A Survey of Vulnerabilities in GPUs: Related Attacks, Threats, Defenses, and Fixes

Nick Arellano   
*Fisher College of Science and Mathematics*  
*Towson University*Towson, United States of America  
narella1@students.towson.eduNathan Ketterlinus   
*Fisher College of Science and Mathematics*   
*Towson University*   
Towson, United States   
nketter1@students.towson.edu  
Ayomide Owoade   
*Fisher College of Science and Mathematics*  
*Towson University*   
Towson, United States   
aowoade1@students.towson.edu

*Abstract*—As the demand for graphical computing power has skyrocketed over time, the amount of consumer-grade electronics with discrete Graphics Processing Units (GPUs) has risen to match this demand. As GPUs become a more integral part of the average person’s devices, it becomes critical that they do not present themselves as a possible attack vector for malicious actors. This paper will examine various vulnerabilities and backdoors that exist in consumer-grade GPUs, and consider various solutions to these vulnerabilities, weighing the pros and cons in an ever-evolving cybersecurity landscape.

Keywords—GPU, Vulnerabilities, Side-channel Attack, Review, Deep Learning, Encryption

# Introduction

Graphics Processing Units (GPUs) emerged from the rising popularity of interactive visual media. In order to render visuals without impacting the CPU, the GPU was designed to be a dedicated processor for executing massive amounts of vertex and fragment calculations across massive amounts of data. The Single Instruction Multiple Data (SIMD) parallelism of GPUs made the unit a target for utilization by the wider scientific communities [1], but the difficulty in casting their data as vertexes in specialized calculations required an additional layer of abstraction to properly leverage the processing power of GPUs. The CUDA language bridged this gap and lead to an increasing amount of use-cases as additional libraries were developed and introduced by a field of scientists specialized in GPU programming [2]. Currently these uses-cases include a myriad of rapidly evolving fields: Machine Learning (ML), Deep Learning, Cloud Computing, Particle Simulation,. These fields of computer science will hopefully be even further utilized to develop technologies that will benefit and revolutionize every area of society they find use in, and even now they see wide-spread use across many industries and sciences impacting everyday society.  
 As such, these technologies that leverage the SIMD processing of GPUs must have a solid foundation that is not susceptible to the violation of the confidentiality, integrity, or availability of the data being processed, or the software and hardware composing it. Essentially, a GPU must be secured from potential threats and attackers with the goal of exploiting vulnerabilities to compromise this foundation. With this in mind, it is important to note that GPUs will see an increasing use in heterogeneous computer systems that seek to utilize specialized hardware for better throughput. The usage of GPUs in security services is growing.[3] Additionally, the GPU must be trusted in systems in every possible use case. This includes being designed to handle massive amounts of data that will be subject to legal scrutiny such as Personally Identifiable Information (PII) and Personal Health Information (PHI). This also includes data used in Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN) where a lack of data integrity could causes errors when these networks must make decisions, potentially in high risk or high consequence situations like identifying harmful tumors or recognizing pedestrians.  
 Vulnerabilities in GPUs and attacks that leverage them are seeing a slow increase in recognition in the past 8 years by researchers in the cyber-security and computer science fields as the importance of DNN in practical applications such as image processing, predictions, and pattern recognition in industry, academia, and military began to rapidly increase. A particularly early example of General-Purpose computing on GPUs (GPGPU) can be found here [4], as early as 2006, but analysis on GPU based vulnerabilities don’t see an emergence until the mid-2010’s, coinciding with the practical application of Deep Learning across fields outside of computer science. The key problem here is that development and implementation of technologies reliant on GPUs are outpacing the wide-spread recognition of the risks and vulnerabilities that need to accounted for when utilizing GPUs, particularly outside the field of computer science. It is the very parallelism that is so widely being leveraged by many fields that could make a GPU a high valuable target for attackers targeting private citizens to public institutions, where-in a compromised GPU’s resources may be used by malware to rewrite itself to avoid detection and then spread itself across the entire system. It is also acknowledged that due to micro-architectural similarities, GPUs face similar vulnerabilities to those faced by CPUs, and it was found in [5] that GPUs were susceptible to side channel attacks (SCA) supporting the need for computer architecture security researchers to extend their awareness beyond the CPU.   
 In an effort to provide a primer on GPU vulnerabilities, attacks, and how to defend against them or strengthen systems that rely on GPUs for inquiring scientists and others who hope to take advantage of the neural networks being powered by the capabilities we have consolidated research on related attacks, threats, defenses, and fixes. Due to the central role of GPUs in enabling the flourishing field of neural networks and deep learning we have placed a particular emphasis on research that targets these models. Additionally we place emphasis on SCAs. Although GPUs have little publicly available documentation, allowing a sort of natural defense through requirement of expert knowledge, the research we cover shows these attacks are very much a possibility. It would be naïve to believe the lack of real world instances of GPU-based attacks will mean potential threats won’t target GPUs or use them as a vector in the future. Rather than complacency, it is our hope that this article promotes further research into possible vulnerabilities and the development of technologies and techniques to combat them.

## Introduction to GPU Architecture

As stated above, the broad appeal of a GPU lies within its parallelism. To utilize SIMD processing to its fullest extent, a GPU must be structured in a particular way. While there are several valid, widely used GPU architectures, this paper will quickly go over the Fermi architecture to highlight the basic principles at play. Figure 1 provides a diagram of a sample GPU, as provided by [6].

