

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

SoK: Another Arms Race – 20 Years of (Anti-)Cheating in Video Games

Philip Klostermeyer, Jan-Ulrich Holtgrave, Jacques Suray, Anne Vonderheide, Nicolas Lodzig, Niklas Busch, Alexander Krause, Jan H. Klemmer, Marten Oltrogge, and Sascha Fahl, *CISPA Helmholtz Center for Information Security*

https://www.usenix.org/conference/usenixsecurity26/presentation/klostermeyer

This paper is included in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

ARTIFACT EVALUATED
USENIX
AVAILABLE

SoK: Another Arms Race – 20 Years of (Anti-)Cheating in Video Games

Philip Klostermeyer, Jan-Ulrich Holtgrave, Jacques Suray,
Anne Vonderheide, Nicolas Lodzig, Niklas Busch,
Alexander Krause, Jan H. Klemmer, Marten Oltrogge,
Sascha Fahl

CISPA Helmholtz Center for Information Security, Germany
{philip.klostermeyer,jan-ulrich.holtgrave,jacques.suray,
anne.vonderheide,nicolas.lodzig,niklas.busch,
alexander.krause,jan.klemmer,fahl}@cispa.de

Abstract

Video gaming is one of the largest sectors of the software industry. Game security is a paramount yet uniquely challenging concern. Beyond defending against conventional security threats, video games must deal with a highly motivated adversary from their own ranks: the cheater.

Cheaters seek to gain unfair advantages, undermining both the economic viability and the competitive fairness of video games. By the virtue of being an ever existing plague, the industry and related work indicate having hardships with addressing cheating. In particular, the community lacks a cohesive, systematic understanding of cheating in video games. Current literature is fragmented, focusing on isolated exploits rather than a unified architectural view of the threat landscape.

Our work addresses this critical gap by providing a comprehensive systematization of video game cheating. We systematize 187 scientific publications spanning the last two decades. Based on established knowledge systematization methodologies, we synthesize our findings into a detailed, holistic attack tree, spanning six major cheating areas with 192 nodes. We map the heterogeneous cheating strategies to the MITRE ATT&CK framework, providing a structured taxonomy of game-specific attack vectors. We enrich our work by presenting 54 safeguards for the attacks.

1 Introduction

Video games have been one of the highest-grossing markets for software products for decades [135]. At the same time, games belong to the most complex software products [100], as they integrate numerous specialized workflows involving many professionals, not all of whom have a software development background [57]. Additionally, game software projects are heavily affected by extremely tight deadlines and budgets [75]. This complex environment provides fertile ground for malicious actors to repeatedly carry out a plethora of security attacks on data, infrastructure, and other monetary values [149, 48, 31, 13, 139]. Serious examples, such as the FromSoftware incident [151] or security vulnerabilities in kernel anti-cheat drivers [92], drastically demonstrate how security flaws in game code or auxiliary mandatory software can compromise millions of devices.

In a study from 2024 with industry stakeholders, Klostermeyer et al. identified five key security areas of concern for game development, with handling *cheating* being named as the top priority [75]. Consalvo defined the concept of cheating through interviews with players as *"giving the player an unfair advantage"* [28]. From a technical perspective, cheating entails activities closely aligned with computer security threats [79]. *Cheaters* need extensive knowledge about, e.g., how to gain unauthorized access to data in networks or storage [162], reverse engineering of protocols, data, and structures [145], process injection and hijacking of execution flows to manipulate gameplay [27], and more. Consequently, the skills and techniques required by a cheater overlap significantly with those used in traditional methods of secure software development and offensive security (i.e., attacker perspective), thus motivating the investigation of cheating as a scientific problem in computer security.

Precariously, *anti-cheating* is associated with several technical pitfalls. Developers require deploying complex, state-of-the-art security tooling outside their domain, comparable to those used in high-security environments such as banking applications (e.g., anomaly detection, zero trust security), to detect and prevent cheating, for which they lack the necessary expertise [79]. In addition, developers must balance other similar important non-functional requirements for a game, which often conflict with anti-cheating measures. For example, popular massively multiplayer online games (MMOGs) such as *World of Warcraft* need to balance scalability with fast synchronization of game states across many connected clients by sacrificing strict consistency [59]. Further, minimal latency and bandwidth requirements, especially in fast-paced shooter games [158] or fluctuations of mobile devices [160], require sacrificing on encryption and authentication. There are various centralized, decentralized, or hybrid approaches to meet these requirements, depending on genre, features,

and design of a game. However, all approaches affect anti-cheating solutions with own challenges. If ignored, cheating can have dramatic consequences, like sales declining, customers migrating away, and development projects failing [75].

Therefore, anti-cheating measures should be considered as early as possible in the development of a game, as fixing software issues at a later stage invariably entails higher costs [54]. Appropriate guidelines and actionable artifacts for addressing cheating are thus crucial. Previous work addressed cheating either in specific areas, such as architectural considerations [38], particular genres [133], or examined the topic more abstract [156]. Yan and Randell provided an important first step with their 2005 cheating taxonomy, highlighting 15 forms of cheating [162]. However, there is a lack of a comprehensive taxonomy that covers all aspects in the necessary technical depth, paired with an exhaustive compilation of actionable safeguards. Moreover, cheating is often described as an *ongoing arms race* between cheaters and defenders, making it necessary to be always up-to-date.

To close this gap, we perform a Systematic Literature Review (SLR) of 187 papers about cheating and anti-cheating from the last 20 years.We then utilized the results of our systematic analysis and extract game specifics, attack patterns, and proposed safeguards from the academic papers to create a detailed attack tree [123, 90] as an actionable artifact for the security and developer community (Figure 2). To the best of our knowledge, our systematization is unprecedented in its depth. We contend that such an elaboration of the overall picture of cheating not only provides valuable insights and suggestions for future research, but also enables practitioners to perform threat modeling for video games [43] using our attack tree as an established baseline for describing security threats [80]. Attack trees are a proven tool in computer security, even in cases involving users with limited technical backgrounds [45], aligning with industry circumstances. Further, we enrich our tree with a mapping to categories from the *MITRE ATT&CK Matrix for Enterprise* [42], building a bridge between standard work for cybersecurity defenses and the games industry. Thus, we formulate the following research questions and contributions:

RQ1 *Comprehensive Systematization: What cheating and anti-cheating research has been conducted in the past 20 years?* We identify which topics have been investigated over the last 20 years of research on cheating and anti-cheating, and highlight key areas for future work.

RQ2 *Developing an Attack Tree: What scenarios for cheating attacks exist, and what do they technically encompass?* Our attack tree provides a comprehensive overview of cheating techniques and enables threat modeling for developers. From a cheater's perspective, we examine all the attacks investigated and provide a vertical breakdown of technical cheating.

RQ3 *Safeguarding Against Cheating: What safeguards exist against which cheating attacks, and what does a comprehensive list of technical safeguard mechanisms against cheating contain?* Cheating attacks span a wide range of scenarios that occur at many different technical levels. By matching safeguards to steps in our attack tree, we make an important technical contribution to combating cheating at every stage of a cheater's process.

2 Related Work

There are several SLRs on cheating and countermeasures in online games. Early work developed taxonomies of cheating, both in general [162, 163] and specifically for MMOGs [55, 52]. Jeng and Lee refined prior classifications by broadening their scope, structure, and comprehensiveness [61]. Kotkov et al. presented an SLR on bot detection and identified underrepresented game categories, reviewed publicly available datasets, and presented a taxonomy of detection methods [76].

Next, attention shifted towards Peer-to-Peer (P2P) game architectures [38, 87]. Fan et al. defined six design issues to address P2P security for MMOGs [38]. They compared alternatives for each issue and evaluated how P2P architectures fulfill the criteria [38]. Ma and Oikonomou provided an overview of current MMOGs' network architectures, discussing various distributed Client–Server (C/S) solutions based on sharding, partitioning, and geographical dispersion of players [87]. Furthermore, Yahyavi and Kemme presented an overview of P2P solutions for MMOGs using a uniform terminology [159].

More recently, AI-based cheat detection has attracted growing attention, with multiple SLRs summarizing the field [130, 171, 56]. Spijkerman and Ehlers conducted an SLR and compared AI-driven tools with other classifiers for cheating detection. Others focused on the capabilities of machine learning (ML) identifying cheating behaviors in competitive gaming [130, 171]. Hu compared the advantages and limitations of AI image recognition cheating software with traditional memory injection cheating software. They analyzed how cheaters could use AI and discussed strategies to combat such cheating behaviors [56].

