Enabling Cheat Detection in Multiplayer Online Games

Hugo Bertin
Univ Rennes, CNRS, INRIA, IRISA
Rennes, France
hugo.bertin@irisa.fr

Salman Shaikh
KAUST
Thuwal, Saudi Arabia
salman.shaikh@kaust.edu.sa

Marc Dacier
KAUST
Thuwal, Saudi Arabia
marc.dacier@kaust.edu.sa

Yérom-David Bromberg
Univ Rennes, CNRS, INRIA, IRISA
Rennes, France
david.bromberg@irisa.fr

***Abstract*—The video games industry reached a revenue of US$455.27 billion in 2024 [1], with Multiplayer Online Games (MOGs) as the figurehead. However, this industry, which depends on user satisfaction, is threatened by cheating actions that degrade the gaming experience. Cheaters can exploit the distributed nature of MOGs with network flow disruption attacks, manipulating the network to disrupt game synchronization. Detecting those attacks is challenging due to the difficulty of distinguishing between instabilities and latency inherent in the network and artificial manipulations. Our conducted research aims to seek for detection techniques. In this objective, the first step is the implementation of a framework that can be leveraged to generate data that helps understand whether it is possible to differentiate between honest and cheating behaviors. This article describes the preliminary experimental results obtained, highlighting the effects of network flow disruption attacks on MOGs. The framework is planned to be deployed online to streamline research on detection solutions as part of future research plans.**

***Index Terms*—multiplayer, online, game, cheating, security**

I. Introduction

In 2024, Multiplayer Online Games (MOGs) attracted 1.2 billion players, while Esports viewership rose to over 500 million viewers [2]. This popularity is threatened by cheaters seeking to gain unfair advantages against opponents. Cheating directly impacts game editors as it drives honest players away from games plagued by dishonest behaviors.

MOGs are distributed systems that must maintain the user experience despite the inherent latency of network communication. This opens a door to a class of cheats that manipulate the information exchange on the network to disrupt the synchronization between the system nodes. Technically, these cheats result in an increased communication latency between two nodes of an MOG system. Examples of such attacks are **Lag Switch** and **DDoS** [3], [4]. Individually, these attacks can be prevented. However, their strength lies in their duality, making them unsolvable when considered together [5]: preventing one attack enables the other. Hence, to defeat cheaters, MOGs should prioritize detection techniques. However, detecting network flow manipulation is challenging for a system that has no control over the network.

Our research aims to study network flow disruption cheats detection mechanisms. The key question is whether genuine latency can be differentiated from cheating latency based on the information received by the nodes of the system. To approach this question, data containing behavioral traces in a MOG environment is needed. To the best of our knowledge, no dataset, labeling cheater and honest behavior, exist.

We started by designing a framework that can be leveraged to generate a labeled dataset containing honest and cheating behavioral traces in a MOG system. This aims to integrate, in a MOG environment, the execution of network flow disruption attacks alongside a logging mechanism. The framework is configurable and generic to different types of games.

We present early results of traces generated with the framework. They highlight the behavioral implications of network flow disruption attacks on the different game synchronization mechanisms. This validates the effectiveness of the framework and serve as a stepping-stone for further work on network flow disruption cheats. We plan to deploy the framework online to collect further data in real gaming environments. In a second phase, after studying the data obtained, the framework can be leveraged to experiment with detection strategies.

The framework and the preliminary results are part of work presented in a paper [6].

Section II presents the background regarding MOGs systems and network flow disruption attacks. Section III presents the framework proposed to study those cheats. Section IV shares the early results of the framework in generating behavioral traces. Section V presents the future research plans. Section VI concludes the paper.

II. Background

MOGs are distributed systems that are often implemented following a Client-Server architecture in the gaming industry [7]. In this model, the server maintains the authoritative game state based on updates received from clients and synchronizes the clients' states accordingly. However, players experience varying network latencies when communicating with the server, which can introduce inconsistencies across players [8].