GPUs divide computational work into units called threads. 32 threads combine to make a warp. GPUs contain many different cores, called Streaming Multiprocessors (SMs), which process warps. Fermi architecture-based GPUs typically contain 15 SMs, which each have 32 SIMD execution units, 16 load/store units, and 4 special function units. 2 warp schedulers and 2 instruction dispatch units issue independent instruction from different warps simultaneously. GPUs also commonly have a memory coalescer unit, which allows for more efficient retrieval of information stored in memory. When more than one thread requests the same information from memory, the requests are combined. This memory coalescing unit, while improving performance drastically, introduces several vulnerabilities that are exploited by the attacks listed below [7] [8].

A diagram of a computer network

Description automatically generated

Fig. 1. Sample GPU architecture, as seen in [6].

## What is a Side Channel Attack?

A side channel attack (SCA) is a passive Electrical-level attack [9]. It is a method of extracting sensitive data from a system, such as Advanced Encryption Standard (AES) keys, keystrokes, or search history, to name a few. The data is extracted in a way that leverages the physical or microarchitectural attributes of hardware components, such as their power consumption, electromagnetic emanation, timing information, and sound, among others [10], [11]. SCAs typically have two components: the transmitter (typically in a trojan that has snuck into a secure zone), and the receiver (in an unprivileged area) [12]. The receiver is programmed to listen to and interpret physical characteristics of the hardware as a binary output. By inserting a trojan that can manipulate the hardware in a way that the receiver can listen to, a side channel is established, where sensitive info that the transmitter has access to can be smuggled out.

As a quick case study of a side channel attack, we will consider Wang and Zhang’s Correlation-based SCA [13]. Their attack leverages the fact that there is a linear proportionality between kernel execution time and the number of unique cache lines requests generated during a kernel’s execution. Because these cache line requests are dependent on input data and an AES encryption key, the key can be reverse engineered (and thus recovered) by encrypting enough files while analyzing and profiling the GPU. Figure 2 shows the results of this attack for each of the 16 bytes of the encryption key. Notice the singular, massive jump in correlation for each of the 16 bytes, these are the values of the key that were successfully recovered by this method.

A screenshot of a graph

Description automatically generated

Fig. 2. Correlation analysis result of 10000 samples, as seen in [13].

This attack outline provides a simple implementation of a side channel attack, in hopes to familiarize the reader with what it may look like in context.

## What is AES Encryption?

AES is a widely used symmetric encryption algorithm designed to secure sensitive information. It serves a cryptographic standard for protecting data confidentiality and integrity. Symmetric encryption algorithms use a single key for both encryption and decryption. Security is AES encryption’s main aspect, ensuring that even if someone were to try intercept the encrypted data, it would be infeasible to deduce the original content without with the correct key. Side channel attacks exploit information leaked during the computation process, such as power consumption patterns, to infer the sensitive data like encryption key. So, it’s important to emphasize the importance of addressing such vulnerabilities to enhance the overall security of parallel computing hardware systems [14]

AES Encryption consists of a variable number of rounds, depending on key size. For the three most common key sizes: 128, 192 and 256 bits, AES has 10, 12 and 14 rounds, respectively. For AES-128, one block of data is organized as a 4x4 array of bytes, termed the state. Each round is a sequence of four operations: SubByte, ShiftRow, MixColumn, and AddRoundKey, except for the initial and last rounds. The initial round only has an AddRoundKey, and the last round excludes Mixcolumn. All the round keys are derived from a single initial key [10], [15], [16].What has also been implemented in GPUs are current cryptographic algorithms but they are not fully able to benefit from the GPU’s abilities[17]

# Attacks

## Memory vulnerabilities

One of the first studies focused on security vulnerabilities present in GPU’s while presenting possible attacks that take advantage of them and possible solutions was presented by Lee et al. [18]. In it, a crucial flaw in GPU security is identified and then an attack model is subsequently described. They found that GPUs leave their memory easily accessible by outside APIs. An attacker can easily access residual data in the memory of a GPU that usually isn’t cleared or emptied out. In the three cases specified the GPU was either not initializing pages that were previously used so they would contain previously stored memory, had portions of memory that could not be deleted by the user or the GPU while being accessible, or there is a lack of division between primate and local memory. All three of these cases are violations of confidentiality in the attack scenario presented where the attacker is assumed to be a lower-level user on a shared system who can now retrieve this left over data through an attack. An attacker could use memory dumps of websites using their own PC then compare it the memory dump from the victims GPU. This attack is a bit unwieldy as it requires a collection of memory dumps of websites using the same GPU as the victim. The attack case makes it easy to have the same GPU, but as Lee et al. points out, there is a large overhead in having enough website dumps to recreate a victim’s browsing history. In lieu of this, another attack is proposed where the attacker compares the memory dump from the victim’s GPU. While less reliable than the previous attack, correctly inferring at ~50% on NVIDIA GPUs, it is still considered significant.

While many attacks revolve around SCA due to their malleable nature there has been other research into how novel attacks can utilize GPU memory as a vector to undermine deep learning frameworks. This is exactly what Park et al. [19] accomplish with their research. The crux of their approach is their introduction of a novel memory manipulation exploit that allows the execution of arbitrary code, something that was not previously possible. From there they can forcibly dilute deep learning inferences with trash output to the point that it unusable.

## Physical Side Channel vulnerabilities

The first physical SCA we will discuss was proposed by Gonzalez-Gomez et al. [12]. It establishes a side channel through the temperature of the GPU. By putting the GPU under load at strategic times, the transmitter can effectively control the operating temperature of the GPU. Encoding 1 to a high temperature and 0 to a low temperature, the attack was able to achieve a bitstream of 8.75 bits per second with less than a 2% error rate. The main hurdles of an attack of this nature lie in the fact that GPUs typically have thousands of cores, with only one thermal sensor for the entire device. This means that for a detectable temperature rise to occur, thousands of computations must happen simultaneously. In addition, the bit rate of a thermal SCA will always be astronomically low, because time must be provided to allow the GPU to properly cool between each transmission.

