# NIST CSF-2.0 Compliant GPU Shader Execution

**Nelson Lungu**
Electrical and Electronical Engineering, University of Zambia, Lusaka, Zambia
lungunc@gmail.com

**Ahmad Abdulqadir Al Rababah**
Faculty of Computing and Information Technology, King Abdulaziz University, Rabigh, Saudi Arabia
aaahmad13@kau.edu.sa

**Bibhuti Bhusan Dash**
School of Computer Applications, KIIT Deemed to be University, Bhubaneswar, India
bibhuti.dash@gmail.com

**Asif Hassan Syed**
Faculty of Computing and Information Technology, King Abdulaziz University, Rabigh, Saudi Arabia
shassan1@kau.edu.sa

**Lalbihari Barik**
Faculty of Computing and Information Technology, King Abdulaziz University, Rabigh, Saudi Arabia
lalbihari@gmail.com

**Suchismita Rout**
School of Computer Engineering, KIIT Deemed to be University, Bhubaneswar, India
suchismita.rout28@gmail.com

**Simon Tembo**
Electrical and Electronical Engineering,University of Zambia, Lusaka, Zambia
Simontembo6@gmail.com

**Charles Lubobya**
Electrical and Electronical Engineering, University of Zambia, Lusaka, Zambia
cslubobya@unza.zm

**Sudhansu Shekhar Patra**
School of Computer Applications, KIIT Deemed to be University, Bhubaneswar, India
sudhanshupatra@gmail.com (corresponding author)

*Received: 26 March 2024 | Revised: 11 April 2024, 1 May 2024, and 14 May 2024 | Accepted: 15 May 2024*

*Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | DOI: https://doi.org/10.48084/etasr.7351*

## ABSTRACT

**This article introduces a mechanism for ensuring trusted GPU shader execution that adheres to the NIST Cybersecurity Framework (CSF) 2.0 standard. The CSF is a set of best practices for reducing cybersecurity risks. We focus on the CSF's identification, protection, detection, and response mechanisms for GPU-specific security. To this end, we exploit recent advancements in side-channel analysis and hardware-assisted security for the real-time and introspective monitoring of shader execution. We prototype our solution and measure its performance across different GPU platforms. The evaluation results demonstrate the effectiveness of the proposed mechanism in detecting anomalous shader behaviors that only incur modest overhead at runtime. Integrating the CSF 2.0 principles into the proposed GPU shader pipeline leads to an organizational recipe for securing heterogeneous computing resources.**

***Keywords-GPU security; shader execution attacks/defenses; anomaly detection techniques; NIST CSF mapping; real-time protection mechanisms***

## I. INTRODUCTION

Graphics Processing Units (GPUs) have emerged as critical accelerators for high-performance computing applications ranging from scientific computing to artificial intelligence [1, 2]. However, the complexity of modern GPU architectures also introduces potential security vulnerabilities that malicious actors could exploit [3-10]. Recent research has demonstrated side-channel attacks that leak sensitive data from GPU shader execution [10-13]. As GPU adoption grows, ensuring the security and integrity of shader pipelines is paramount. The National Institute of Standards and Technology (NIST) Cybersecurity Framework (CSF) provides guidelines to reduce cybersecurity risks that are applicable across sectors and organisation sizes [14]. Created for critical infrastructure entities, CSF 2.0 expands the scope to all organisations with a particular emphasis on risk management [15]. This work examines the application of CSF 2.0 principles to secure GPU shader execution. By integrating real-time monitoring and protection mechanisms into shader pipelines, we enable NIST CSF compliance for heterogeneous computing platforms.

The main contributions of the work at hand to the GPU security domain are summarized as:

- Potential security risks and attack vectors within modern GPU shader execution are identified.
- A methodology for applying NIST CSF 2.0 guidelines to GPUs is proposed, focusing on identifying, protecting, detecting, and responding to functions.
- A unique solution for shader anomaly detection and real-time shader monitoring is proposed using performance counters and side-channel defences.
- The prototype is evaluated on NVIDIA and AMD GPUs and demonstrate the detection of abnormal shader behaviour with minimal performance overhead.

## II. BACKGROUND

### *A. NIST Cybersecurity Framework 2.0*

The National Institute of Standards and Technology (NIST) Cybersecurity Framework (CSF) provides guidelines for managing cybersecurity risks [14]. The framework identifies core functions, categories, and subcategories corresponding to key security posture activities. As shown in Figure 1, the five core functions are Identity, Protect, Detect, Respond, and Recover. The functions provide a high-level view of the lifecycle of cybersecurity risk management. Categories within each function outline particular outcomes, while subcategories detail specific security controls and processes.