The software executed on the nodes of MOGs is based on a game engine. Game engines are software frameworks implementing libraries commonly needed by game developers, such as graphics rendering, physics engine, as well as networking and synchronization components [9]. This paper solely focuses on Unreal Engine, whose source code is available publicly. However, it is quite likely that other game engines, such as Unity, are not immune to these issues.

A. Cheating in Online Games

Cheating is a major problem in the video game industry [10]. Current research focuses on client-level cheats. They are malicious programs that tamper with the game executable to modify legitimate game behavior [11]–[13]. Such examples are bots [14]–[16], or Information Exposure cheats [16], [17]. Software solutions exist to counter these threats with anti-cheats [18], [19]. However, as in the anti-virus domain, a never-ending arms race persists between developers and cheaters.

Cheats can also be carried out at the network level. Network-level cheats have largely been overlooked, which emphasizes the reasons for this work. We classify them into the following categories:

Packet Tampering. These attacks exploit the content of the packets and the related network protocols used by MOGs [20]–[22]. Nonetheless, packet tampering cheats can be mitigated by secure protocols to authenticate users and protect packets' integrity and confidentiality [23]. Furthermore, mitigation at the receiving end of the packet is possible.

Network Flow Disruption Cheats. This is a class of cheating attacks that manipulate the flow of messages exchanged between the different nodes of a MOG to disrupt their synchronization. Technically, they delay (or drop) the packets transiting on the network link without modifying the data embedded. Two types of attacks are described hereafter. Their effectiveness is demonstrated by a video recording available in a publicly accessible Dropbox folder [24].

(i) *Lag Switch (LS)*: Attackers buffer outgoing updates for a specific amount of time before sending them to the server simultaneously [5], [18], [25]. Packets arrive in a burst at the server. Upon receiving the packets, the server may apply different synchronization mechanisms to reconcile these updates and maintain the game state consistency for all players [8]. In the experiments, we can observe that after an attack, the server considers objects individually to reconcile their states [24]. An attacker who buffers movement updates introduces inconsistencies between his avatar's location on the client and the server. When he releases the packets in a burst at the end of the attack, the reconciliation cancels those movements and relocates the attacker to a location dictated by the server. However, when the attacker buffers shooting actions and sends them in a burst, the shots are not canceled by the server. They are accepted by the server and replicated to the other clients. This happens even when the shot is not possible on the other clients' versions. This object-based synchronization makes LS effective as the opponent may not see the attacker because his movements were canceled, but still dies because the server accepted the attacker's shot.

(ii) *Distributed Denial-of-Service (DDoS)*: Attackers launch attacks that use multiple compromised devices to overwhelm a target [26], which is either the server or a specific player. Server DDoS can disrupt gameplay for all players. It can be used by cheaters to stop a game they are likely to lose [27]. Player-targeted DDoS delays game updates to and from the player [5]. The victim of this attack experiences conditions that are similar to suffering from a poor network infrastructure.

B. Network Flow Disruption Cheats are Unsolvable

Benhabbour et al. [5] highlight the duality between LS and DDoS. With LS attacks, attackers add delay to their outgoing updates. On the other hand, DDoS attacks target other players, preventing them from sending or receiving updates on time. If a server accepts late incoming packets, it makes the game vulnerable to LS attacks. If a server rejects late incoming packets, it allows DDoS attacks to affect honest players, causing the server to invalidate their legitimate actions. This duality hinders the development of mitigation strategies, as the prevention of one attack enables the existence of the other.

Thus, if this class of attack cannot be prevented, the target should shift on detecting cheaters. Detecting network flow disruption attacks appears to be challenging as it requires differentiating between genuine network latency and delay artificially introduced by attackers.

The literature does not provide effective methods to detect LS. Tandon et al. [25] uses a duration threshold to distinguish between accidental and artificial delay. This approach risks misclassifying legitimate players experiencing abnormally high delay as cheaters. Additionally, no method to select the threshold value is proposed. Chen et al. [28] propose to detect cheating by comparing Round Trip Time (RTT) with ping times. However, an attacker may also manipulate the timing of ping messages, including delaying them to match the delays applied to game packets.

