainly due to the pseudo-syncing mechanism used to "decouple" the patterns within the metrics for preventing overlaps. Instead, the BER is highly influenced by the encoding scheme. In our setting, best results are achieved when the binary digit 1 is encoded through a volume of operations performed with the `cupy.sort` and the digit 0 with the `cupy.transpose`.

5.3. Covert Channel Mitigation

An essential aspect concerns the mitigation of threats taking advantage of covert channels targeting the GPU. Our findings prove the intuition that having a resource with a "global" visibility should be considered a suitable carrier for hiding information (see, e.g., for the case of virtualized software components sharing portions of the `/proc` filesystem [24]). Therefore, countermeasures could be designed to act at different software layers. For instance, the most important computing libraries (i.e., CuPy in our work) could be checked at the design stage to identify and limit the behaviors that can be manipulated by disjoint processes sharing some visibility properties [25]. Moreover, according to the security requirements, most sensitive deployments could use hardened device drivers that introduce some variability in the scheduled operations, for instance, when they have a commutative relation. Another possible approach is to make the overall OS more secure, e.g., by not reporting precise GPU metrics. In our case, it could be sufficient to patch the `nvidia-smi` command for returning noisy data to untrusted processes.

When eliminating the channel is impossible, detecting colluding GPU processes at runtime would be desirable. To this aim, the results obtained when designing the AI-based detector can play a major role (see Section 5.1). In fact, despite being simple, the model trained demonstrated its effectiveness in decoding the hidden information starting from patterns within the various GPU metrics. Hence, suitable models could be deployed to recognize processes trying to exfiltrate data via a covert channel [17, 18].

6. Conclusions

In this paper, we presented a new, proof-of-concept implementation of a covert channel targeting GPU architectures. To this aim, we exploited the CuPy library to create "suitable" signatures within

globally-observable metrics (e.g., the % of used cache) to encode secret information. Results indicate the feasibility of this class of covert attacks, especially when using the `cupy.sort` and `cupy.transpose` functions. The limitations of this prime investigation are the use of a partially controlled environment and the adoption of an AI-based decoder. Removing such constraints is part of our current research work.

A relevant ongoing research item focuses on creating suitable countermeasures against threats endowed with covert communication capabilities. To this aim, we are working towards a more robust implementation (e.g., through a larger encoding alphabet based on additional `CuPy` operations) to have more advanced test conditions. For instance, a viable approach concerns the introduction of random "fluctuations" on the GPU workload to disrupt the secret data or impair the various hidden communication paths. Yet, this approach should be carefully engineered, as it could degrade the performance of legitimate operations flows. Another idea is to act at the device driver/software level, for instance, by not reporting precise GPU metrics (e.g., by patching the `nvidia-smi` command) to process not considered trusted or by reducing its accuracy.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] Mittal, Sparsh and Abhinaya, SB and Reddy, Manish and Ali, Irfan, A survey of techniques for improving security of gpus, Journal of Hardware and Systems Security 2 (2018) 266–285.

[2] Zhu, Zhiting and Kim, Sangman and Rozhanski, Yuri and Hu, Yige and Witchel, Emmett and Silberstein, Mark, Understanding the security of discrete GPUs, in: Proceedings of the General Purpose GPUs, 2017, pp. 1–11.

[3] Tan, Sijun and Knott, Brian and Tian, Yuan and Wu, David J, CryptGPU: Fast privacy-preserving machine learning on the GPU, in: 2021 IEEE Symposium on Security and Privacy (SP), IEEE, 2021, pp. 1021–1038.

[4] Hettwer, Benjamin and Gehrer, Stefan and Güneysu, Tim, Applications of machine learning techniques in side-channel attacks: a survey, Journal of Cryptographic Engineering 10 (2020) 135–162.

[5] Pietro, Roberto Di and Lombardi, Flavio and Villani, Antonio, Cuda leaks: a detailed hack for cuda and a (partial) fix, ACM Transactions on Embedded Computing Systems (TECS) 15 (2016) 1–25.

[6] Lee, Sangho and Kim, Youngsok and Kim, Jangwoo and Kim, Jong, Stealing webpages rendered on your browser by exploiting GPU vulnerabilities, in: 2014 IEEE Symposium on Security and Privacy, IEEE, 2014, pp. 19–33.

[7] Dutta, Sankha Baran, Covert-and Side-Channel Attacks on Integrated and Distributed GPU Systems, University of California, Riverside, 2022.