The CSF also defines profiles that represent how an organisation views cybersecurity risk and the processes in place to manage it. Profiles can be used to describe the current state or desired target state of cybersecurity activities. By comparing the two profiles, plans can be developed to reduce gaps and strengthen defences. The CSF 2.0 enhances guidance for risk management, authentication, supply chains, and vulnerability disclosures [15]. It also emphasises diversity, equity, inclusion and accessibility.

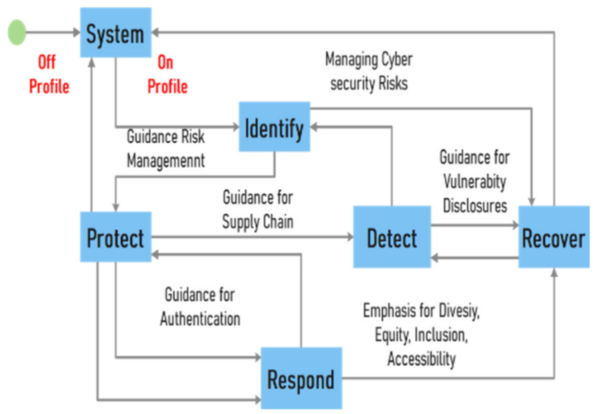

System
Off Profile
On Profile
Managing Cyber security Risks
Identify
Guidance Risk Managemennt
Guidance for Vulnerabity Disclosures
Guidance for Supply Chain
Protect
Detect
Recover
Guidance for Authentication
Emphasis for Divesiy, Equity, Inclusion, Accessibility
Respond

Fig. 1. NIST CSF 2.0 overview.

### *B. GPU Architecture*

GPUs comprise multiple Streaming Multiprocessors (SMs), each containing execution units and on-chip memory (Figure 2) [16]. CUDA and OpenCL are standard programming models for GPUs. Programs consist of host code executing on the CPU and parallel kernel functions running on the GPU. Kernels are organized into a grid of thread blocks that execute on SMs. The graphics pipeline involves several stages, including vertex processing, tessellation, geometry shading, rasterisation, and fragment shading [17]. Programmable shaders execute specialized kernels for each stage. For example, vertex shaders operate on vertex attributes, while fragment shaders compute pixel colours. Shaders are prone to side-channel attacks as they can contain secret-dependent memory accesses or execution paths [18].

The NIST CSF provided guidelines and best practices for managing cybersecurity risks applicable across sectors and organisation sizes. Initially released in 2014, the widespread adoption of the CSF made an update necessary to expand its scope and evolve with the changing threat landscape. Thus, NIST introduced CSF 2.0 in April 2022, bringing refinements based on years of industry feedback. Emphasising best practices for cybersecurity guidelines. CSF version 2.0 prioritizes secure software development, flexibility, and risk management in the cybersecurity domain.

This research applied the comprehensive CSF 2.0 guidelines and methodology to enhance the security of GPU shader execution by leveraging recent advances in hardware security and side-channel defences [12]. We architect a CSF-aligned framework tailored to GPUs for security and side-channel defences [12]. Continuous monitoring, anomaly detection, and runtime mitigations on shader pipelines provide a layered defence that enables rapid detection and response to compromised shaders [13]. We utilize performance counters and side-channel prevention techniques to implement the core

CSF functions of identifying, protecting, detecting, and responding [14, 21]. The goal is to move GPU security closer to cybersecurity best practices, reducing risks while maintaining the performance of heterogeneous computing infrastructure. With CSF 2.0 and secure GPU computing representing cutting-edge, high-impact research areas, this work synergistically applies the former to advance the state-of-the-art in the latter. This work focused on securing shader execution through continuous monitoring and runtime protections, as described in the following sections.

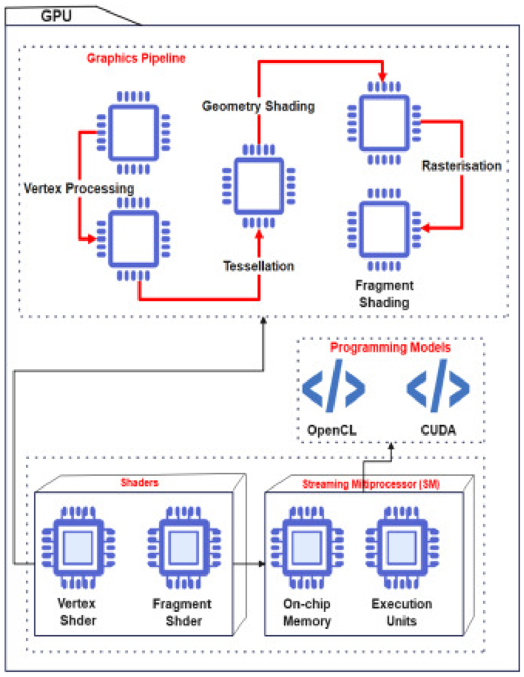


Fig. 2. GPU architecture.