Additionally, researchers reviewed specific cheating topics. Woo and Kim examined academic research and industry practices about illegal activities in online games and their countermeasures. This includes botting, in-game hacks, gold farming, private servers, system/network hacking, and theft [156]. Chen et al. provided an overview of online cheating threats and the associated security flaws. They also examined consequences for online businesses and outlined technical implications [21]. Sablatnig et al. analyzed requirements to prevent cheating, detect fraud, and ensure provability [121].

While previous research offered diverse perspectives on specific fields of cheating, our work provides a technical, in-depth examination that connects technical requirements for cheats with potential safeguards. We unify prior work and

address the existing research gap by developing an attack tree that references established MITRE techniques. This approach enables developers and researchers to better identify pain points, apply impactful security concepts and to formally confirm the validity of their mitigation processes.

3 Methodology

To create our intermediate taxonomy and the attack tree, we conducted an SLR [18] of peer-reviewed publications on cheating and anti-cheating in games. We followed the steps defined by Kitchenham et al. for conducting SLRs for software engineering [73]. This includes (i) a systematic search of potential literature (Sec. 3.1), (ii) the pre-filtering of publications (Sec. 3.2), and (iii) the content analysis (Sec. 3.3). Further, we visualized our results as an attack tree (Sec. 3.4). Figure 1 provides an overview of our methodology.

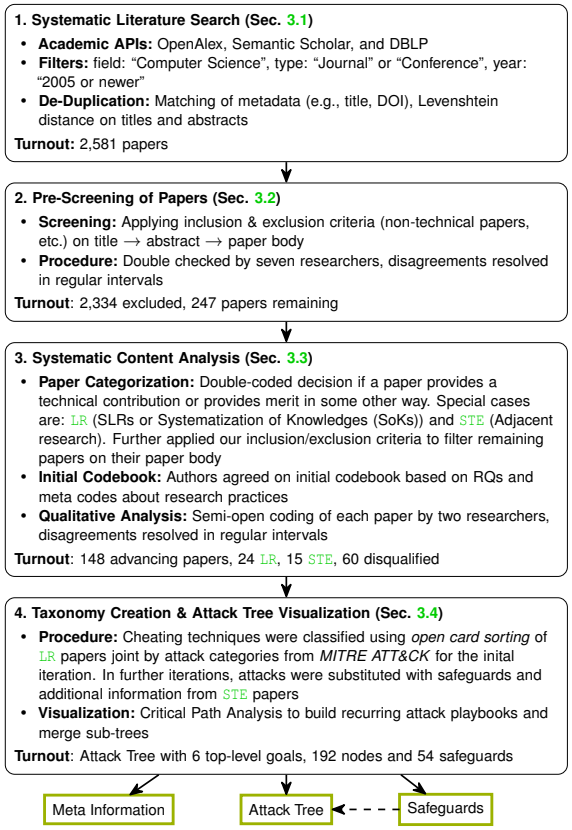


Figure 1: Overview of our multi-stage systematic analysis resulting in our attack tree, safeguards, and metadata.

3.1 Systematic Literature Search

In the following, we describe the pipeline to build our dataset. We repeated the search during our study to include the most recent work. The last crawl happened on December 31, 2025.

Search Query. We chose the anchor words *"game"* or *"gaming"* as a starting point and extended them with a combination of *"cheat"* or *"fraud."* Additionally, we added *"bot"* and *"detect"* to ensure we also covered botting detection. We used wildcards to enlarge strings, resulting in the following query:

```
(game* OR gaming) AND
((cheat* OR fraud*) OR
((bot*) AND (detect*)))
```

API Filters. We used the query to search in the titles and abstracts of papers using the APIs of Semantic Scholar [124], OpenAlex [105], and DBLP [32] to find relevant academic publications. We further applied three technical filters to our search. First, we only allowed papers classified by the APIs as belonging to the field of computer science, as we focus on a technical reflection on cheating. Second, we only considered papers being published in conferences or journals, as we focus on peer-reviewed works. Lastly, we used `publicationDate` to only include publications since 2005's taxonomy of cheating by Yan and Randell [162], covering 20 years of research.

Data Cleaning. As we noticed duplicates in our dataset, we added a de-duplication step to our pipeline. This included matching metadata (title, DOI, etc.) and using a generalized Levenshtein distance [169] as a similarity metric for titles and abstracts. For identified duplicates, the most recent version in the dataset was retained. For duplicates based on extended versions of a paper, we kept the most comprehensive version. We removed 15 duplicates in this step. Our dataset thus included 2,581 publications before pre-filtering.

3.2 Pre-Screening of Papers

Each paper was classified by two researchers whether it was relevant to our work or out of scope. We excluded non-English works. Further, we used the following exclusion criteria:

- **Social Sciences, Psychology, and Law:** We excluded papers that mainly covered psychological, social, or legal aspects of cheating.
- **Game Theory:** Our query returned several theoretical studies on game theory without technical contributions, which we excluded.
- **Financial Fraud:** *Chargeback frauds* [153] and *money laundry* [152, 78] (including *gold farming groups* and *real money trading*) are present topics in the gaming ecosystem, but we consider these as advantages resulting from cheating or exploiting external systems.
- **Special Scenarios:** We excluded screen-cheating in (co-located) split-screen games, serious or training games, capture-the-flag, hack-the-box, or other contests in a gamified environment, and pure social network games.
- **Other Gaming-related Issues:** We excluded studies on boosting, smurfing, griefing, and other toxic behavior.

For each paper, we applied our criteria in the following order: title, abstract, and paper body, depending on when a decision could be made. Conflicting decisions between the two researchers were discussed and resolved. We provide a list of all papers and decisions in our supplementary material (Open Science), including when a paper was disqualified. We removed 2,376 papers in this step, leaving 247 for analysis.

3.3 Systematic Content Analysis

Initially, each paper was read by two researchers and then jointly classified as either a technical contribution for the attack tree or as one of the following special cases:

- **Literature Review (`LR`)** (24) are SLR or SoK papers that provide summaries of the concept of cheating or anti-cheating. We used these papers to initially structure our attack tree (Sec. 3.4).
- **Surrounding Technical Environment (`STE`)** (15) papers do not directly cover attack scenarios or technical safeguards but extend specific aspects, e.g., explainable AI for existing bot detection models [141]. We added them to applicable nodes for further information but neither use them for creating the taxonomy nor the tree.
- We **disqualified** 60 papers during this step based on our exclusion criteria, some of which proposed full architectures for video games and only briefly addressed cheating (Sec. 3.2).

The 187 not disqualified publications were assigned to and systematically double-coded by seven researchers using the analysis software MAXQDA [91] and a semi-open approach [34] based on thematic analysis [25, 26]. To structure our codebook, we defined four top-level code groups *a priori* to organize codes regarding data about the examined cheating attack scenario, context regarding applicability dimensions, presented safeguards, and any meta-information.

We then continued to fill the codebook with results from the papers in scope. To obtain a holistic representation of cheating scenarios, we added codes for the goal of the cheat, e.g., utilizing aim assistance in shooter games, what technical component the cheat is attacking, and what the cheat may further require from a cheater, e.g., reverse-engineering a component. Further, we added codes to assign cheats or safeguards to certain genre- or platform-specific subgroups of games for which they are applicable or with which they were introduced, e.g., first-person shooters (FPSs) or architecture types such as P2P or C/S. For safeguards, we added codes for how the actual mechanism works, where it is implemented architecturally, if there are any prerequisites, e.g., log data, kernel access, cryptographic material, etc., and what security control type(s) [157] the safeguard can be classified as. Lastly, we added meta-codes about, e.g., how the paper provides advice and any miscellaneous general observations. Each pair of researchers compared their resulting coding of papers, resolved any conflicts, and merged new papers into the main codebook. Our final codebook is available in our supplementary materials (Open Science).

3.4 Taxonomy & Attack Tree Construction

To facilitate threat modeling and provide game developers with structured guidance, we visualized our intermediate taxonomy as an attack tree [123] following the process described by *MITRE ATT&CK* [43].

We developed the attack tree from an attacker-centric cheater view with the goal of gaining any unfair advantage over honest players. First, we leveraged *open card sorting* [132] with the contents of the 24 `LR` papers influencing the top-level primary exploitation categories of the attack tree (Open Science). Next, we iteratively added the identified cheating patterns from the technical contributions recorded in our codebook to the skeleton tree, where each cheat was structured as a bottom-up attack tree. In the last step, we performed a critical path analysis to identify common components within the branches and formulated these as seperate *playbooks*, supported by categories from the *MITRE ATT&CK Matrix for Enterprise* [42] where appropriate. We used Miro [95] boards for translating the contents of our codebook into the tree.