Detecting DDoS attacks is also challenging as victims of DDoS experience regular connectivity issues like latency, packet loss, or disconnection.

Thus, the next step should be to develop effective detection techniques to identify network flow disruption cheats. However, to the best of our knowledge, there is no real-world representative dataset that would enable researchers to come out with applicable detection techniques. Addressing that issue is the goal of the framework implemented for my research, presented in Section III.

III. A Framework to record cheating behaviors

The framework presented aims to experiment with network flow disruption attacks and generate detailed traces that can serve as a foundation for future works on developing mitigation techniques.

A. Specifications and Design of the Framework

The first objective of the framework is to allow players to carry out network flow disruption attacks in a MOG to observe how they proceed. For ethical reasons and to make the game playable, we plan to hide cheats behind a game logic providing *superpowers* to the users, hiding the execution of a cheat. For instance, using a DDoS can be hidden behind a superpower, freezing an opponent. These *superpowers* must behave similarly to the observation realized in previous real-life examples. The framework is interleaved

with the underlying game engine, enabling the collection of data about the attacks and the game behavior on the different nodes of the system. Among attacks presented in Section II-A, the framework implements the following attacks: lag switch and DDoS targeted against a player.

Thus, using the framework permits the collection of data that helps understand the system's behavior under attacks and paves the way for further dataset analysis to understand whether genuine latency can be differentiated from artificially added latency. The data includes logging game events, labeled when a specific player is using a specific cheat. These logs are collected on each client and the server. On the client side, the framework pushes back the logs to the server, enabling synchronization to compare the logs over time.

The second objective of the framework is to automate both the provided attacks and the game scenarios. This implies automating the possible game events (e.g. player's movements, shooting action) as well as the cheats. This facilitates the reproducibility of the experiments and allows for testing different scenarios and their variants to identify their respective implications. By automating these processes, a large dataset can be generated through multiple runs of each scenario, minimizing the impact of outliers.

The third objective of the framework is to be generic with different MOGs. The interleaving in the game engine facilitates its deployment with any game implemented on top of the same game engine (UE in our implementation). This feature can also be leveraged to determine if the same cheat presents the same effect on another type of game and to deduce if detection techniques can be approached in a generic way (a technique applicable to all types of games) or if they should target a specific type of game. Furthermore, the framework is configurable and adaptable for other types of cheats that researchers may want to implement for other purposes.

B. Implementation

The implementation is made of three main modules, described briefly hereafter: i) the attacks, ii) the logging, and iii) the automation modules.

1) Attacks Module: The cheats are directly integrated into the framework, which facilitates the mappings between the game UI and the cheats through UE's console commands. We developed LS and DDoS with `NetEmulation`, a utility included within UE to simulate network conditions such as latency and packet losses.

Lag Switch. The framework buffers the packets for a specific delay. Then, all the packets are sent together once the delay has elapsed. This causes a burst of packets to be received by the other end.

DDoS. Executing a DDoS attack is more complex than LS to implement. For ethical reasons, we do not want to overload players' networks with an excessive amount of packets, which could disrupt other services than the game, as it is the case in a real DDoS attack. Hence, we simulate the outcomes of a DDoS attack. It is directly realized by the victim's client, who is told to drop packets (incoming and outgoing). The attacker starts the attack by sending the victim's game identifier to the server, which instructs the victim's client to simulate the DDoS for a specified delay. The severity of the attack simulated can be configured: the amount of packet losses and delay.

2) Logging mechanism: The framework provides a logging mechanism to gather traces that can be analyzed to study the behavior at the different nodes of the system under an attack. The framework can collect data about the game objects, such as the player's location or health, to observe and compare their evolution on the different nodes. The data managed at the game layer (e.g., health) is logged through methods provided to the game late and thus requires minimal adjustment from the game developer. Objects managed at the engine level (e.g., movements) are directly logged. The framework can be extended to log other objects' states accessible from the engine level.