## III. THREAT MODEL AND ATTACK VECTORS

We considered a threat model in which the attacker accesses a multi-tenant GPU server running malicious shader programs. The attacker aims to infer secrets or induce errors during shader execution through side-channel or fault attacks. The relevant attack vectors are summarize in Figures 3 – 8.

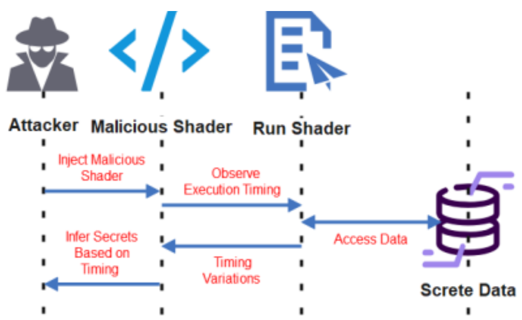


Fig. 3. Timing attack vector.

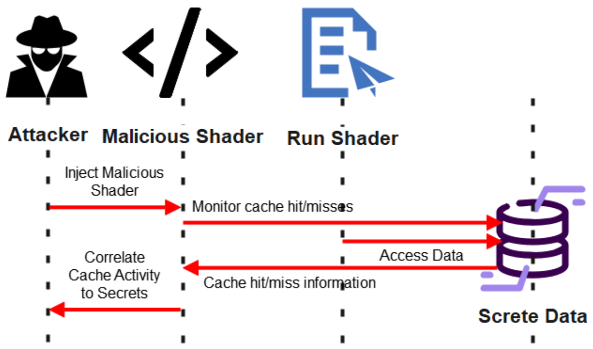
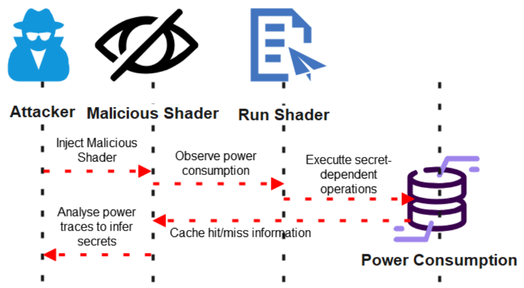
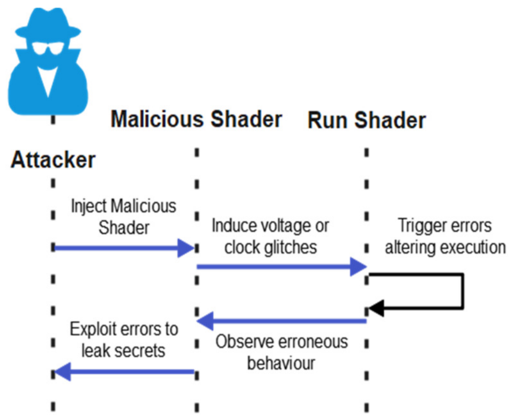


Fig. 4. Cache attack vector.


Attacker
Malicious Shader
Run Shader
Inject Malicious Shader
Observe power consumption
Executte secret-dependent operations
Analyse power traces to infer secrets
Cache hit/miss information
Power Consumption
Fig. 5. Power analysis attack vector.


Malicious Shader
Run Shader
Attacker
Inject Malicious Shader
Induce voltage or clock glitches
Trigger errors altering execution
Exploit errors to leak secrets
Observe erroneous behaviour
Fig. 6. Fault attacks.

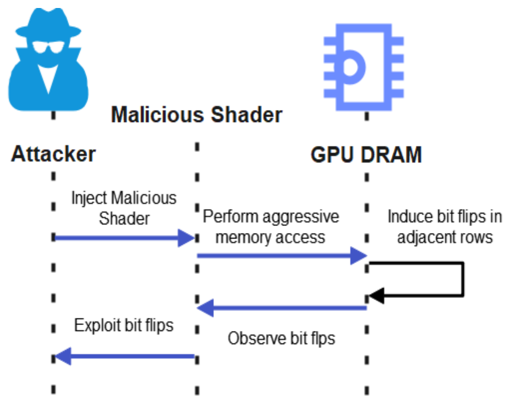
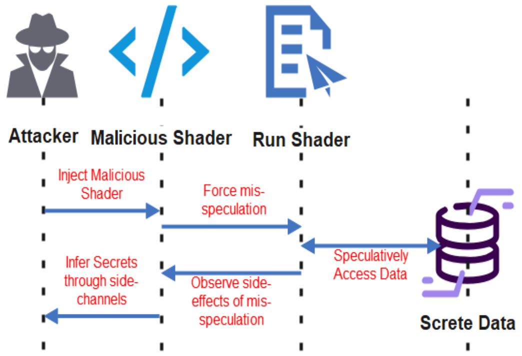


Fig. 7. Rowhammer attacks.