An interesting twist was found in a similar attack that took advantage of how small, usually portable, devices that contain ARM SoC (System-on-Chip) rely on Dynamic Voltage Frequency Scaling (DVFS) to account for and balance out power, frequency, and heat. Taneja et al. [20] ‘s research on the topic details the basis and execution of the attack: software on the device itself provides the measurements needed to make this physical SCA possible. By its nature the DVFS must rely on internal measurement sensors and this data can be retrieved and analyzed to identify the instructions being run as well as the data that it is being ran on. This is because these three elements are being balanced by the DVFS. By observing the changes in the measurements of at least two of these elements as they are throttled, the attacker can fingerprint match these values with websites to identify the victim’s browsing history.

This attack observes that SoCs can leak information through frequency, power, or temperature [20]. Their paper then outlines different methods of recovering a user’s web-based activities, such as visited sites, entered passwords, or search history. The attack abuses stacked SVG filters on certain objects in JavaScript, such as iframes and hyperlinks. Since certain colors take longer to apply than others, a receiver can essentially rebuild screenshots of a user’s activity once the filters are applied, as shown in figure 3.

A table with numbers and symbols

Description automatically generated

Fig. 3. Chrome pixel stealing results (from left to right): Original image, M1 MacBook Air, M2 MacBook Air, Pixel 6 Pro, OnePlus 10 Pro, Nvidia RTX3060, AMD RX 6600, Intel Iris Xe. The table summarizes recovery rates and accuracies, as shown in [20].

Next, we will consider one last attack that exploits magnetic emanations that the GPU gives off while under load [21]. Maia et al., were able to successfully infer and reconstruct a high-level overview of a neural network topology that the GPU was operating on, as shown in figure 4. These results were achieved with a simple three-dollar induction sensor.

A graph of a graph

Description automatically generated with medium confidence

Fig. 4. Leaked magnetic signal. (left) an induction sensor captures a magnetic signal when a CNN is running on the GPU. Note the strong correlation between the signal and the network steps. Across two steps, the GPU must synchronize, resulting in a sharp drop of the signal level (highlighted by selected red circles). (right) We can accurately classify the network steps and reconstruct the topology, as indicated by the labels under the x-axis. Here we highlight the signal regions associated with convolutions (conv), batch-norm (BN), Relu non-linear activations (relu), max-pooling (MP), and adding steps together (add), as shown in [21]

While an explanation of a neural network is outside the scope of this paper, one can use common sense to understand that being able to retroactively recover a neural network’s topology for 3 dollars can be of incredible value, and companies that rely on neural networks for income would want to take steps to protect their revenue source.

Now we will discuss a multi-layered attack that combines multiple attacks to retrieve a 16-bit secret key in a GPU-based AES implementation done by Gao et al. [22]. They propose an attack where-in the hardware is manipulated by cache-collision attacks. The idea is that there is a significant enough difference in power consumption when there is a cache collision and when there isn’t, and this directly correlates with the level of electrical emanation which can be recorded. Through multiple traces of electro-magnetic leakages, they were able to successfully retrieve the 16-bit secret key. While this is a very simplified explanation of their attack, it shows how novel attacks can be achieved through a combination of known techniques. Specifically, cache-collision attacks are being used to instigate indirect measurement of EM leakage via power consumption fluctuations. This Differential Electro-Magnetic Analysis (DEMA) is further bolstered by their use of a key enumeration algorithm. In this case the required 5000 EM traces can be reduced to only 600 EM traces.

The importance of ensuring the reliability of Deep Neural Networks (DNNs) in safety-critical systems, particularly when addressing the impact of soft errors, cannot be understated. It's important to identify the vulnerable instructions in DNN models to avoid the performance degradation associated with full instruction duplication. What should be used to pinpoint an issue like this would be the Instruction Vulnerability Factor to see the most vulnerable instructions in models. Certain research demonstrates the precision of determining vulnerable instructions and highlights that are in certain models is more sensitive to changes caused by soft errors compared to others [23]. The need for high reliability in DNNs is critical especially now in healthcare applications, where even minor errors can have severe consequences. As DNN models often rely on GPUs for computation, the impact of soft errors is a concern. Through the application of mitigation strategy, it has been shown through this strategy that it reduces the vulnerability of instructions from 4.42% to 0.14% of injected faults [24]. What can be brought into discussion is machine learning-based prediction frameworks for soft errors [25].

In addition, when considering Dynamic Voltage and Frequency Scaling (DVFS) mechanisms, the advantages they present must be compared with danger of the software-visible hybrid side channels attacks they allow. The challenges these hybrid side channels can have and the potential impact on security can be big [26].

As presented in above examples of GPUs such as NVIDIA and AMD, there reveals to be a novel electromagnetic side-channel vulnerability. Some practical attack scenarios, such as website fingering printing and keystroke timing inference attacks exploit this vulnerability. This shows the root cause of the widely used DVFS feature in GPUs. But disabling DVFS alone proves ineffective as it introduces another exploitable electromagnetic side-channel vulnerability [26].

What also can be looked at is the pretrained models in transfer learning, particularly focusing on the fine-tuning process and presenting the security vulnerabilities in transfer-earned models and introducing a novel two-level model extraction attack. Part of this attack involves identifying the pre-trained model of a transfer-model through model fingerprinting off-the-shelf GPUs [27].

What has also been examined is the comprehensive review of side-channel attacks specific to digital agriculture. Securing the digital agriculture should be in the discussion of addressing these challenges in the future[28].