Ultimately, we analyzed the safeguards from a game developer's perspective. The attack tree was used as a basis to determine what a game developer wants to prohibit or avoid in their game. For each cheat, we placed all identified safeguard mechanisms and requirements on a separate Miro board and, considering the objective of preventing a certain cheat in the game, also performed closed card sorting with three researchers. The result of this analysis step was the final attack tree containing all identified cheating types with the variety of safeguards that aim to prevent or detect a cheat.

3.5 Limitations

Our dataset depends primarily on the discoverability and retrievability of the publications. To minimize this bias and reduce the likelihood of missing publications, we used three different services and generally employed a rather inclusive search query. Furthermore, we limited ourselves to peer-reviewed publications from the last 20 years. This time range was chosen to base our findings on established foundations of research published in 2005 by Yan and Randell [162], including the latest research and emerging topics. A brief search revealed that older publications are either no longer accessible or too outdated, especially considering that MMOGs are a relatively young phenomenon that only emerged in the early 2000s.We are therefore confident that our dataset contains all relevant as well as the latest findings.

Furthermore, our work is limited to the academic discussion of cheating and excludes gray literature or industry reports, resulting in potential blind spots. However, our dataset

includes studies on real cheating attacks and industry research; thus, we are confident that the attack tree does not contain major blind spots. Moreover, by using *MITRE ATT&ACK* [42], we were able to identify several underlying technical similarities that presumably also apply to left-out cheats.

4 Cheating Attack Tree

Due to space constraints, we are unable to include the entire attack tree; instead, we present a scaled-down version in Figure 2, of which we will present the six top-level nodes in the following sections. We provide a complete, scalable version in our supplementary materials (Open Science).

For referencing, we assigned all nodes a **C**heating **A**ttack **T**ree identifier in the form of **CAT**-<*Number*>. The ultimate goal of an attacker is to gain an unfair advantage over others in a video game, characterized by unfair behavior (CAT-000). When describing the major branches of the attack tree, we further adhere to the vocabulary from the MITRE ATT&CK TTP framework [136]. We use the term "tactic" for higher-level nodes, such as the use of automation to gain an advantage (botting). "Techniques" are sub-nodes of a tactic, such as different ways to realize the execution of a bot. Specific attack sequences from top to bottom, e.g., a bot using object detection for navigation requiring input, are referred to as "procedures." We modeled 192 nodes which we refer to as "steps" of an attacker. The attack tree connects sub-steps with a logical *OR* by default [123], implying that an attacker often has several options to achieve their goal.

Playbooks. In line with the MITRE guidelines for creating attack trees [43], we have grouped several recurring techniques involving the same steps into sub-trees, which we refer to as *playbooks*, each with their own `PB`-ID (Appendix A). Playbooks showed apparent patterns that enabled grouping, e.g., the hijacking of a process, and we were able to trace these back to attack patterns listed by MITRE [42].

4.1 Automation & Assistance

The most thoroughly researched issue concerns automation in games. We divided this tactic into two sub-tactics. First, fully automated systems handle all the gameplay without a player necessarily present. Second, assisting systems are only partly automated while a player still has to play.

Fully Automated Systems. So-called *bots* play a game autonomously to achieve any goal in the game that can be automated, generally performing repetitive tasks that a benign player would normally have to perform with sufficient diligence (CAT-100). We have encountered this attack tactic particularly frequently in connection with massively multiplayer online role-playing games (MMORPGs), where players achieve progress through levels or items.

We modeled four key *AND* sub-steps as requirements for any botting attacker we identified. The goal of the first step is to gather information about the surroundings, which effectively supports performing the controls, e.g., the avatar's coordinates. Typical techniques cover writing or recording simple macros, e.g., certain paths that a bot repeatedly traverses for a task [102, 23]. More reactionary techniques include dynamic memory reads of a game process to obtain live in-game information, e.g., other characters [47, 97]. Similarly, an attacker can use a debugger attached to a game process to examine memory data from a target's address space [93, 39]. Another approach is to read network packets; this may necessitate the decryption and reverse engineering of the game's packet structure, to extract intelligence [145]. Further, the graphics of a game can be utilized by either capturing information using OS graphics APIs or employing a real-time ML object detection pipeline, such as the You Only Look Once (YOLO) series of convolutional neural networks (CNNs), trained on the in-game view such as screenshots [118, 60]. Regarding further OS functionalities, an attacker may also utilize information from the window messaging system [94, 84].

Next, the logic of a bot needs to be executed. This can happen outside a game process by sending keyboard or mouse events to OS I/O API functions [39] or sending OS window messages with inputs [84]. Another way includes to manipulate the game process, either by calling functions in the game code directly to control the avatar, or by inserting code to be executed [47]. However, such a procedure requires reverse engineering the game's process and developing a precise technique for executing inserted controls. Sophisticated ways to insert and execute code may also leave the user-mode and manipulate kernel or hardware features [39].

Another major step concerns operating the bot without being detected. In general, a bot is interested in simulating human behavior to evade detection mechanisms based on behavioral data, e.g., through anomaly detection [79]. We have found that a bot can mimic behavior either (i) through adjusting avatar behavior, e.g., varying in walked paths [97], (ii) through operational changes, such as not being online all the time [10], or (iii) through methodological measures, by, e.g., hiding from process scans if any chosen technique involved interfacing with the game process or memory [39].

Finally, for a bot program to work, a cross-sectional step concerns scripting. From obtaining information, to processing, to the actual execution of the desired actions, "glue code" is needed to translate the logic of a bot into the control system, e.g., in the technique of using ML object detection and controlling the bot accordingly to interact with objects, i.e., to send an input. This step may also include additional static configurations that customize the functions of a bot [47, 113].

Player-Assisting Systems. Automation does not have to be fully autonomous; it can also be used to assist a real player (CAT-200). Predominantly, we identified so-called *aimbots* and their various forms (i.e., triggerbots, aimkeys, etc. [166, 3, 22, 27, 86]). The assistance further includes tactics based on semi-automatic inputs, including the use of *macros* [39],