The game events taking place are recorded on each client and on the server. The mechanism works as follows:

- Logging starts at the beginning of the game. The object logged are configurable and can include player data such as *Location, Health, Inventory*. This data is labeled as honest.
- When a player starts using a *superpower* (i.e., cheating), the logs recorded are labeled accordingly.
- The logs are continuously serialized and stored in a separate file on each client and on the server. The clients periodically send their logs to the server, where they are managed and organized by game time.

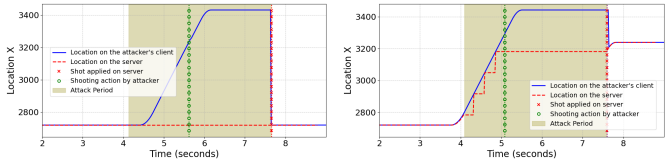
3) Automation Module: To reproduce multiple runs without manual interference that may introduce discrepancies, we design a Domain Specific Language (DSL) to automate the game scenarios, based on JSON.

The DSL defines game events containing fields such as times, positions, shots, and attacks. Each field is configured to ensure that the player's actions are executed at the desired time and in the intended manner. Once the run finishes, the logs are collected, and the same run can be repeated or modified by adjusting the game scenario.

IV. Early Experimental Results

This section presents the preliminary experimental results obtained, leveraging the framework presented in Section III. They highlight the implications of a MOG system while launching the different types of network flow disruption cheats implemented. The MOG and the framework are developed with Unreal Engine. The evaluation has been realized in a LAN environment. The results presented are examples of patterns observed consistently across multiple runs. The figures in this paper have been produced by parsing the logs generated by different game scenarios. The testbed includes one server and two clients. We perform experiments on each network flow disruption attack. Our DSL enables us to write a specific game scenario for each attack (and its variants).

1) Consequences of Lag-Switching: This experiment presents the implications of LS attacks and highlights the extrapolation implemented for the movements in UE.



(a) Attacker Stationary (b) Attacker Moving

Fig. 1: Variation of LS Cheats, with time elapsed on the x-axis and the avatar's x-coordinate on the y-axis

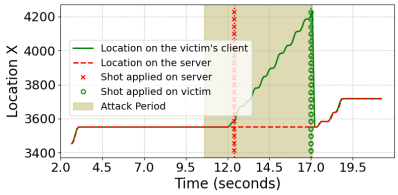


Fig. 2: Consequences of DDoS attacks, with time elapsed on the x-axis and the avatar's x-coordinate on the y-axis

Server extrapolation is a common technique used by MOGs to minimize the lag for players experiencing unstable network connectivity [8]. When the server does not receive information from the client (or receives it delayed), it extrapolates the player's position based on the last data it received. To demonstrate this behavior, we designed two game scenarios. The first scenario demonstrates that if the player is not moving, the server assumes the player will remain stationary. The results of this scenario are presented in Figure 1a, where the player starts to move (the straight, blue, line) only after having started to buffer its packet (starting a LS attack). The second scenario demonstrates an attacker who is already moving when starting the attack. In this case, the server assumes that the player continues moving in the same direction for some time. This scenario is illustrated in Figure 1b, showing an attacker in motion before initiating the attack. It is worth noting that the extrapolation is realized following a pattern that resembles stairsteps, highlighting that the engine is waiting to receive information for a specific delay before extrapolating the player's position. In both scenarios, the attackers's shoot is only considered by the server at reception.

At the end of the LS attack (after 3 seconds), when the burst of delayed information arrives at the server, the server compares the position claimed by the attacker with its extrapolated position. If there is a discrepancy, the server forces the attacker to match the server's recorded position, as seen in Figures 1a and 1b.

2) Consequences of DDoS on a targeted player.: We recorded the behavior of the MOGs system while a DDoS attack targets a player. The framework simulates 95% of packet losses on the network link between the server and the victim in both directions. The results are presented in Figure 2.