There has been shown to be vulnerabilities of GPU- based AES implementation to cache-collision attacks, specifically utilizing electro-magnetic leakages. This is found in an efficient leakage model based on simultaneous cache-collision in multi-thread scenarios and employs a Differential Electro-Magnetic Analysis for key recovery. It was found that with as few as 5,000 EM traces, the 16-byte secret key of GPU-based AES can be successfully recovered. Additionally, when assisted by an appropriate key enumeration algorithm, a mere 600 EM traces are sufficient for key recovery [22].

There has also been found a side-channel attack called GLINT (Geo-location Inference Attack) focused on the WebGL framework. It leverages a lightweight browser extension to measure the time taken for rendering frames on popular map websites like Google maps. The attack measures time series data related to rendering frames and utilizes an online segmentation algorithm for streaming data, coupled with dynamic time warping and k-nearest neighbors algorithms to infer geo-location privacy, such as specific location searches by users [29].

## Micro-architectural vulnerabilities

The first micro-architectural attack we will look at, proposed by Almusaddar and Naghibijouybari, notes that in systems where the CPU and an integrated GPU, or iGPU are on the same chip (called a system on a chip, or SOC), CPU memory read requests can be predictably stalled with parallel iGPU memory write requests [30]. This attack has the transmitter encode 1s by writing to several different cache lines to quickly fill and drain the write buffer, stalling the CPU. Note that because the requests are to different cache lines, the coalescing effect does not occur. To encode a 0, the transmitter sends the same number of writes, but to the same cache line to avoid filling the write buffer. This creates a detectable side channel which can leak information at a rate of 1.65 kbps @ 0.49% error or 4.41kbps @ 4.32% error.

The second micro-architectural attack is named Trident [31]. Trident was developed because previous SIMT leakage based SCAs largely rely on a positive correlation that simply does not exist on modern hardware. Trident, however, has 3 parts to its attack.

1. Recover the first 4 bytes of an AES key through a negative correlation analysis.
2. Exploit the timing difference between L1 and L2 cache access for a cache collision-based attack.
3. Utilize a chosen plaintext attack to control the number of memory accesses.

It is worth noting that the first part of the attack is independent, while the third part enables the second. This attack allows for the full recovery of an AES key on modern GPUs.

Timing-based SCAs are also prevalent on mobile platforms as well. Karimi et al., outline a sample timing based attack on a Qualcomm Snapdragon, but note that the attack is generalizable to similar mobile platforms [8]. After identifying a correlation between memory load requests and execution time, Karimi et al. were able to launch a test kernel a million times and collect a million timing samples. Using these samples, a correlation analysis allows for full recovery of an AES key. While this attack is very similar to part two of the Trident attack described above, the implications of such an attack on a mobile platform can be dangerous. Moreso than on personal computers and laptops, mobile devices typically contain an abundance of personally identifying information (PII) and personal health information (PHI), which is critical to keep secure.

On the topic of mobile attacks, another attack, as described by Yang et al, outlines a how a trojan application, with only access to GPU performance counters, can effectively become a keylogger by measuring the amount of GPU overdraw that occurs on certain areas of a touchscreen [32]. By mapping these overdrawn areas to a digital keyboard, one can reconstruct a victim’s inputs with over 80% accuracy, as demonstrated in figure 5. This can be thought of similarly to the SVG filters from above, where multiple effects stacked on the same screen location results in a vector of information loss.

A screenshot of a phone

Description automatically generated

Fig. 5. GPU overdraw on popups of key presses in Android. Blue: 1x overdraw; green: 2x overdraw; pink: 3x overdraw; red: 4x overdraw, as seen in [32]

The mitigations for this attack are quite simple and non-transferrable, so they will be covered here briefly. Proposed defenses include disabling popups of key presses, enabling malware detection, and restricting access to GPU performance counter information [32].

In a fifth micro-architecture-based attack, proposed by Liu et al., attackers can infer what victims are doing across virtual machines (VMs) that share the same physical GPU [33]. While these inferences are relatively vague (like being able to differentiate idling from streaming 4k video to deep learning algorithms), This attack relies on the fact that, while separate VMs have their own dedicated virtual GPU, or vGPU, their performance is ultimately bottlenecked by the limitations of the hypervisor’s GPU. By running a simple read-compute-store operation repeatedly and measuring the execution time, an attacker can accurately infer what other types of loads are on the physical GPU, inferring a victim’s activities. This becomes an issue especially when considering the rising prevalence of cloud computing, where separate user’s workloads may be offloaded onto the same remote GPU. On the off chance that a malicious actor is listening in on sensitive or even classified information, even the insight that someone is doing something like deep learning can be critically important.

A prominent threat in computer security is memory corruption attacks. With contemporary GPUs supporting unified address space, vulnerabilities like buffer overflow and use-after-free can be exploited, enabling attackers to tamper with CPU data or even take care control of the control flow [22].

There can be multiple challenges as well when integrating complex, parallel and computationally demanding software functions on GPU devices for the development of high-performance safety-critical systems, particularly in autonomous driving applications. The most important thing to have involved in eradicating these issues certifying and taking note of the challenges that arising such as random hardware failures, systematic failures and issues related to the independence of execution on share GPU hardware resources [34].

There is a methodology proposed to identify architecturally vulnerable sites within GPU modules, focusing on locations that, if corrupted, would significantly impact correct instruction execution. It involves the Register-Transfer Level fault injection experiments. This can be made more effective by also using selective hardening strategies, including Triple Modular Redundancy, Triple Modular Redundancy against Single Event Transients, and Dual Interlocked Storage Cells, to mitigate the impact of faults. It was shown that the proposed method can tolerate a substantial percentage of faults in different GPU components with reduced hardware overhead compared to traditional Triple Redundancy Modular. Also further adapted for evaluating permanent faults, identifying critical sites prone to propagate fault effects across the GPU. This in turn enhances the reliability of GPUs in safety-critical applications [35].