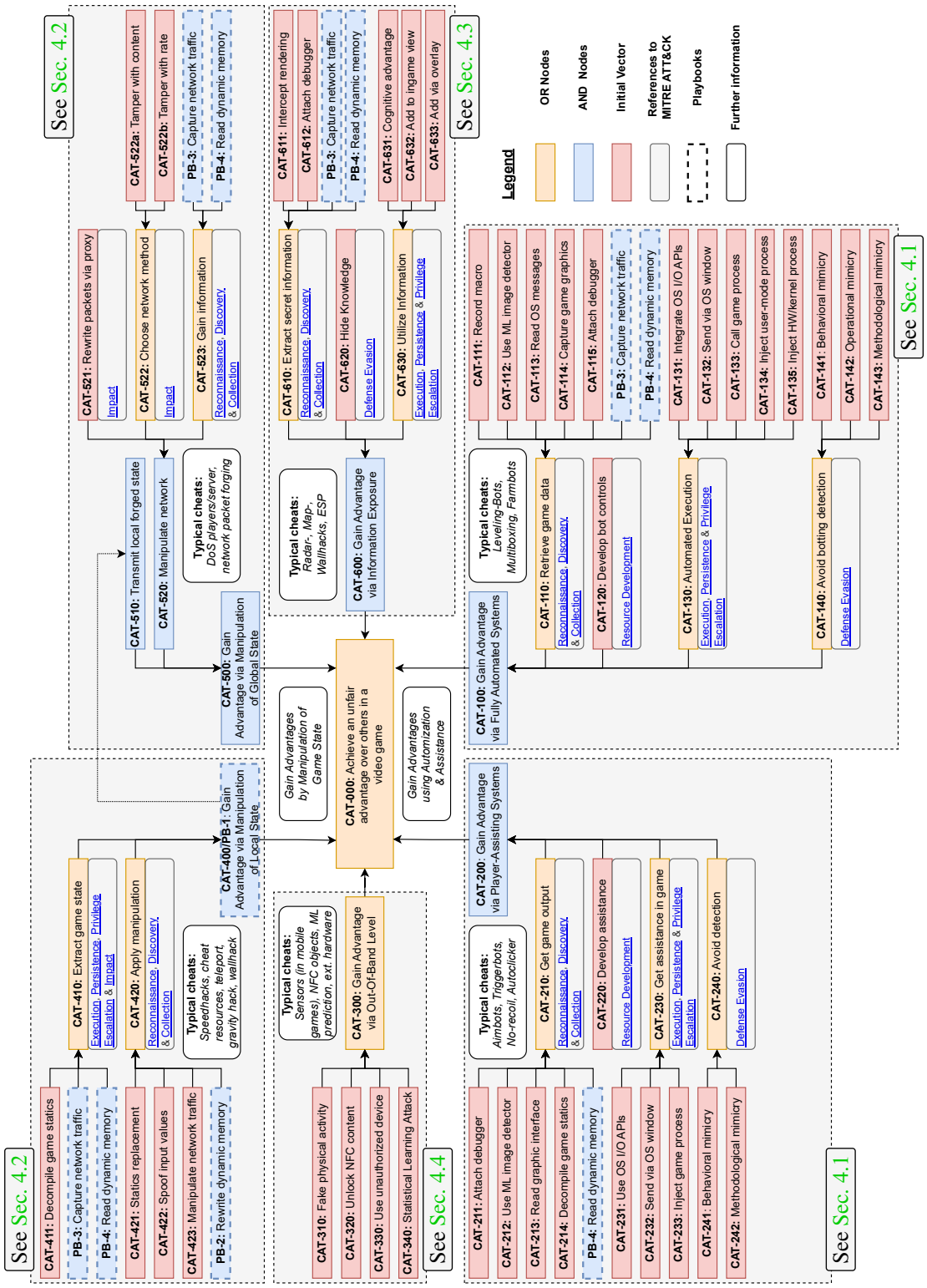


Figure 2: This figure illustrates a hierarchical classification of attacks to achieve unfair advantages in video games (CAT-000). The tree is organized into six primary exploitation categories: *Fully Automated Systems* (CAT-100), *Player-Assisting Systems* (CAT-200), *Out-of-Band Level* (CAT-300), *Manipulation of Local State* (CAT-400), *Manipulation of Global State* (CAT-500), and *Information Exposure* (CAT-600). We map each category to the MITRE ATT&CK framework (e.g., Reconnaissance, Defense Evasion, Persistence) to align game-specific threats with established cybersecurity domain knowledge.

auto-clickers [40, 62], or *spinbots* [120], simulating fast key or mouse movements. This can be leveraged, e.g., for *no recoil* cheating in shooter games [165]. The close similarities to fully automated systems are reflected in our attack tree in that both share the same four key *AND* sub-steps, and MITRE references. In the following, we focus on the special features of assisted play and refer the reader to the previous section for detailed information on similar techniques (Sec. 4.1).

Initially, access to game information is required, e.g., for a generic aimbot, to detect opponents' presence near the player's avatar. To solve this problem, an attacker can use several techniques analogous to information capturing for automated play. Options include reading the OS graphics device APIs for information on pixel colors, as Laurens et al. noted: *"An example of such a hack involved changing the player model skins in the game so that all enemies would appear flat-shaded red, the external hack would then monitor the output of the graphics driver, constantly scanning for red-pixels"* [81]. This procedure shows that some attacks leverage a combination of several tactics and techniques in our attack tree, combining the replacement of a player model skin, covered in the tactic of manipulation of the local state (Sec. 4.2), with assisted play. Other techniques use ML-based object detection, with captured live material, e.g., by using *Open Broadcaster Software (OBS)* [116] or *NVIDIA ShadowPlay* [103], as discussed by Yang et al. [165]. Moreover, attackers can leverage static analysis of available code to find useful segments [27] or use dynamic memory reads [165], e.g., attaching a debugger to the game process [39].

Next, the attacker must execute code or a script [85]. This can be done again by sending keyboard or mouse events to OS I/O API functions [101], sending OS window messages with movement inputs, or manipulating the game process [86].

Moreover, attackers must operate covertly without being detected. In terms of methodological mimicry, we found that the cheat software process can be obscured through seemingly harmless naming of the executable files [77]. Further, assuming screen recordings for object detection, capture cards can be utilized to remove parts of the pipeline from the client and can therefore no longer be detected by, e.g., process scans [165]. In terms of avatar behavior mimicry, Kanervisto et al. proposed generative adversarial networks (GANs) trained on benign human gameplay recordings to smooth out aim assistance in shooter games by, e.g., providing automatic movement input in a more human-like speed trajectory or by slightly overshooting a target [66]. Finally, an attacker may blend in between benign players by irregularly turning on and off the assistance, displaying both human and bot-like behaviors [168, 166, 167].

4.2 Game State Manipulation

Another tactic concerns game state manipulation, i.e., an attacker actively tries to influence the game state to their advantage. Baughman and Levine defined the game state as *"the set of information needed to describe the game at any time, [...] composed of entity states"* [7]. Technically, this corresponds to a collection of concrete *data values* that represent the current situation of the world and its entities, e.g., scores [144], positions [11], resources [147] or item inventory [138], *rule parameters*, e.g., speed constants [58] used by a physics engine, and client-side artifacts, e.g., *game files* such as 3D models to render the display state [6]. Modifying the latter, e.g., replacing wall assets to make walls transparent for a wall hack [65] or increasing the volume of certain sound assets [81], changes the player's game state locally. By dividing this attack into two sub-tactics, i.e., changes to the local game state, followed by the dissemination and replication of a corrupted global game state, the actual steps taken by an attacker can be tracked in detail through our attack tree. The subsequent steps also enabled us to attach differentiated safeguards. We based our approach on a similar distinction made by Park et al. between local and global game states [108].

Manipulating the Local State. For the manipulation of the local state (CAT-400), we set two necessary *AND* sub-steps. First, an attacker needs information about a game's state. They can obtain this through dynamic memory reads, e.g., by identifying memory locations for values such as experience points, in-game currency, or other resources associated with the game [147]. An attacker can also obtain information from the network by intercepting and examining packets [144]. Another attack vector concerns static files and code [39] if, e.g., constant values for a physics engine are fed from static values in configurations and are tampered with for speed-, movement-, or teleportation-hacks that manipulate physics such as gravity, acceleration, or the location of an avatar, e.g., in racing or role-playing games (RPGs) [114].

There are various methods to manipulate a value. Dynamic memory writes can be used to overwrite memory with desired values [147]. Similarly, static files can be replaced [39]. At the network level, a proxy instance can intercept packets, rewriting them or creating new ones, and sending them to the client, impersonating the server entity and state changes, thereby manipulating the local game state on the client by tricking it, e.g., to unlock paid content [144]. Finally, the local game state can also be influenced by blocking game state updates, ultimately filtering or blocking incoming packets [51], e.g., by switching off the network adapter [160].

Manipulating the Global Game State. Manipulating the global game state is more complex, as it entails persuading other clients or servers to accept forged states and values or respond in a manner beneficial to the attacker (CAT-500). In this respect, so-called *protocol-level cheating* often takes place at the network level [64] and depends on how receivers handle manipulated states [126]. We differentiate whether manipulation has occurred at the local level and is merely being propagated or whether direct manipulation is taking place at the network level. For the latter, we have added

three essential *AND* sub-steps for an attacker. Initially, an attacker needs to access information about the game state by using techniques from Sec. 4.2. Interfering directly with global game states further requires man-in-the-middle (MitM) techniques to modify or forge packets [58, 99].

Next, we identified two options to meddle with the state on a protocol level, either by manipulating the contents of a packet or by tampering with the packet rate. For the former, almost all options for modifying a packet are open to the attacker. Among these, we found modifications to the state update packet that are technically correct but semantically false, i.e., impossible state transitions to be particularly common in related work [158]. Further, packets can also be manipulated technically: senders and recipients can be spoofed, and packets can be timestamped incorrectly so tampered packets reach a receiver earlier [127, 29].

Regarding the sent event packet rate, often coined as *timing cheating*, an attacker can try to increase the speed of sending packets [41], which could simulate faster inputs than are humanly possible, similar to *auto-clicker* cheats [40]. Further, flooding with event packets can also be used to, e.g., overload a game server during a denial-of-service (DoS) attack [158]. Packets can also be replayed, e.g., to trick an authentication instance into unlocking content for a client multiple times, such as in an in-game shop [144]. However, an attacker can also try to slow down the sent event packet rate, simulate high client latency (*lag-switches* [62]), or stop sending state updates completely, enabling an attacker to carry out procedures such as so-called *lookahead* cheating by waiting for the event update packets of all other participants and adjusting their strategy based on the decisions of everyone else [8]. With so-called dead reckoning, a client technology that is designed to compensate for latency issues experienced by other participants in MMOGs by extrapolating user inputs, so-called suppress-correct cheating can occur. By deliberately suppressing state update packets, dead reckoning is triggered, but the attacker then sends packets more advantageous to them [55].

4.3 Accessing Unauthorized Information

Another major tactic for attackers is to access and utilize additional information from the game state (CAT-600). This tactic is related to what is frequently discussed as *information exposure*, i.e., a client has more information about the game state than is necessary to create a display state after the rendering process [53, 81]. In this respect, an attacker using this tactic does not attack the game state but the generation of the display state from the game state by intervening in the generation process and inserting additional information or manipulating existing functions that create the display state. Commonly, such an attack tactic can be leveraged in real-time strategy (RTS) games, e.g., as *map hacks* [20], or in shooter games as *wall hacks* [108].