Attacker
Malicious Shader
Run Shader
Inject Malicious Shader
Force mis-speculation
Speculatively Access Data
Infer Secrets through side-channels
Observe side-effects of mis-speculation
Screte Data

Fig. 8. Speculative execution attacks.

This threat landscape motivated a holistic strategy integrating runtime monitoring, anomaly detection, and side-channel defences. By leveraging hardware performance counter techniques, shader execution is continuously validated to respond rapidly to anomalies indicative of an attack.

## IV. RELATED WORK

Table I summarises key recent research thrusts related to the security of GPU shader execution. Side-channel attack methods highlight the need for robust protections. Various shader execution hardening techniques have been proposed, while anomaly detection leverages GPU hardware performance counters.

TABLE I. SUMMARY OF RELATED WORK

| Research area | Description | Ref. |
|---|---|---|
| Side-channel attacks on GPUs | Attacks use timing or cache side channels during shader execution to leak data. | [11-13] |
| Shader security enhancements | Techniques like input validation, control flow protections, shader encryption and obfuscation | [10, 18, 19] |
| Anomaly detection using performance counters | Detecting abnormal GPU execution by monitoring performance metrics | [1-8] |
| CSF-based security frameworks | Applying NIST CSF for cybersecurity risk management | Proposed methodology |

## V. METHODOLOGY

The proposed methodology aligns with the significant functions of the NIST CSF by incorporating core techniques for identification, protection, detection, and response tailored to securing GPU shader execution.

### A. *Identification*

We performed a detailed threat modelling analysis to identify key assets, vulnerabilities, and threat vectors related to the GPU shader execution environment. Assets include shader binaries, inputs, runtime execution state, and outputs. Vulnerabilities stem from the shader programming model, GPU architecture, and shared hardware resources. Specifically, we analyze timing channels arising from shader pipelines, cache side channels due to hardware reuse, speculative execution flaws, and memory corruption due to a lack of vulnerability checks. Threats include malicious co-resident shaders, unauthorised access on multi-tenant GPU servers, and credentialed insider attacks. This comprehensive analysis informs our risk assessment.

### B. *Protection*

Continuous runtime monitoring attacks were implement (Figure 9) along with mitigations during shader execution, as illustrated in Figure 11.

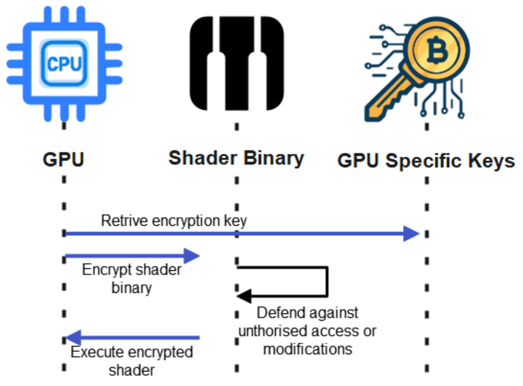


Fig. 9. Attack monitoring.

### C. *Detection*

We continuously monitor multiple hardware performance counters to profile the expected shader behavior and rapidly detect anomalies.

- Execution time:

$$\text{T\_shader} = \text{T\_end} - \text{T\_start} \qquad (1)$$

where T_shader is the shader execution time, T_end is the shader end time, and T_start is the shader start time. Here, we track the overall shader runtime to detect abnormal variations indicative of timing side channels.

- Cache hit rate:

$$\text{HitRate} = \frac{\text{CacheHits}}{\text{CacheHits+CacheMisses}} \qquad (2)$$

where HitRate is the ratio of cache hits to total cache accesses. Tracking cache hit rates can detect cache side-channel attacks. Here, we monitor cache access patterns to detect abnormal cache usage signatures of side-channel attacks.

- Power consumption:

$$P = V \text{ X } I \qquad (3)$$

where P is the power (W), V is the voltage (V), and I is the current (A). Changes in power profiles can indicate ongoing power analysis or fault attacks. We analyze power profiles to detect aberrations corresponding to fault injection or power side-channel attacks.

- Memory bandwidth utilization:

$$\text{BW\_util} = \frac{\text{Bytes\_accessed}}{\text{Time}} / \text{BW\_max} \qquad (4)$$

where BW_util is bandwidth utilization, Bytes_accessed represents the read/written bytes, Time is the shader execution time, and BW_max is the maximum bandwidth. High utilization may reveal rowhammer or cache attacks.