Based on API calling features, there is a risk of memory data residue in GPU codes. It was found through the use of pass module development capability provided by the LLVM compiler project to implement a prototype for detecting memory leaks in CUDA-based GPU codes [36]. Beyond using the API calling features there is also a risk of targeting the memory API and probing it. Naghibijouybari et al. developed an attack that spies on applications using the GPU through a hostile application [37]. By seeing how memory is allocated the attacker can make inferences on the victims browsing history.

# Defenses

Thermal channel attacks have a few ways in which they can be prevented. Some proposed defenses include having GPUs randomly solve garbage computations in an effort to add noise to the system, as well as changing the operating frequency of the GPU over regular intervals to make interpreting data much harder [12]. However, these countermeasures introduce significant speed reductions to systems, and must be implemented carefully to maximize GPU performance. It’s important to bring awareness of vulnerabilities to light and increase potential countermeasures [38]. Often, it is this introduction of overheads that may make users of the CUDA platform for GPUs and vendors wince at the idea of reducing performance when it is the high performance of a GPU that makes it use and implementation so valuable. Having to constantly initialize or run code specifically to erase may not seem like such an attractive idea, but memory vulnerabilities such as the one presented by Lee et al. could be easily avoided as such. Otherwise, researchers are left to ask for GPU vendors to reorient their efforts towards making this possible without a large impact on performance [18].

Pixel stealing attacks based on SVG filters are quite easy to defend against. Some proposed countermeasures include:

1. Running the system well under power or thermal budgets so that frequency is never throttled [20].
2. Preventing SVG filters from applying to iframes and hyperlinks [20]. This feature is already implemented in Firefox.

Timing based SCAs on SoCs have also had defenses proposed which would make it much harder to successfully pull off an attack. Among these include prioritizing new memory read requests in the iGPU, and partitioning channels to multiple memory controllers (MCs) to make it harder for the transmitter to send data [30]. In addition, continuously randomizing the width of the coalescing unit can make it harder to execute attacks based on timing differentials [7].

The developers behind the Trident attack also proposed TridentShield, a countermeasure to their timing-based correlation attack [31]. They note that randomly coalescing memory requests or introducing bucket-based coalescing techniques can work, but either have limitations or significant computing overhead. In contrast, TridentShield prevents countermeasures by making memory access times constant, instead of variable and exploitable.

Regarding cross VM attacks, Liu et al., recommend incorporating side-channel aware resource scheduling on all GPUs used in a hypervisor, as well as having the hypervisor monitor GPU activities on each vGPU [33].

Mitigation strategies can be put in place for GPUs. This becomes especially true when considering the susceptibility of other components within heterogeneous systems, including the interconnect and memory components. Considering the implementation of such strategies may provide insights into addressing security challenges in contemporary designs [39]. A lot can be said with heterogenous computing [40].

The concept of Kernel Vulnerability Factor and Layer Vulnerability Factor as metrics for assessing the likelihood of faults in specific software portions, such as kernels or layers in neural networks, affecting the computation. For Histogram of Oriented Gradients, a selective hardening technique is proposed with an 85% critical error detection rate and a performance overhead as low as 11.8%. For You Only Look Once, Layer Vulnerability Factor is found to be within the subject of architectural-level fault-injection, distinguishing between tolerable and critical errors and proposing a smart layer duplication method that detects over 90% of errors with an overhead below 60% [41].

What has been employed for a first line defense against timing and cache attacks is the constant-time implementation of RSA exponentiation for both CPU and GPU platforms. In this case the GPU implementation utilizes the Residue Number Systems representation for modular arithmetic. A fascinating performance analysis by the two prominent RNS modular reduction algorithms across various platforms. The software library outperforms in terms of RSA exponentiation computation timings with 1024-, 2048-, and 3072-bit private keys [42].

What also could be proposed in some instances is a taxonomy for attack and defense approaches, offering guidelines for selecting appropriate strategies based on specific goals and available resources. This will in turn identify what defenses may be less effective against current attack strategies, contributing to a holistic understanding of model stealing vulnerabilities, for example in Machine-Learning-as-a-Service [43].

There are some innovative methods that were brought about by malware writers to enhance the resilience of their malicious software against analysis and detection. Their focus is on leveraging the capabilities of GPUs to implement code armoring techniques, specifically brute force unpacking and runtime polymorphism. These techniques, based on computational power of modern GPUs, present significant challenges to existing malware detection and analysis systems, which are traditionally geared towards CPU code analysis. What could be anticipated moving forward is the potential use of upcoming GPU features for constructing even more resilient and evasive malware, along with discussing potential directions for defense against GPU-assisted malware. It has been shown to have the detection system of the presence of GPU-assisted malware [44].

What also needs to be addressed is the critical, architectural considerations for secure executions GPUs, in the context of security applications and libraries relying on the GPU’s massively parallel computations. Previous defenses have introduced significant performance overhead by randomizing or restricting memory coalescing, a crucial feature for GPU performance. The proposed GhostLeg defense selectively applies secure executions for load warps, minimizing the impact on performance induced by concealing memory coalescing behavior. Ghostleg pinpoints load warps vulnerable to security attacks based on the class of a source register, achieving secure executions with minimal performance overhead. What has been seen from this is that GhostLeg provides secure executions against correlation-based attacks, with GhostLeg-ND exhibiting a 54.7% higher performance compared to the state-of-the-art GPU defense solutions [29].

In some cases, security threats were posed by System-Level Cache in Advanced RISC Machines (ARM) processors. They can be exploited to create a cache occupancy channel, leading to website fingerprinting. The attack can be optimized based on various browsers based on the ARM cache design, reducing the attack duration while increasing accuracy. The need for defenses against such security threats is vital [45].