We identified three top-level *AND* sub-steps for this tactic. In general, an attacker must gain access to data from the game state. Relevant data, e.g., for extrasensory perception (ESP), can be obtained via dynamic memory reads, containing information about other players, such as their position or health points [27, 63]. Using a debugger to connect to the game process can provide an attacker with additional information [39]. Further, data can also be obtained by intercepting and analyzing network packets [175]. Moreover, it is possible for an attacker to reverse engineer and intercept parts of the rendering pipeline between the game process, GPU, and CPU to identify information sources [108].

In the second step, an attacker uses this information, whereby they have two options for displaying the information obtained. On the one hand, the information can be embedded in a display overlay. Such an overlay is located "over" the game process and therefore does not need to interact directly with it. This technique requires scripting to create the window and populate it with the information [175]. On the other hand, the information can also be embedded directly in the game view by interfering with the creation of the display state. This requires an execution handle or hook and also the insertion of code into the game process, e.g., inserting a radar system [46] or extending an existing radar [27]. In addition, the attacker has several options for manipulating the display state creation. We found possible techniques such as manipulation of the GPU driver or interference in the communication between the CPU and GPU [108], but also through manipulation of functions at the game engine level [88] or manipulation of memory-mapped assets used for the display state [39]. In a specific side case, reading the information may also be sufficient; the attacker gains a cognitive advantage, e.g., to be able to make better-informed moves than other players [20].

Lastly, an attacker must operate without being detected.

4.4 Out-of-Band Cheating

Lastly, we identified a tactic utilizing techniques from an out-of-band level (CAT-300). In these cases, an attacker uses additional external aids unrelated to the game or exploits weaknesses arising from the fact that a game or client device is located in a manipulable environment outside the game's control. Such an advantage can arise by connecting additional hardware to a mobile device that makes the game easier for them, e.g., a controller adapter, which makes an attacker's inputs more precise than those of their fellow players [5].

Particularly in location-based games on mobile devices, sensors can also be deceived if the game logic relies on their inputs. This can be done, e.g., by mounting a mobile device that requires distance traveled on other physically moving objects, such as a ceiling fan [107].

External clients can also be used, allowing an attacker to perform statistical learning on them to predict the probabilities of their opponent's future moves. This tactic was

presented by Bursztein and occurs in games that allow such an approach because they are predictable and also allow sufficient time between moves for the model to make calculations, e.g., in trading or collectible card games [16]. However, a ML model that is supposed to deliver predictions requires input about the current game state. Bursztein explained that this can be achieved by using a game's official APIs or by other means, such as recording and analyzing network traffic. A client can also be manipulated directly, e.g., by inserting code that can be used to read out the game state. Furthermore, a ML object detection pipeline may also be feasible, which can read the game state from the player's view, e.g., which playing cards are currently in play. However, the game client can also be manipulated to activate additional functions that enable access, e.g., a debug log for developers. Ultimately, the player's inputs must be made, which can be automated but can also be done manually after the attacker has received a new prediction and adjusted their strategy accordingly.

Finally, we found another tactic that relies on external near field communication (NFC) objects. In cases where a game can interact with such a device, e.g., to unlock game content, the NFC object itself, reader communication, the reading software, and the transmission between the reader and the game, as well as to a prospective server, provide opportunities for attacks. These include object duplications, changes to firmware and software, or MitM attacks during data transmission [44].

5 Safeguards

In this section, we first introduce requirements for safeguards and then present the safeguards that emerged in our SLR.

5.1 Requirements of Safeguards

We categorized requirements into the 16 types shown in Table 1, and assigned IDs to match requirements and safeguards.

Table 1: Requirement Categories with IDs.

ID	Requirement	Description and Examples
R01	Kernel-Level Access	Extended privileges on client
R02	Cryptographic Key Material	Encryption, hashing, signing
R03	Trusted Execution Environment	Intel SGX, AMD SEV, ARM TrustZone
R04	Game Replays	Log data or recordings of past sessions
R05	Hardware ID (HWID)	A fixed player hardware handle, e.g., serial numbers
R06	Update Channel	For example to obtain blacklist signatures
R07	Flagging of Players	Storage system for holding flags, e.g., database
R08	Reputation System	System for scoring and holding a player reputation vector
R09	Labeling Data	Prerequisites for (some) ML analysis
R10	Monitoring System	System to obtain log data
	Log Data Sub-Types	
R11	Meta Player Information	IP address, payment information, etc.
R12	Meta Gaming Data	Session information, frequencies, etc.
R13	Technical Data	Raw device inputs (keyboard/mouse), etc.
R14	Game-related Data	Avatar actions, movement, etc.
R15	Avatar Information	Avatar values, health, experience points, etc.
R16	Social Interactions	Communication, activities, trades, etc.

A common requirement for safeguards is the need for extended privileges for kernel-level access (R01). This allows, e.g., critical memory areas to be monitored in client-side scanning. Moreover, cryptographic key material (R02) is necessary for techniques like encrypted network traffic. Using a trusted execution environment (TEE) further requires capable hardware (R03). For several corrective measures, it may also be necessary to have hardware IDs on players to execute more severe corrective hardware bans (R05), or to flag a player (R07) to feed a reputation system (R08), which is used in various safeguards.

Monitoring & Log Data. Many approaches dealt with the analysis of log data, often in connection with statistical or ML methods, to detect anomalies in player behavior that imply cheating. Anomaly detection requires different types of log data, as well as a monitoring system (R10) to capture required data, or, complementary, access to replay data (R04), e.g., for training purposes. Depending on the chosen machine learning paradigm, it may also be necessary to label this data (R09). We defined six different log data types (R11–R16) across game genres and safeguards. Below, we briefly present each type with non-exhaustive examples. A complete list of all data types is provided through our codebook export in our supplementary materials (Open Science).

In the area of meta player information (R11), we have classified the client's connectivity data. This includes the IP address, various devices used by the same player, other connection data such as the use of a VPN or PPTP, additional available metadata from the network provider, and any higher-level player account statuses such as "VIP" or similar. Meta gaming data (R12) includes everything related to the player's sessions, such as timestamps, play time, or how often a player logs in and out. Technical data (R13) refers to inputs received by the game, such as mouse actions, keyboard presses, or periods of absence of inputs. Game-related data (R14) covers all interactions that an avatar has performed and experienced, such as movement and location data, firing or getting hit in shooter games, or item and world interactions in RPGs. However, this category also includes all data closely related to the game itself, such as which map was played in a shooter or the ID of the server. Separately, we have distinguished between static avatar information (R15), such as data from a scoreboard, scores, levels, classes, or acquired skills in RPGs. Finally, literature often dealt with the analysis of interactions between players, which we included in social interactions (R16). We included data such as direct communications, but also group activities such as joint questing or trading.

5.2 Safeguard Techniques

We identified 54 safeguards against the various cheating tactics from the attack tree and provide an overview in Table 2. We classified them into four control types: *directive*, *preventive*, *detective*, and *corrective* (as defined by Wright [157]). Further, we marked the deployment location, possible requirements for their usage, and identifiers to attack steps from our attack tree addressed by a safeguard.

Table 2: Identified Safeguards with IDs, Control Types (**Di**rective, **Pr**eventive, **De**tective and **Co**rrective), their **Loc**ation, specific requirements and our referenced attack scenarios.