We observe the evolution of the victim's location with time, on the victim's client (straight green line) and the server (dashed red line). We also notice when the attacker's shot is registered by the server (vertical red 'x' line) and the victim's client (vertical green 'o' line). Discrepancies are seen within the movements; when the victim is under attack, their movements are not seen on the server instantly. Sometimes, some of their movement updates are considered by the server, but the major effect observed is that the victim is relocated to their original position whenever packets from the server reach them. In particular, when the victim eventually gets the packet from the server informing of the shot, one can also see that the server moves the client back to where, in the server world's view, the victim was when the shot happened. This results from information embedded in 5% of packets being transmitted. This demonstrates that DDoS attacks cause significant deviations between the server's state of the game and the state of the victim's client.

V. Research Plans

This section presents the future research plans to seek detection strategies against network flow disruption cheats.

First, the objective is to collect data that helps to better understand the inherent implications of such cheats, leveraging our framework. The preliminary results presented were conducted within a LAN, which minimizes network latency. However, this is not the case for real MOGs, which rely on the Internet, introducing inherent latency and instability. Thus, we plan to deploy our framework online to collect logs at a larger scale on the Internet. The framework will be integrated within the Lyra Starter Game [29], an open-source game provided by Unreal Engine as a sample for multiplayer game development. This will allow players from different locations to participate in game sessions. The use of cheats will be authorized and gamified as superpowers or extra abilities, facilitating the collection of cheating and honest data in real-world online gaming environments.

In a second phase, this dataset collected online can be leveraged to differentiate between cheaters and honest players in MOGs and enable research on detection strategies.

The potential detection strategies could be implemented within the framework for experimentation and evaluation.

VI. Conclusion

Network flow disruption cheats manipulate the latency to disrupt game synchronization. This work demonstrated their effectiveness on real MOGs. To streamline research for detection techniques, we propose a dedicated framework providing attacks implementations, a DSL to automate game events and attacks, and a logging system to generate labeled datasets of MOGs behavior under cheating attacks. The preliminary experimental results illustrated the framework's effectiveness in collecting data and emphasized the complexity of the implications that network flow disruption cheats have on game synchronization.

As part of future works, we plan to deploy the framework online to collect players' data in a real MOG environment. This is to enable the research on distinguishing between honest and cheating behaviors.

REFERENCES

[1] J. Clement, "Global video game industry revenue 2029," Sep 2024. [Online]. Available: https://www.statista.com/forecasts/1344668/revenue-video-game-worldwide

[2] N. Kumar, "23 esports viewership statistics of 2025 (audience growth data)," Nov 2024. [Online]. Available: https://www.demandsage.com/esports-statistics/

[3] capitalraider, "Cheats: You know someone is using a lag switch..." [Online]. Available: https://gtaforums.com/topic/359863-cheats-you-know-someone-is-using-a-lag-switch/

[4] Gender_Pronouns, "Lag switch hack is ruining the game." [Online]. Available: https://forum.escapefromtarkov.com/topic/116665-lag-switch-hack-is-ruining-the-game/

[5] I. Benhabbour, Y.-D. Bromberg, M. Dacier, S. Dietrich, R. M. Rodrigues, and P. Esteves-Verissimo, "Attacks on tomorrow's virtual world," in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, 2023, pp. 105–110.

[6] S. Shaikh, H. Bertin, M. Dacier, and Y.-D. Bromberg, "A framework for generating datasets to improve cheating detection in multiplayer online games," in *Proceedings of the 2025 European Interdisciplinary Cybersecurity Conference*, 2025.

[7] A. M. Khan, I. Arsov, M. Preda, S. Chabridon, and A. Beugnard, "Adaptable client-server architecture for mobile multiplayer games," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, 2010, pp. 1–7.

[8] S. Liu, X. Xu, and M. Claypool, "A survey and taxonomy of latency compensation techniques for network computer games," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–34, 2022.