In GPU-accelerated computing, there are multiple dimensions of CUDA security. A scrutiny of the GPU stack reveals that stack overflow in CUDA can impact the execution of other threads by manipulating various memory spaces. The identification of vulnerabilities related to integer overflow and function pointer overflow instructions on GPUs is important. Attacks against format strings and exception handlers seem infeasible due to design choices in CUDA runtime and programming language features [46].

There are multiple cyber-attacks and integrity tests on GPUs, but proposing a framework for inspecting code executed on GPUs to detect and respond to security threats helps focus on identify process that require GPU access, intercepting them and scanning for any suspicious activities[47]. Timing attacks can be mitigated by building a timing model to capture the similar characteristics of an RSA public-key implemented on a GPU, which is presented with error detection and correction mechanism [48].

The GPU con-encryption mechanism can also be used to alleviate timing attacks to help secure platforms for autonomous systems and greatly enhance system security [49].

In some cases, reverse engineering can prove to be a security problem. Reverse engineering attacks can be done with a unique family of malware that escapes detection using AMD and NVIDIA’s architecture decisions as a hiding place. What can be used is Open CL to remove the malware from said GPU and avoid threats in the future and this does so without any drawbacks [50].

# Conclusion

Since the introduction of GPUs as a processor optimized for complex shader and vertex computations, we have seen its rapid evolution and improvement over the years. Its capability to handle immensely parallel problems at ever increasing rates has enabled it to renew interest in neural networks and push their possibilities forward. As this trend continues and GPUs find utilization in self-driving cars, medical diagnosis and prognosis, and A.I., they will become an ever-more attractive target and vectors for attackers looking to exploit their vulnerabilities will continue to be discovered and abused. It is imperative that cyber security researchers and other fields do not neglect to preempt these attacks by building up knowledge on how these attacks may be perpetuated and to take the necessary precautions. It is our hope that this survey of GPU vulnerabilities and attacks will provide enough foundation to promote further research on GPU security or prompt those who utilize GPUs to understand how they could be putting their data and their systems at risk, and that there must be consideration in the implementation of GPU backed solutions.

# References

[1] T. Kim and Y. Shin, “GPU Side-Channel Attacks are Everywhere: A Survey,” in *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, Seoul, Korea (South): IEEE, Nov. 2020, pp. 1–4. doi: 10.1109/ICCE-Asia49877.2020.9276805.

[2] W. J. Dally, S. W. Keckler, and D. B. Kirk, “Evolution of the Graphics Processing Unit (GPU),” *IEEE Micro*, vol. 41, no. 6, pp. 42–51, Nov. 2021, doi: 10.1109/MM.2021.3113475.

[3] Z. H. Jiang, Y. Fei, and D. Kaeli, “A complete key recovery timing attack on a GPU,” in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Barcelona, Spain: IEEE, Mar. 2016, pp. 394–405. doi: 10.1109/HPCA.2016.7446081.

[4] S. A. Jeffers, R. O. Baldwin, and B. E. Mullins, “Accelerating Missile Threat Simulations Using Personal Computer Graphics Cards,” *SIMULATION*, vol. 82, no. 8, pp. 549–558, Aug. 2006, doi: 10.1177/0037549706072047.

[5] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, “Beyond the CPU: Side–Channel Attacks on GPUs,” *IEEE Des. Test*, vol. 38, no. 3, pp. 15–21, Jun. 2021, doi: 10.1109/MDAT.2021.3063359.

[6] X. Wang and W. Zhang, “Cracking Randomized Coalescing Techniques with An Efficient Profiling-Based Side-Channel Attack to GPU,” in *Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy*, Phoenix AZ USA: ACM, Jun. 2019, pp. 1–8. doi: 10.1145/3337167.3337169.

[7] E. Karimi, Y. Fei, and D. Kaeli, “Hardware/Software Obfuscation against Timing Side-channel Attack on a GPU,” in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, San Jose, CA, USA: IEEE, Dec. 2020, pp. 122–131. doi: 10.1109/HOST45689.2020.9300259.

[8] E. Karimi, Z. H. Jiang, Y. Fei, and D. Kaeli, “A Timing Side-Channel Attack on a Mobile GPU,” in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, Orlando, FL, USA: IEEE, Oct. 2018, pp. 67–74. doi: 10.1109/ICCD.2018.00020.

[9] D. G. Mahmoud, V. Lenders, and M. Stojilović, “Electrical-Level Attacks on CPUs, FPGAs, and GPUs: Survey and Implications in the Heterogeneous Era,” *ACM Comput. Surv.*, vol. 55, no. 3, pp. 1–40, Mar. 2023, doi: 10.1145/3498337.

[10] C. Luo *et al.*, “Power Analysis Attack of an AES GPU Implementation,” *J. Hardw. Syst. Secur.*, vol. 2, no. 1, pp. 69–82, Mar. 2018, doi: 10.1007/s41635-018-0032-7.

[11] A. A. Ahmed and M. K. Hasan, “Design and Implementation of Side Channel Attack Based on Deep Learning LSTM,” in *2023 IEEE Region 10 Symposium (TENSYMP)*, Canberra, Australia: IEEE, Sep. 2023, pp. 1–6. doi: 10.1109/TENSYMP55890.2023.10223652.

[12] J. González-Gómez, K. Cordero-Zuñiga, L. Bauer, and J. Henkel, “The First Concept and Real-world Deployment of a GPU-based Thermal Covert Channel: Attack and Countermeasures,” in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium: IEEE, Apr. 2023, pp. 1–6. doi: 10.23919/DATE56975.2023.10137090.

[13] X. Wang and W. Zhang, “pacSCA: A Profiling-Assisted Correlation-based Side-Channel Attack on GPUs,” in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, Hartford, CT, USA: IEEE, Oct. 2020, pp. 525–528. doi: 10.1109/ICCD50377.2020.00094.