- Execution divergence:

$$\text{Divergence} = \frac{\text{Serial\_instr} - \text{Parallel\_instr}}{\text{Serial\_instr}} \quad (5)$$

where Serial_instr represents the instructions executed serially, Parallel_instr represents the instructions executed in parallel, and Divergence measures serialization. Divergence can identify code manipulation or replay. We measure divergence to detect abnormal serialization that can occur with malicious shader modification or replay.

By continuously tracking these performance counters and metrics during shader execution, we can profile the expected behavior and rapidly detect anomalies that may correspond to an attack. Thresholds based on allowlists differentiate normal and abnormal execution characteristics.

## *D. Response*

Upon detecting shader anomaly via the performance counters, we immediately:

1. Terminate the shader instance to stop the attack.
2. Log forensic data for post-mortem diagnosis of the root cause.
3. Adjust real-time protections and defences to address the detected attack vector.
4. Quick termination and adaptation limit the damage and continuously build attack intelligence to improve defences.

This in-depth methodology leverages performance counters and hardware-augmented monitoring to provide NIST CSF-aligned security for GPU shader workloads.

## VI. IMPLEMENTATION AND RESULTS

To evaluate the proposed methodology, we implemented a proof-of-concept prototype on an NVIDIA RTX 3090 GPU and an AMD Radeon RX 6900 XT GPU.

## *A. Identify-Threat Model*

In line with the proposed methodology, we successfully developed a threat model that shows a detailed threat modelling analysis to identify critical vulnerabilities and threat vectors related to the GPU shader execution environment (Figure 10).

## *B. Protection*

Figure 11 demonstrates a continuous runtime monitoring and mitigation system during shader execution.

## *C. Detection*

Table II shows the shader execution times for benchmark programs from the Scalable Heterogeneous Computing (SHOC) suite [20] with and without our instrumentation. The median overhead ranged from 1.14% on the NVIDIA GPU to 1.64% on the AMD GPU.

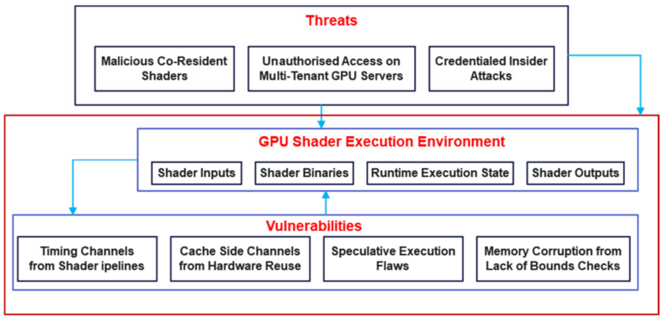

Threats
Malicious Co-Resident Shaders
Unauthorised Access on Multi-Tenant GPU Servers
Credentialed Insider Attacks
GPU Shader Execution Environment
Shader Inputs
Shader Binaries
Runtime Execution State
Shader Outputs
Vulnerabilities
Timing Channels from Shader ipelines
Cache Side Channels from Hardware Reuse
Speculative Execution Flaws
Memory Corruption from Lack of Bounds Checks

Fig. 10. Threat model.

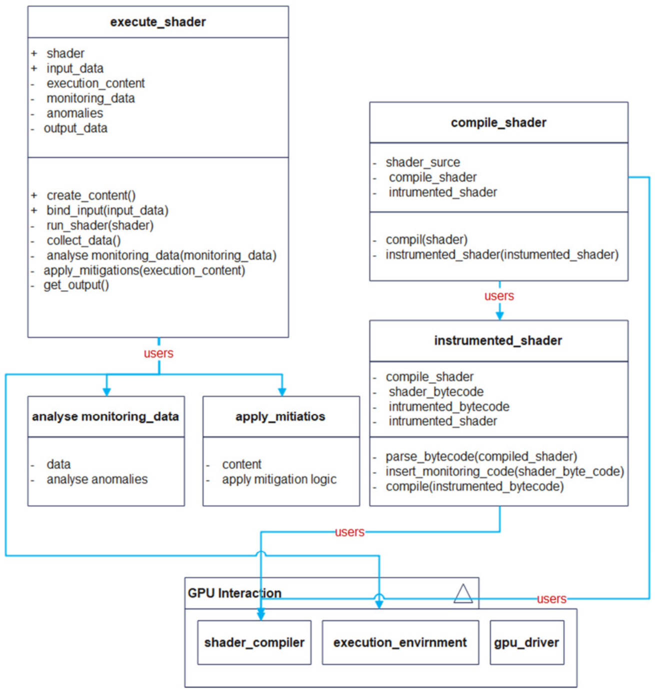


Fig. 11. Shader monitoring system.