ID	Safeguard	Di.	Pr.	De.	Co.	Loc.	Requirements	Arch.	Addressed CATs
SG01	Transparent game policies about cheating	x				C,S		General	
SG02	Educate players about consequences and repercussions	x				C,S		General	
SG03	CAPTCHAs		x			C		General	CAT-100
SG04	Disincenting game design		x			C		General	CAT-100,CAT-310,CAT-522.2.2,CAT-522.2.3
SG05	Player judgement feature			x		C,S	R04	General	CAT-200,CAT-600
SG06	Log Analysis (ML)			x		C,S	R10,R11-R16,(R04),(R09)	General	CAT-100,CAT-200,CAT-600
SG07	Log Analysis (Statistics)			x		C,S	R10,R11-R16,(R04),(R09)	General	CAT-100,CAT-200,CAT-600
SG08	Integrity checks: Files, binary, code, registry, etc.		x	x		C		General	CAT-421
SG09	Integrity checks: Loaded game process modules		x	x		C		General	PB-6
SG10	Integrity checks: Dynamic memory and stack validation		x	x		C		General	PB-2
SG11	Integrity checks: Function pointer validation		x	x		C		General	PB-5
SG12	Integrity checks: Drivers, e.g. graphics		x	x		C	(R01)	General	CAT-143.1,CAT-422,CAT-423.2.2,CAT-632.3.3
SG13	Integrity checks: Kernel-space		x	x		C	R01	General	PB-2,PB-6
SG14	Scans: Changes to registers			x		C	R01	General	CAT-135
SG15	Scans: Changes to instruction / exception / interrupt counts			x		C	R01	General	CAT-135
SG16	Scans: Changes to EIP behavior			x		C	R01	General	CAT-135
SG17	Scans: Loaded game process modules			x		C,(S)	(R06)	General	PB-6
SG18	Scans: (New) Processes, handles, threads; block			x		C,(S)	(R01),(R06)	General	PB-2.2,PB-3.1,PB-4.2,PB-6.2.2,PB-6.2.3
SG19	Scans: Dynamic memory			x		C,(S)	(R01),(R06)	General	PB-2,PB-6
SG20	Scans: Drivers, e.g. graphics			x		C,(S)	(R01),(R06)	General	CAT-143.1,CAT-422,CAT-423.2.2,CAT-632.3.3
SG21	Scans: Kernel-space			x		C,(S)	R01,(R06)	General	PB-2,PB-6
SG22	Scans: Open network connections			x		C	(R01)	General	CAT-120,CAT-220,CAT-632.1,CAT-633.2
SG23	Scans: Measuring of changing execution timings / game ticks			x		C		General	PB-6
SG24	Scans: Check for callback or hook techniques, process misdirects		x	x		C	(R01)	General	PB-5
SG25	Use honeypots for game APIs			x		S,(C)		General	
SG26	Isolate cheaters in matchmaking				x	C,S	R07	General	
SG27	Terminate cheaters' game				x	C		General	
SG28	Visual flagging of cheater				x	C,S	R07	General	
SG29	Public reputation system				x	S	R08	General	
SG30	Kick cheaters				x	C		General	
SG31	Ban cheaters (Account)				x	C		General	
SG32	Ban cheaters (HWID)				x	C	R05	General	
SG33	Adding noise to display state (Adversarial ML)		x			C		General	CAT-212,(CAT-112)
SG34	Apply obfuscation on statics, source code		x			C		General	CAT-214,CAT-411,(CAT-421)
SG35	Apply obfuscation on dynamic execution flow, source / byte code		x			C		General	PB-5,PB-6
SG36	Apply obfuscation on dynamic memory		x			C		General	PB-2,PB-4,PB-6
SG37	Protect memory region		x	x		C	(R01)	General	PB-2,PB-4,PB-6
SG38	Debugger prevention from attaching		x	x		C		General	CAT-115,CAT-211,CAT-612,PB-5.3
SG39	Debugger detection			x		C	(R01)	General	CAT-115,CAT-211,CAT-612,PB-5.3
SG40	Outsourcing files / memory / GPU rendering into TEE		x			C	R03	General	PB-2,PB-4,PB-5,PB-6
SG41	Check if being executed in VM			x		C	(R01)	General	CAT-143.3
SG42	Compare I/O HW signals with game engine			x		C,(S)		General	CAT-131,CAT-231
SG43	Block MS Windows drivers test signing mode		x	x		C		Windows	CAT-143.1,CAT-422,CAT-423.2.2,CAT-632.3.3
SG44	DB Integrity checks in backend		x			S		General	CAT-510
SG45	Encryption of network traffic / state / event		x			C,S	R02	General	PB-3,CAT-324,CAT-323.2
SG46	Message hashing of network traffic / state / event		x	x		C,S	R02	General	PB-3,CAT-324,CAT-323.2
SG47	Message signing of network traffic / state / event / hash		x	x		C,S	R02	General	PB-3,CAT-324,CAT-323.2,CAT-522.2.1,CAT-522.1.3
SG48	Periodically change key material		x			C,S	R02	General	
SG49	Authoritative server design		x	x		S		General	CAT-400,CAT-510,CAT-522.1.1
SG50	Area of interest region partitioning, zoning; information minimization		x			C,S		General	CAT-410,CAT-523,CAT-610,(PB-3),(PB-4)
SG51	Monitor node, referee, coordinator, trusted entity, proxy		x	x		C	(R02)	P2P	CAT-400,CAT-510,CAT-522.1.1
SG52	Use event (commitment) protocols		x	x		C	R02	P2P	CAT-522.1.4,CAT-522.2.1,CAT-522.2.3,CAT-522.2.4.1,CAT-522.2.4.3
SG53	Root/Jailbreak detection		x	x		C,S		Mobile	
SG54	Use of additional sensor/meta data		x	x		C,S		Mobile	CAT-143.1,CAT-310,CAT-422

Hardening the Client. A lot of the game state handling happens on the client level, but the client cannot simply be trusted, as it offers many points of attack. After all, however, a game must be run on a client to be played. Thus, to make a client as resistant as possible, we collected 32 safeguards regarding blacklisting, whitelisting, scanning, and hardening.

First, we found many integrity measures that ensure that both the static game data in advance and the dynamic memory area have not been manipulated (SG08–SG13) [39, 27]. In addition, various blacklist scans can be used to examine dynamic areas, detect tampering with the game process, or detect malicious activity on the client outside the game process, e.g., by tracking open network connections to known cheating hosts (SG14–SG24) [36]. Finally, various methods can be implemented at the code, process, and network levels to increase difficulty accessing critical entry points. These include obfuscating code, execution flow or memory, outsourcing game data to TEEs or the encryption, hashing, and signing of network data (SG34–SG48). Tian et al. noted that: *"If done effectively, code obfuscation and hiding can be a sufficient (though not perfect) deterrent to game hackers who try to reverse engineer the game*" [144]. The scope of these safeguards can also be expanded by (i) implementing measures at a lower kernel level, (ii) starting anti-cheat measures before the game is launched, or (iii) starting the game from the safeguard process as its parent with providing permissions.

Anomaly Detection & ML Threats. Many publications covered anomaly detection to find cheaters. We distinguished between papers using statistical methods for outlier detection [68, 96, 106] and those using ML approaches. We identified a wide range of ML approaches, including supervised, unsupervised, and reinforcement learning. A large part discussed variants of deep learning, including recurrent (RNN) [142], convolutional (CNN) [110], graph (GNN) [117], or cascade correlation (CCNN) [47] neural networks. Unsupervised learning was proposed using latent Dirichlet allocation (LDA) [173], among other things, while reinforcement learning included genetic algorithms [115]. A detailed list can be found with our codebook (Open Science).

Anomaly detection was often linked to the need for a

dataset. Yahyavi et al. noted: *"No game research framework can realistically emulate commercial multiplayer games, which take years and millions of dollars to develop*" [158]. To simulate data as realistically as possible, some researchers used cheating software themselves in controlled environments, both self-written [46] and "real" cheats [47]. Some researchers invited participants to generate datasets, both with and without cheats, especially for FPSs [166, 66]. Generating human-like datasets was also the largest use case for human subject studies we found overall. However, especially for MMORPGs, we found several cooperations with video game companies, which in turn investigated attacks from the field of fully automated play with data from their products [72, 174, 69]. Sometimes, reference was also made to the privacy conformity of using this data [10, 67, 9]. Labeling-wise, reliance was placed on the previous classification of the data by the company [17, 174, 96]. Platzer conducted a human subjects study to generate data on MMORPGs for research, but reported that it was *"a time-consuming task*" and that *"it does not pay of to gather player data by setting up a homebrew game server*" [113]. Others used public APIs to obtain game replays, including Steam [106]. Further, a few papers also examined artificial or synthetic data [134, 83].

Unsupervised methods were used mainly because, as Choi and Ko commented, *"we simply need to gather benign data, which are far easier to obtain than cheating data – cheaters inherently do not want to reveal their identity, thus hiring them to collect real-world cheating data is not feasible*" [22].

On the other hand, to defend against ML object detection-based attacks, adversarial ML proposed to insert noise into game graphics assets (SG33), which are becoming increasingly common due to ever improving object detection techniques posing a threat [137, 89, 101].

Architectural Considerations. Besides security and fairness, game architectures must also take other non-functional requirements into account. These include latency and scalability, especially in MMORPGs, where many clients must be connected [159]. This was reflected in our dataset by research on P2P and distributed state synchronization [158]. However, the issue of security and cheating is more complex when there is no authoritative server (SG49) [144] to oversee and decide on the game state. To this end, approaches proposed partitioning the game world into regions (SG50), e.g., so that not every client is provided with information from all players, but only what is necessary to render their local display state [53]. A certain return to server authoritative design can be found in proposed monitor nodes, referees, trusted coordinators, or proxies (SG51) [154] and the associated challenge of selecting a trustful node for this task [150]. Further, we found many elaborate protocols, including those to ensure event commitment for client actions (SG52) [155], mainly based on using encryption (SG45), hashing (SG46) and signing (SG47) of and for game state network traffic [112]. The relevant literature was published mainly between 2005 and 2010. In later work, hybrid approaches were increasingly proposed [129].