[8] Sankha Baran Dutta and Hoda Naghibijouybari and Arjun Gupta and Nael B. Abu-Ghazaleh and Andrés Márquez and Kevin J. Barker, Spy in the GPU-box: Covert and Side Channel Attacks on Multi-GPU Systems, 2022. URL: https://api.semanticscholar.org/CorpusID:247793965.

[9] Kim, Hodong and Hahn, Changhee and Kim, Hyunwoo J and Shin, Youngjoo and Hur, Junbeom, Deep learning based detection for multiple cache side-channel attacks, IEEE Transactions on Information Forensics and Security (2023).

[10] Wei, Junyi and Zhang, Yicheng and Zhou, Zhe and Li, Zhou and Al Faruque, Mohammad Abdullah, Leaky dnn: Stealing deep-learning model secret with gpu context-switching side-channel, in: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2020, pp. 125–137.
[11] Caviglione, Luca and Mazurczyk, Wojciech, Never mind the malware, here's the stegomalware, IEEE Security & Privacy 20 (2022) 101–106.
[12] Cabaj, Krzysztof and Caviglione, Luca and Mazurczyk, Wojciech and Wendzel, Steffen and Woodward, Alan and Zander, Sebastian, The new threats of information hiding: The road ahead, IT professional 20 (2018) 31–39.
[13] Betz, Johann and Westhoff, Dirk and Müller, Günter, Survey on covert channels in virtual machines and cloud computing, Transactions on Emerging Telecommunications Technologies 28 (2017) e3134.
[14] Saenger, Jens and Mazurczyk, Wojciech and Keller, Jörg and Caviglione, Luca, VoIP network covert channels to enhance privacy and information sharing, Future Generation Computer Systems 111 (2020) 96–106.
[15] Wendzel, Steffen and Zander, Sebastian and Fechner, Bernhard and Herdin, Christian, Pattern-based survey and categorization of network covert channel techniques, ACM Computing Surveys (CSUR) 47 (2015) 1–26.
[16] Bartolini, Davide B and Miedl, Philipp and Thiele, Lothar, On the capacity of thermal covert channels in multicores, in: Proceedings of the Eleventh European Conference on Computer Systems, 2016, pp. 1–16.
[17] Caviglione, Luca, Trends and challenges in network covert channels countermeasures, Applied Sciences 11 (2021) 1641.
[18] Makhdoom, Imran and Abolhasan, Mehran and Lipman, Justin, A comprehensive survey of covert communication techniques, limitations and future challenges, Computers & Security 120 (2022) 102784.
[19] Zhang, Yinqian and Juels, Ari and Reiter, Michael K and Ristenpart, Thomas, Cross-VM side channels and their use to extract private keys, in: Proceedings of the 2012 ACM conference on Computer and communications security, 2012, pp. 305–316.
[20] Mazurczyk, Wojciech and Caviglione, Luca, Steganography in modern smartphones and mitigation techniques, IEEE Communications Surveys & Tutorials 17 (2014) 334–357.
[21] Luo, Chao and Fei, Yunsi and Luo, Pei and Mukherjee, Saoni and Kaeli, David, Side-channel power analysis of a GPU AES implementation, in: 2015 33rd IEEE International Conference on Computer Design (ICCD), IEEE, 2015, pp. 281–288.
[22] Bier, Agnieszka and Jastrzebska, Agnieszka and Olszewski, Paweł, Variable-length multivariate time series classification using ROCKET: a case study of incident detection, IEEE Access 10 (2022) 95701–95715.
[23] Dempster, Angus and Petitjean, François and Webb, Geoffrey I, ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels, Data Mining and Knowledge Discovery 34 (2020) 1454–1495.
[24] Zuppelli, Marco and Guarascio, Massimo and Caviglione, Luca and Liguori, Angelica, No Country for Leaking Containers: Detecting Exfiltration of Secrets Through AI and Syscalls, in: Proceedings of the 19th International Conference on Availability, Reliability and Security, 2024, pp. 1–8.
[25] Kemmerer, Richard A, Shared resource matrix methodology: An approach to identifying storage and timing channels, ACM Transactions on Computer Systems (TOCS) 1 (1983) 256–277.

A. Online Resources

The sources for the proof-of-concept implementation of the GPU covert channel are available online at:

- GitHub: Covert Channel.