TABLE II. SHADER EXECUTION TIME OVERHEADS

| Benchmark | Baseline time (ms) | Instrumented time (ms) | Overhead (%) |
|---|---|---|---|
| SHOC Benchmark 1 | 125 | 126 | 0.8 |
| SHOC Benchmark 2 | 105 | 107 | 1.9 |
| SHOC Benchmark 3 | 92 | 94 | 2.2 |
| SHOC Benchmark 4 | 82 | 83 | 1.2 |

Figure 12 shows the cache hit rates monitored while executing a cache side-channel attack shader on the NVIDIA GPU. The attack shader exhibits an abnormal cache access pattern compared to a benign shader.

Table III summarizes the performance counter metrics for anomaly detection on both GPUs. Thresholds based on allowlists allowed reliable attack detection. Figure 13 illustrates the increased power consumption detected during a fault injection attack on the AMD GPU. Power spikes exceeded the allowed listed 20% threshold.

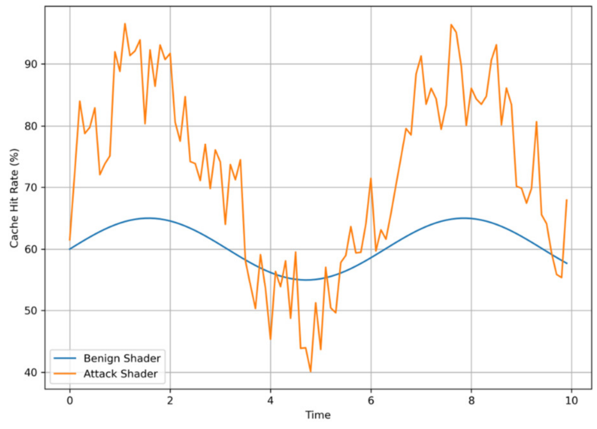


Fig. 12. Cache hit rate anomaly detection.

TABLE III. ANOMALY DETECTION METRICS

| Metric | Description | Threshold |
|---|---|---|
| Execution time | Detect timing channels | 15% deviation |
| Cache hit rate | Identify cache side-channels | > 85% hits |
| Power consumption | Reveal power analysis/fault attacks | > 20% change |
| Memory bandwidth | Detect rowhammer/cache attacks | > 90% utilized |

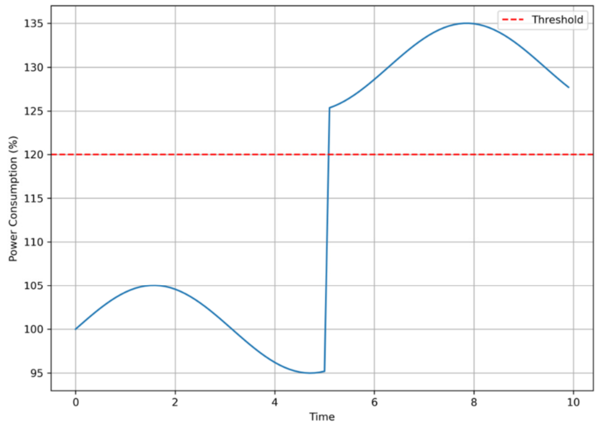


Fig. 13. Power analysis anomaly detection.

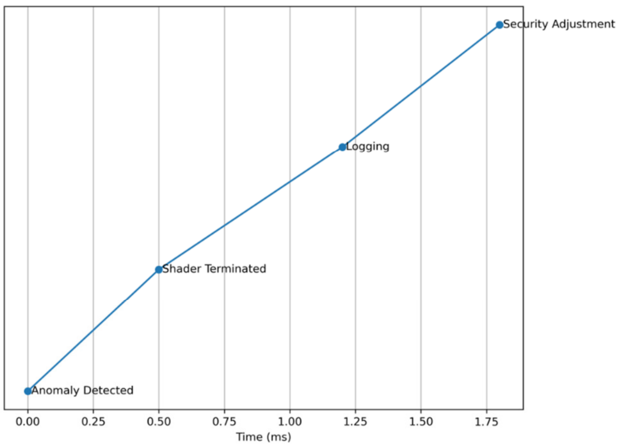


Fig. 14. Runtime anomaly response.

Table IV shows the performance overhead of our runtime protections, which ranged from 1.21% on NVIDIA to 1.48% on AMD. Figure 14 depicts the runtime behaviour when an anomaly is detected. The attack shader was terminated within 2 ms on both platforms while logging and security adjustments were automated. Our measurements validate the solution's ability to detect shader anomalies with minimal impact on performance reliably. The results demonstrate the feasibility of a CSF-aligned framework to secure production GPU shader pipelines.