[14] Y. Gao, Y. Zhou, and W. Cheng, “Efficient electro-magnetic analysis of a GPU bitsliced AES implementation,” *Cybersecurity*, vol. 3, no. 1, p. 3, Feb. 2020, doi: 10.1186/s42400-020-0045-8.

[15] M. Sabbagh, Y. Fei, and D. Kaeli, “A Novel GPU Overdrive Fault Attack,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA: IEEE, Jul. 2020, pp. 1–6. doi: 10.1109/DAC18072.2020.9218690.

[16] C.-S. Chen, S.-Y. Chiu, and S.-Y. Li, “AES Encryption Method Based on Chaotic Cryptography,” in *2023 International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan)*, PingTung, Taiwan: IEEE, Jul. 2023, pp. 293–294. doi: 10.1109/ICCE-Taiwan58799.2023.10226728.

[17] A. Fanfakh, H. Noura, and R. Couturier, “Simultaneous encryption and authentication of messages over GPUs,” *Multimed. Tools Appl.*, May 2023, doi: 10.1007/s11042-023-15451-5.

[18] S. Lee, Y. Kim, J. Kim, and J. Kim, “Stealing Webpages Rendered on Your Browser by Exploiting GPU Vulnerabilities,” in *2014 IEEE Symposium on Security and Privacy*, San Jose, CA: IEEE, May 2014, pp. 19–33. doi: 10.1109/SP.2014.9.

[19] S.-O. Park, O. Kwon, Y. Kim, S. K. Cha, and H. Yoon, “Mind control attack: Undermining deep learning with GPU memory exploitation,” *Comput. Secur.*, vol. 102, p. 102115, Mar. 2021, doi: 10.1016/j.cose.2020.102115.

[20] H. Taneja, J. Kim, J. J. Xu, S. van Schaik, D. Genkin, and Y. Yarom, “Hot Pixels: Frequency, Power, and Temperature Attacks on GPUs and ARM SoCs.” arXiv, May 22, 2023. Accessed: Nov. 06, 2023. [Online]. Available: http://arxiv.org/abs/2305.12784

[21] H. T. Maia, C. Xiao, D. Li, E. Grinspun, and C. Zheng, “Can one hear the shape of a neural network?: Snooping the GPU via Magnetic Side Channel.” arXiv, Sep. 15, 2021. Accessed: Nov. 06, 2023. [Online]. Available: http://arxiv.org/abs/2109.07395

[22] Y. Gao, W. Cheng, H. Zhang, and Y. Zhou, “Cache-Collision Attacks on GPU-Based AES Implementation with Electro-Magnetic Leakages,” in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, New York, NY, USA: IEEE, Aug. 2018, pp. 300–306. doi: 10.1109/TrustCom/BigDataSE.2018.00053.

[23] I. Younis, J. Liu, X. Yang, H. Sha, P. Li, and H. Wang, “Analyzing the Impact of Soft Errors in Deep Neural Networks on GPUsfrom Instruction Level,” *WSEAS Trans. Syst. CONTROL*, vol. 15, pp. 699–708, Dec. 2020, doi: 10.37394/23203.2020.15.70.

[24] K. Adam, I. I. Mohd, and Y. Ibrahim, “Analyzing the instructions vulnerability of dense convolutional network on GPUS,” *Int. J. Electr. Comput. Eng. IJECE*, vol. 11, no. 5, Art. no. 5, Oct. 2021, doi: 10.11591/ijece.v11i5.pp4481-4488.

[25] B. Topçu and I. Öz, “Soft error vulnerability prediction of GPGPU applications,” *J. Supercomput.*, vol. 79, no. 6, pp. 6965–6990, Apr. 2023, doi: 10.1007/s11227-022-04933-2.

[26] Z. Zhan, Z. Zhang, S. Liang, F. Yao, and X. Koutsoukos, “Graphics Peeping Unit: Exploiting EM Side-Channel Information of GPUs to Eavesdrop on Your Neighbors,” in *2022 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA: IEEE, May 2022, pp. 1440–1457. doi: 10.1109/SP46214.2022.9833773.

[27] M. A. Rafi, Y. Feng, F. Yao, M. Tang, and H. Jeon, “Decepticon: Attacking Secrets of Transformers,” in *2023 IEEE International Symposium on Workload Characterization (IISWC)*, Oct. 2023, pp. 128–139. doi: 10.1109/IISWC59245.2023.00028.

[28] A. N. Alahmadi, S. U. Rehman, H. S. Alhazmi, D. G. Glynn, H. Shoaib, and P. Solé, “Cyber-Security Threats and Side-Channel Attacks for Digital Agriculture,” *Sensors*, vol. 22, no. 9, p. 3520, May 2022, doi: 10.3390/s22093520.

[29] W. Mai and Y. Xiao, “A novel GPU based Geo-Location Inference Attack on WebGL framework,” *High-Confid. Comput.*, vol. 3, no. 4, p. 100135, Dec. 2023, doi: 10.1016/j.hcc.2023.100135.

[30] G. Almusaddar and H. Naghibijouybari, “Exploiting Parallel Memory Write Requests for Covert Channel Attacks in Integrated CPU-GPU Systems.” arXiv, Jul. 30, 2023. Accessed: Nov. 06, 2023. [Online]. Available: http://arxiv.org/abs/2307.16123

[31] J. Ahn *et al.*, “Trident: A Hybrid Correlation-Collision GPU Cache Timing Attack for AES Key Recovery,” in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Seoul, Korea (South): IEEE, Feb. 2021, pp. 332–344. doi: 10.1109/HPCA51647.2021.00036.