[9] Phil, "What is a game engine: An essential overview for beginners - draw & code," Jan 2024. [Online]. Available: https://drawandcode.com/learning-zone/what-is-a-game-engine/

[10] N. Granados, "Report: Cheating is becoming a big problem in online gaming," Apr 2018, [Accessed 12-03-2025]. [Online]. Available: https://www.forbes.com/sites/nelsongranados/2018/04/30/report-cheating-is-becoming-a-big-problem-in-online-gaming/

[11] M. S. Anwar, C. Zuo, C. Yagemann, and Z. Lin, "Extracting Threat Intelligence From Cheat Binaries For Anti-Cheating," in *26th International Symposium on Research in Attacks, Intrusions and Defenses*, ser. RAID '23. ACM, 2023, p. 17–31.

[12] E. Kaiser, W.-c. Feng, and T. Schluessler, "Fides: remote anomaly-based cheat detection using client emulation," in *16th ACM Conference on Computer and Communications Security*, ser. CCS '09. ACM, 2009, p. 269–279.

[13] P. Karkallis, J. Blasco, G. Suarez-Tangil, and S. Pastrana, "Detecting video-game injectors exchanged in game cheating communities," in *26th European Symposium on Research in Computer Security (ESORICS)*. Springer, Oct 2021, pp. 305–324.

[14] M. Choi, G. Ko, and S. K. Cha, "BotScreen: Trust Everybody, but Cut the Aimbots Yourself," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 481–498.

[15] M. L. Han, J. K. Park, and H. K. Kim, "Online game bot detection in fps game," in *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems-Volume 2*. Springer, 2015, pp. 479–491.

[16] S. D. Webb and S. Soh, "Cheating in networked computer games: a review," in *2nd International Conference on Digital Interactive Media in Entertainment and Arts*, 2007, p. 105–112.

[17] J. Hao and W. Cai, "Measuring information exposure attacks on interest management," in *2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*. IEEE, 2012, pp. 133–144.

[18] A. Maario, V. K. Shukla, A. Ambikapathy, and P. Sharma, "Redefining the risks of kernel-level anti-cheat in online gaming," in *2021 8th International Conference on Signal Processing and Integrated Networks (SPIN)*. IEEE, 2021, pp. 676–680.

[19] S. Lehtonen *et al.*, "Comparative study of anti-cheat methods in video games," *University of Helsinki, Faculty of Science*, 2020.

[20] P. Laurens, R. F. Paige, P. J. Brooke, and H. Chivers, "A novel approach to the detection of cheating in multiplayer online games," in *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*. IEEE, 2007, pp. 97–106.

[21] J. Smed, T. Kaukoranta, and H. Hakonen, "Aspects of networking in multiplayer computer games," *The Electronic Library*, vol. 20, no. 2, pp. 87–97, 2002.

[22] H. Bertin, I. Benhabbour, M. Dacier, and Y.-D. Bromberg, "Disconnecting games with a single packet: An unreal untold story," hack.lu 2024. [Online]. Available: https://www.youtube.com/watch?v=ZpMHGBULZgU

[23] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018.

[24] game-sec lab, "Fd, ls and ddos attacks on fortnite," https://www.dropbox.com/scl/fo/k1gjem9akbc1io7vw8ysk/AM_Uza91qNa9DTlHV_noCy0?rlkey=blaa5db9mcmjg816r0q9l0yhg&st=jdusdade&dl=0, 2024, dropbox folder.

[25] V. Tandon and J. Devore, "Detecting lag switch cheating in game," May 2 2017, uS Patent 9,636,589.

[26] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet denial of service: attack and defense mechanisms (Radia Perlman Computer Networking and Security)*. Prentice Hall PTR, 2004.

[27] S. Moosa, "Overwatch 2 cheaters are crashing servers to avoid losing games," Jun 2024. [Online]. Available: https://www.ginx.tv/en/overwatch-2/cheaters-are-crashing-servers-to-avoid-losing-games

[28] B. D. Chen and M. Maheswaran, "A cheat controlled protocol for centralized online multiplayer games," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, 2004, pp. 139–143.

[29] Andy, "Lyra starter game," https://dev.epicgames.com/community/learning/paths/Z4/lyra-starter-game, 2022, accessed: (2025-01-13).