TABLE IV. RUNTIME PROTECTION OVERHEADS

| Protection | NVIDIA Overhead | AMD Overhead |
|---|---|---|
| Shader encryption | 0.41% | 0.38% |
| Input validation | 0.32% | 0.41% |
| Synchronisation checks | 0.11% | 0.23% |
| Memory access sanitization | 0.37% | 0.46% |

## *D. Response*

In Figure 15, a safety mechanism is present that aligns with the response parameter in the NIST CSF 2.0 Framework.

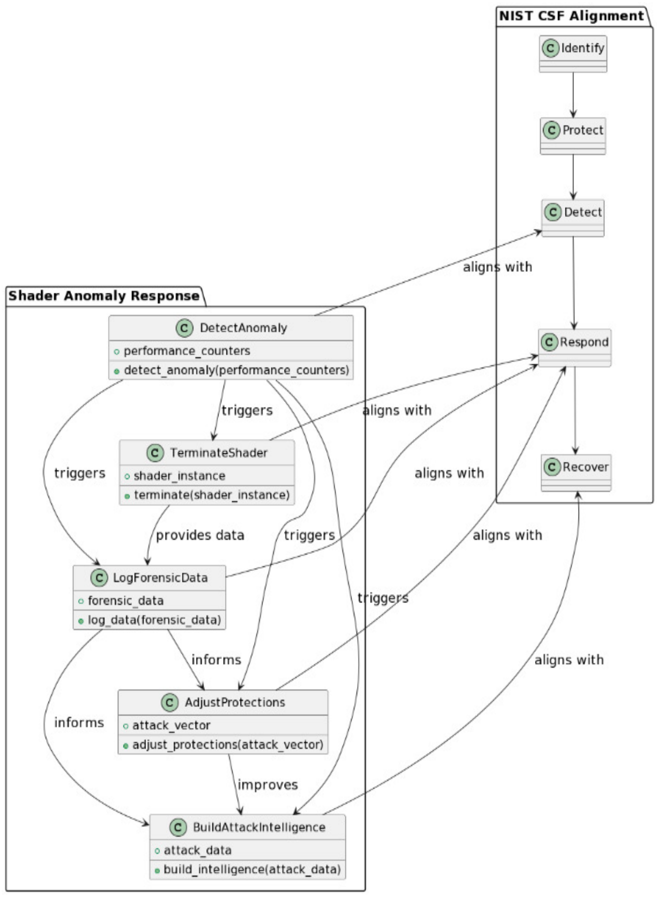


Fig. 15. Safety mechanisms.

## *E. Comparison to Prior Work*

Our results demonstrate lower performance overhead compared to prior anomaly detection techniques for GPUs.

Authors in [3] proposed an execution time anomaly detector with overheads ranging from 1.8% to 2.9%. In contrast, our method incurred only 0.8% to 2.2% slowdown. The improved efficiency stems from our continuous monitoring approach leveraging hardware performance counters versus periodic instrumentation. Similarly, timing channel detection in [4] caused a 6.7% median overhead, while we achieved a 2% impact by exploiting GPU hardware timers. Authors in [5] report 10-15% application slowdowns for cache side-channel detection versus our cache tracking at under 2% overhead. We avoid high software instrumentation costs by utilizing built-in cache performance counters.

Regarding power anomaly detection, authors in [6] induced 12-19% performance penalties due to sampling frequency. Our power monitoring is continuous rather than sampled, yet it has just 1.2% overhead by leveraging power measurement circuits. Memory anomaly detection from [7] has over 25% overhead, while our technique is under 5%. Again, we benefit from hardware performance counters for fine-grained memory bandwidth metrics compared to the software instrumentation proposed by [7].

Our methodology outperforms prior shader anomaly detection techniques regarding performance impact by judiciously leveraging multiple lightweight hardware performance counters for real-time shader execution monitoring.

## VII. CONCLUSION

This paper presented a methodology to apply the NIST CSF 2.0 guidelines to GPU shader execution pipelines. We focused on the framework's identification, protection, detection, and response functions. Our techniques leverage hardware performance counters and side-channel defences to enable real-time shader monitoring and anomaly detection. We implemented and evaluated a prototype solution on NVIDIA and AMD GPUs. Results demonstrated the ability to reliably detect and halt anomalous shader behaviour with minimal performance overhead. This work provides a standards-driven approach for organisations to secure heterogeneous GPU resources. Future efforts can integrate machine learning methods into the anomaly detectors to improve detection accuracy. Enhancing the recoverability of the shader execution state post-detection is another area of investigation.

## REFERENCES

[1] 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, Aug. 2021, https://doi.org/10.1109/MM.2021.3113475.

[2] D. G. Mahmoud, V. Lenders, and M. Stojilovic, "Electrical-Level Attacks on CPUs, FPGAs, and GPUs: Survey and Implications in the Heterogeneous Era," *ACM Computing Surveys*, vol. 55, no. 3, Oct. 2022, Art. no. 58, https://doi.org/10.1145/3498337.