[32] B. Yang, R. Chen, K. Huang, J. Yang, and W. Gao, “Eavesdropping user credentials via GPU side channels on smartphones,” in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Lausanne Switzerland: ACM, Feb. 2022, pp. 285–299. doi: 10.1145/3503222.3507757.

[33] S. Liu, Y. Wei, J. Chi, F. H. Shezan, and Y. Tian, “Side Channel Attacks in Computation Offloading Systems with GPU Virtualization,” in *2019 IEEE Security and Privacy Workshops (SPW)*, San Francisco, CA, USA: IEEE, May 2019, pp. 156–161. doi: 10.1109/SPW.2019.00037.

[34] J. Perez-Cerrolaza, J. Abella, L. Kosmidis, A. J. Calderon, F. Cazorla, and J. L. Flores, “GPU Devices for Safety-Critical Systems: A Survey,” *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–37, Jul. 2023, doi: 10.1145/3549526.

[35] J. E. Rodriguez Condia, P. Rech, F. F. D. Santos, L. Carro, and M. S. Reorda, “An Effective Method to Identify Microarchitectural Vulnerabilities in GPUs,” *IEEE Trans. Device Mater. Reliab.*, vol. 22, no. 2, pp. 129–141, Jun. 2022, doi: 10.1109/TDMR.2022.3166260.

[36] J. Yang, R. He, and B. Cui, “A GPU memory leakage code defect detection method based on the API calling feature,” in *The 2nd International Conference on Computing and Data Science*, Stanford CA USA: ACM, Jan. 2021, pp. 1–13. doi: 10.1145/3448734.3450774.

[37] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, “Rendered Insecure: GPU Side Channel Attacks are Practical,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Toronto Canada: ACM, Oct. 2018, pp. 2139–2153. doi: 10.1145/3243734.3243831.

[38] S. Mittal, S. B. Abhinaya, M. Reddy, and I. Ali, “A Survey of Techniques for Improving Security of GPUs,” *J. Hardw. Syst. Secur.*, vol. 2, no. 3, pp. 266–285, Sep. 2018, doi: 10.1007/s41635-018-0039-0.

[39] H. Naghibijouybari, E. M. Koruyeh, and N. Abu-Ghazaleh, “Microarchitectural Attacks in Heterogeneous Systems: A Survey,” *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–40, Jul. 2023, doi: 10.1145/3544102.

[40] J. J. Garcia-Hernandez, M. Morales-Sandoval, and E. Elizondo-Rodríguez, “A Flexible and General-Purpose Platform for Heterogeneous Computing,” *Computation*, vol. 11, no. 5, Art. no. 5, May 2023, doi: 10.3390/computation11050097.

[41] F. Fernandes dos Santos, L. Carro, and P. Rech, “Kernel and layer vulnerability factor to evaluate object detection reliability in GPUs,” *IET Comput. Digit. Tech.*, vol. 13, no. 3, pp. 178–186, 2019, doi: 10.1049/iet-cdt.2018.5026.

[42] E. Ochoa-Jimenez, L. Rivera-Zamarripa, N. Cruz-Cortes, and F. Rodriguez-Henriquez, “Implementation of RSA Signatures on GPU and CPU Architectures,” *IEEE Access*, vol. 8, pp. 9928–9941, 2020, doi: 10.1109/ACCESS.2019.2963826.

[43] D. Oliynyk, R. Mayer, and A. Rauber, “I Know What You Trained Last Summer: A Survey on Stealing Machine Learning Models and Defences,” *ACM Comput. Surv.*, vol. 55, no. 14s, pp. 1–41, Dec. 2023, doi: 10.1145/3595292.

[44] D. Balzarotti, R. Di Pietro, and A. Villani, “The impact of GPU-assisted malware on memory forensics: A case study,” *Digit. Investig.*, vol. 14, pp. S16–S24, Aug. 2015, doi: 10.1016/j.diin.2015.05.010.

[45] P. Cronin, X. Gao, H. Wang, and C. Cotton, “An Exploration of ARM System-Level Cache and GPU Side Channels,” in *Annual Computer Security Applications Conference*, Virtual Event USA: ACM, Dec. 2021, pp. 784–795. doi: 10.1145/3485832.3485902.

[46] B. Di, J. Sun, and H. Chen, “A Study of Overflow Vulnerabilities on GPUs,” in *Network and Parallel Computing*, G. R. Gao, D. Qian, X. Gao, B. Chapman, and W. Chen, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 103–115. doi: 10.1007/978-3-319-47099-3\_9.

[47] T.-P. Apostol, R. Velea, and R. Deaconescu, “A Framework for Analyzing GPU-Executed Malware,” in *2021 23rd International Conference on Control Systems and Computer Science (CSCS)*, Bucharest, Romania: IEEE, May 2021, pp. 165–171. doi: 10.1109/CSCS52396.2021.00034.

[48] C. Luo, Y. Fei, and D. Kaeli, “Side-channel Timing Attack of RSA on a GPU,” *ACM Trans. Archit. Code Optim.*, vol. 16, no. 3, pp. 1–18, Sep. 2019, doi: 10.1145/3341729.

[49] Z. Wang, R. Wang, Z. Jiang, X. Tang, S. Yin, and Y. Hu, “Towards a Secure Integrated Heterogeneous Platform via Cooperative CPU/GPU Encryption,” in *2021 IEEE 30th Asian Test Symposium (ATS)*, Nov. 2021, pp. 115–120. doi: 10.1109/ATS52891.2021.00032.

[50] Y. Albabtain and B. Yang, “The Process of Reverse Engineering GPU Malware and Provide Protection to GPUS,” in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, New York, NY, USA: IEEE, Aug. 2018, pp. 1669–1673. doi: 10.1109/TrustCom/BigDataSE.2018.00248.