Many safeguards are also applicable to mobile environments. They also offer further possibilities, such as additional sensors that might be used for safeguards (SG54) [160], but especially Android devices are affected by rooting (SG53) [79].

Anti-Cheating by Game Design. Some papers proposed incorporating safeguards into the game design. Since using CAPTCHAs as an approach to detect botting was identified as having a significant impact on the gaming experience due to interupting gameplay, it was proposed that a CAPTCHA could be incorporated more organically into a game design, e.g., as a mini-game (SG03) [24]. Moreover, considerations for cheating-disincentivizing game designs were proposed by identifying which areas are particularly vulnerable to attacks, such as a player-driven in-game auction house, and embedding noninteractive tests (SG04) [161].

6 Discussion

Video games are under constant attack. Regarding cheating, the many references to the *arms race* [2] make it clear that this issue demands an ongoing effort and attention. Today's video games are diverse, as are potential cheating attack vectors; developers need to think carefully early on about what their game design includes and what types of cheating could occur. However, the reality of the gaming industry often does not allow for such considerations [100, 75]. Many identified attack tactics are technically complex and lie far outside the scope of developing a game's core content, while there is a lack of detailed technical descriptions of cheating to understand attacks, develop safeguards, and, most importantly, apply them. Our work contributes to shifting the boundaries of this arms race in favor of developers and fairness in video games. In the following, we present trends in cheating and related research, examine key attack and defense mechanisms, discuss the challenges involved in researching anti-cheating measures, and highlight future research directions.

6.1 Trends in 20 Years of Research

We investigated shifts in research over the last 20 years and were glad to notice a slight increase in overall publications. Prior work mostly focused on MMORPGs, FPSs, or cross-genre-specific topics. Further, many papers focused on the PC platform, especially MS Windows, followed by mobile games or platform-independent work. The recent Game Developers Conference (GDC) report, however, indicated growing interest among developers in Linux gaming platforms, such as the Steam Deck [104], which was not reflected in our dataset.

We noticed that research on P2P methods has declined and been summarized in comprehensive works such as Yahyavi and Kemme [159]. In the early 2000s, computers became

widely connected, and MMOGs emerged, necessitating research, particularly on latency and scalability. Today, the existence of high-performance data centers around the world are a major advantage for video games, as they make it possible to counter classic problems such as game state synchronization and cheating issues with a traditional server authoritative C/S structure with unprecedented low latency compared to the past, so that P2P became less relevant. Game designs can still use zoning or sharding to distribute scalability across different servers. If games leverage P2P network architecture these days, they are usually hybrid in nature [159, 75].

Occasionally, we found work regarding TEEs. Hardware enclaves such as Intel SGX offer the advantage to perform operations or provide key material for integrity checks outside the user space, among other things, to make it more difficult for attackers to access data or operations. Recently, Choi and Ko presented important work on the feasibility of such technology for anti-cheating [22]. However, attacks against TEEs are not uncommon [172, 49, 19, 98, 15]. Meanwhile, Intel, as a major gaming CPU supplier, decided not to include SGX in consumer electronics for several years [30]. Apart from their general availability it remains questionable to what extent secure enclaves are a reliable option for anti-cheating, especially given their comparatively high technical complexity, the high-performance requirements of video games, and also cheaters potentially attacking TEEs.

Finally, as ML capacities increase, detective approaches are being researched that examine log data and behavior patterns across games of all genres. Previously, the literature shifted from using "simpler" statistical methods to more comprehensive ML pipelines. Complementarily, as ML approaches require human-in-the-loop reviewing classifications, research started investigating their explainability [141, 140, 170], and large language model assistance [131] as emerging topics.

6.2 Key Attacks & Defenses

The attack tree (Figure 2) models cheating attacks in video games, grouped into six tactics, and helped to identify recurring steps, which we summarized in playbooks. Safeguards against a playbook are therefore a major advantage, as they indirectly counter multiple cheating tactics. In essence, any cheating attack procedure typically involves three major steps.

Gathering Intelligence. Regardless of the tactic, an attacker usually needs information about their target. Typically, it is important to reverse-engineer parts of the game data to obtain information, often involving passive, unauthorized access to static or dynamic game data stored in files, memory, or network transmissions. However, a critical challenge remains: simple read access is more difficult to detect and prevent.

Newer attack approaches increasingly use object detection, e.g., for automated assistance. We assume that advances in the field of object detection and the availability models make such approaches much more feasible. Critically, these methods do not interfere with deeper layers of the client, thereby reducing the technical knowledge burden on attackers. One option could be to block screen recording technologies such as OBS or NVIDIA ShadowPlay [165]. However, this could have serious consequences, especially for video game streaming, which plays a critical role in the marketing.

Using Information. In a simple case, the information alone may be enough to give an attacker an advantage. Otherwise, the next step for an attacker is to decide what to do with this information. This involves, e.g., actively tapping into a game process and inserting code, data, or input control. Such execution is easier to detect because changes to files or memory are more obvious. Attacks can be detected and prevented, or made more difficult using the three-step safeguard measures to (i) harden the client, (ii) check the integrity of data, and (iii) perform dynamic scans during runtime. However, execution can also occur at the network level in the case of global game state attacks disrupting a game, which implies that safeguarding measures must also be directed against network traffic, e.g., a server that needs to protect against forged updates [126]. Another example concerns the proliferation of lag switches, showing that it is not always clear how to handle clients with high latency. Those can be easily detected, but the question remains of how to distinguish normal fluctuations in network latency from attacks [62].

In addition, many tactics require creating a script, an executable or "glue code" to connect information and execution, e.g., by linking object detection and bot controls. In the literature, we often found little description of how this step is performed. However, threat intelligence may offer an appealing approach by dismantling and examining cheats for their behavior [4]. Automating such approaches, which should also include prepared reporting for developers, would be highly desirable, as they would simplify the knowledge and time-intensive task of vulnerability analysis and enable developers to take immediate action.

Acting Covertly. Finally, we observed work on operating cheats undetected, representing the third step for an attacker, such as to deceive ML detection approaches by having aimbots mimic a player's normal behavior rather than generating suspicious outliers [66]. However, research on how protective measures can be deceived, or how steps in the cheating process can be concealed, was only examined marginally. Here, we see further potential to detail what stealth measures for cheats might look like, offering refinement of attack models and, in turn, safeguards accordingly.

6.3 Challenges for Research & Future Work

Based on the papers' meta comments, we concluded that video games are difficult to research. They are sophisticated software products that are closely linked to an industry known for being reluctant to discuss its work, particularly regarding security [113, 16, 75].

Openly Standardizing Anti-cheating. Standardization efforts, e.g., from Valve Corporation, by releasing open-access source code for networking stacks for games [148], which consider security that in turn affects anti-cheating, demonstrate, however, that the opposite may also be possible. We see further opportunities to standardize anti-cheating in client-side security. Client safeguards to counter access and execution steps, and thus concerning many attack procedures can be divided into three categories we previously introduced (Sec. 6.2). For example, developers can perform integrity checks on their software using signatures and checksums to detect tampering at file or process level. However, we argue that integrity-checking which extends beyond the game client is increasingly beyond developers' responsibility. This is similar with scans at game runtime which require monitoring processes to identify malicious attackers. Here, developers need significantly more knowledge of security scanning at the process and memory level [143]. Regarding safeguarding scans in the kernel space, it becomes increasingly difficult for game developers to act, as the technical complexity becomes outstandingly high and critical errors in kernel-level software can have serious security and privacy consequences for clients beyond cheating [92]. We argue that safeguards for ensuring client-side security must either be implemented outside the game development context or be examined alongside game developers in human subject studies.

Commercial providers of externalized client-side black box anti-cheat software, however, tend to keep a low profile when it comes to the exact content of their products. This raises questions about security, trust, and privacy in such solutions, given the vulnerabilities found in them [36, 122]. Cheating on a client cannot be completely prevented, but standardization and the exchange of knowledge, especially about hardening a client, can accelerate and simplify the secure production of games and a later lifecycle step of releasing updates to close gaps within the limits of developers, e.g., by reapplying obfuscation, or changing execution graphs or key material.