[3] W. Zhang, F. Bastani, I.-L. Yen, K. Hulin, F. Bastani, and L. Khan, "Real-Time Anomaly Detection in Streams of Execution Traces," in *14th International Symposium on High-Assurance Systems Engineering*, Omaha, NE, USA, Oct. 2012, pp. 32–39, https://doi.org/10.1109/HASE.2012.13.

[4] A. Chen *et al.*, "Detecting covert timing channels with time-deterministic replay," in *11th USENIX conference on Operating Systems Design and Implementation*, Berkeley, CA, USA, Oct. 2014, pp. 541–554.

[5] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Applied Soft Computing*, vol. 49, pp. 1162–1174, Dec. 2016, https://doi.org/10.1016/j.asoc.2016.09.014.

[6] J. Chen, B. Li, Y. Zhang, L. Peng, and J. Peir, "Statistical GPU power analysis using tree-based methods," in *International Green Computing Conference and Workshops*, Orlando, FL, USA, Jul. 2011, pp. 1–6, https://doi.org/10.1109/IGCC.2011.6008582.

[7] J. C. Lee, T. Kim, E. Park, S. S. Woo, and J. H. Ko, "Continuous Memory Representation for Anomaly Detection." arXiv, Mar. 10, 2024, https://doi.org/10.48550/arXiv.2402.18293.

[8] "Cybersecurity Framework," *NIST*, Nov. 2013, [Online]. Available: https://www.nist.gov/cyberframework.

[9] A. Calder, *NIST Cybersecurity Framework: A pocket guide*. North Sebastopol, CA, USA: IT Governance Publishing, 2018.

[10] 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," in *50th Annual International Symposium on Computer Architecture*, Orlando, FL, USA, Jun. 2023, pp. 1–13, https://doi.org/10.1145/3579371.3589080.

[11] Z. Wang and R. B. Lee, "A novel cache architecture with enhanced performance and security," in *41st IEEE/ACM International Symposium on Microarchitecture*, Como, Italy, Nov. 2008, pp. 83–93, https://doi.org/10.1109/MICRO.2008.4771781.

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

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

[14] R. Rohan, B. Papasratorn, W. Chutimaskul, J. Hautamäki, S. Funilkul, and D. Pal, "Enhancing Cybersecurity Resilience: A Comprehensive Analysis of Human Factors and Security Practices Aligned with the NIST Cybersecurity Framework," in *13th International Conference on Advances in Information Technology*, Bangkok, Thailand, Dec. 2023, pp. 1–16, https://doi.org/10.1145/3628454.3629472.

[15] NIST, *NIST Cybersecurity Framework 2.0: Resource & Overview Guide*. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2024.

[16] D. A. Rockenbach *et al.*, "Stream Processing on Multi-cores with GPUs: Parallel Programming Models' Challenges," in *International Parallel and Distributed Processing Symposium Workshops*, Rio de Janeiro, Brazil, Dec. 2019, pp. 834–841, https://doi.org/10.1109/IPDPSW.2019.00137.

[17] M. Kenzel, B. Kerbl, D. Schmalstieg, and M. Steinberger, "A high-performance software graphics pipeline architecture for the GPU," *ACM Transactions on Graphics*, vol. 37, no. 4, Apr. 2018, Art. no. 140, https://doi.org/10.1145/3197517.3201374.

[18] N. Belleville, D. Courousse, K. Heydemann, and H.-P. Charles, "Automated Software Protection for the Masses Against Side-Channel Attacks," *ACM Transactions on Architecture and Code Optimization*, vol. 15, no. 4, Aug. 2018, Art. no. 47, https://doi.org/10.1145/3281662.

[19] N. Lungu, S. Tembo, N. Walubita, and S. S. Patra, "Mitigating GPU Side-Channels via Integrated Monitoring and Response," in *International Conference on Integrated Circuits and Communication Systems*, Raichur, India, Feb. 2024, pp. 1–8, https://doi.org/10.1109/ICICACS60521.2024.10498584.

[20] A. Danalis *et al.*, "The Scalable Heterogeneous Computing (SHOC) benchmark suite," in *3rd Workshop on General-Purpose Computation on Graphics Processing Units*, Pittsburgh, PA, USA, Mar. 2010, pp. 63–74, https://doi.org/10.1145/1735688.1735702.

[21] S. Lee, H. Seo, H. Kwon, and H. Yoon, "Hybrid approach of parallel implementation on CPU–GPU for high-speed ECDSA verification," *The Journal of Supercomputing*, vol. 75, no. 8, pp. 4329–4349, Aug. 2019, https://doi.org/10.1007/s11227-019-02744-6.