Acquiring Data & Collaboration. Another challenge concerns the simulation of games and players to generate datasets for research, e.g., for detective ML measures [22, 158]. Regarding bot detection in MMORPGs, we found several publications by and with gaming companies presenting research, which are welcome examples of cooperation between research and industry [68, 82, 142]. Nevertheless, the privacy policies between players and gaming companies must be complied with in all cases, and data protection rights must be upheld, as all players' data would be analyzed, including, e.g., chats that may contain personal data. It can be assumed that this poses a challenge for collaboration with researchers. Possible efforts to create a privacy-protecting framework for data sharing could use differential privacy approaches [35], which, to the best of our knowledge, have not yet been investigated in the context of gaming data for cheating detection. As attackers use ML for object detection, safeguard research, which deals with the embedding of adversarial noise into rendering the display state [164, 101, 89, 137], also faces a growing need to collaborate with developers, as such interventions can have serious consequences, such as rendering performance losses, thus, they must be well integrated.

Scientifically Underexplored Issues. Ultimately, all academic discussions on cheating that we examined focused exclusively on methods within privilege Ring 3 and 0 of computer systems [125]. However, cheating attacks at deeper privilege levels or entirely outside the standard ring model are known to exist [2]. These include techniques such as manipulating a hypervisor, which operates at Ring -1 via virtualization extensions, or by tampering with UEFI firmware or the System Management Mode at Ring -2, comparable to bootkits [12, 14, 70]. Similarly, additional external hardware devices, such as PCI direct memory access cards, pose a threat [1]. Such manipulations can be used to deploy malicious code or to read or write data in memory undetected for cheating. It is, however, possible to detect and prevent such cheating attacks, e.g., by using modern hardware features such as virtualization-based security [37, 146]; but this also requires hardware that supports such features. In addition, advances in generative AI for coding are likely to lower the barriers to entry for creating cheats [111, 128]. Given that cheating is a constantly evolving topic, it is essential to stay on top of developments and advance research on current issues so that a research–practice gap does not widen further.

7 Conclusion

In this paper, we conducted a systematic analysis of papers on technical cheating and anti-cheating from the last 20 years of security research to outline attack vectors and corresponding safeguards. We processed our findings in an attack tree, modeling six main tactics of a cheating attacker. In addition, we incorporated references to the MITRE ATT&CK framework to provide standardized guidance on attack procedures and solutions. Our analysis provides an important building block for developing threat models for video games and for standardizing anti-cheating efforts. Furthermore, we contributed a meta discussion of trends and an examination of challenges in researching cheating for academia and the gaming industry.

Acknowledgements

Dr. Marten Oltrogge made significant contributions to the ideas and analysis presented in this paper. Sadly, he passed away in December 2025, before this paper was finalized and submitted; we alone are responsible for any errors. The authors dedicate this work to his memory. This work was funded by zukunft.niedersachsen, the VolkswagenStiftung Niedersächsisches Vorab (ZN3695), and the European Union as part of the program *Digital Europe*.

Ethical Considerations

In our study we investigated cheating attack vectors, countermeasures, and their interconnection to cybersecurity topics. As we conducted a SLR and did not involve human participants, our study was not applicable by our institution's ethical review board (ERB), but we did a consultation with our ethics officers to discuss other potential side effects of our research. Further, we followed ethical principles of the *Menlo Report* for research involving information and communication technologies [71]. In the following, we structure ethical considerations related to our study according to the principles of the Menlo Report.

Respect for Persons. As we conducted an SLR and did not involve any humans, the authors doubt that our research had any consequences on harming the respect of individuals.

Beneficence. The aim of this study was to provide academia, developers, and game designers with a comprehensive taxonomy about the cheating phenomena and outline potential safeguards. By doing so, we are convinced that our research enables developers to develop more cheat-resistant games, also improving the overall security of games. However, our results could also be used by cheaters to support the development of cheats and thus harm honest players or companies. As a response to that potential harm, we would argue that our provided benefit outscales this potential harm by a lot. All described cheats in our study were already published and used by cheaters, and by providing a systematic overview of technical requirements, we provide developers with the opportunity to solve those issues.

Justice – Equity and Fairness. As we conducted a literature-based study, the authors doubt that we harmed *Equity and Fairness* of any group in any way.

Respect for Law and Public Interest. The research was conducted with care not to violate any license agreements of the publishers of our analyzed publications.

Open Science

We followed the transparency guidelines by Klemmer et al. on how to report important study information to enable the reproducibility of our results [74] and also aligned our artifacts with the USENIX Security open science policy [119]. Therefore, we provide supplementary materials that contain several intermediate results and artifacts from our attack tree construction. Please note that we do not share the PDFs of the analyzed papers because some publications are protected by publishers' license restrictions that prevent us from distributing copies. However, we provide a list of all investigated publications, including our used access links and the resulting analysis results for each publication.

We also note that we did not preregister this study.

Availability. In the supplementary materials, we publish the following study artifacts:

- **Obtained Publications:** A list containing meta information, access links, and pre-screening classifications of the 2,581 papers.
- **Investigated Publications:** A list containing the same data for the 247 papers that advanced to the systematic content analysis.
- **Codebook:** An export of our MAXQDA codebook.
- **LR Classification:** The card sorting result used for the initial cheating and safeguard assortment using the 24 SLR and SoK papers.
- **PDF Attack Tree:** As our complete attack tree does not fit into the page limitations of this paper, we provide a poster-size PDF version of our attack tree.
- **CATs:** A flattened list of all identified CAT steps.

We host our supplementary material via Zenodo at: https://doi.org/10.5281/zenodo.18428151

Usage of AI. The authors used AI for the following tasks: (i) Language translation [33] and spell checking [50], both in processing papers from the dataset and in writing this paper, (ii) large language models (Sonnet 4.5, 4.6, GPT 5.2, 5.3) [109] in (a) providing information about concepts mentioned in the analyzed papers, e.g., "Explain LSTMs [long short-term memory] in the context of neural networks," (b) assistance in distinguishing, and (c) sorting out duplicates and follow-up works in the dataset based on paper content, e.g., "Highlight the differences between *these* two papers."

References

[1] ACE. *Emerging Threats to Game Integrity: Unpacking the DMA Cheat Conundrum*. 2025. URL: https://intl.anticheatexpert.com/resource-center/content-68.html (visited on 06/09/2026).

[2] Adrián Díaz Aguilar. *How Kernel Anti-Cheats Work: A Deep Dive into Modern Game Protection*. 2026. URL: https://s4dbrd.github.io/posts/how-kernel-anti-cheats-work (visited on 06/09/2026).

[3] Hashem Alayed, Fotos Frangoudes, and Clifford Neuman. *"Behavioral-Based Cheating Detection in Online First Person Shooters Using Machine Learning Techniques"*. In: *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. IEEE, Aug. 2013.

[4] Md Sakib Anwar, Chaoshun Zuo, Carter Yagemann, and Zhiqiang Lin. *"Extracting Threat Intelligence From Cheat Binaries For Anti-Cheating"*. In: *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*. RAID '23. Association for Computing Machinery, 2023.

[5] Shilei Bai, Bin Liang, Jianjun Huang, Wei You, Jiachun Li, Yaping Li, and Wenchang Shi. *"Detecting the Capacitance-Based Gamepad for Protecting Mobile Game Fairness"*. In: *IEEE Transactions on Computers* (Sept. 1, 2021).

[6] Shane Balfe and Anish Mohammed. *"Final Fantasy – Securing On-Line Gaming with Trusted Computing"*. In: *Autonomic and Trusted Computing*. Springer Berlin Heidelberg, 2007.

[7] N.E. Baughman and B.N. Levine. *"Cheat-proof playout for centralized and distributed online games"*. In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*. 2001.

[8] Nathaniel E. Baughman, Marc Liberatore, and Brian Neil Levine. *"Cheat-Proof Playout for Centralized and Peer-to-Peer Gaming"*. In: *IEEE/ACM Transactions on Networking* (Feb. 2007).

[9] Mario Luca Bernardi, Marta Cimitile, Fabio Martinelli, and Francesco Mercaldo. *"A Time Series Classification Approach to Game Bot Detection"*. In: *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics*. ACM, June 19, 2017.

[10] Mario Luca Bernardi, Marta Cimitile, Fabio Martinelli, and Francesco Mercaldo. *"Game Bot Detection in Online Role Player Game through Behavioural Features:"* in: *Proceedings of the 12th International Conference on Software Technologies*. SCITEPRESS - Science and Technology Publications, 2017.

[11] Darrell Bethea, Robert A. Cochran, and Michael K. Reiter. *"Server-Side Verification of Client Behavior in Online Games"*. In: *ACM Transactions on Information and System Security* (Dec. 2011).

[12] binarly. *UEFI Bootkit Hunting: In-Depth Search for Unique Code Behavior*. 2025. URL: https://binarly.io/blog/uefi-bootkit-hunting-in-depth-search-for-unique-code-behavior (visited on 06/09/2026).

[13] BleepingComputer. *GTA 5 source code reportedly leaked online a year after Rockstar hack*. 2023. URL: https://bleepingcomputer.com/news/security/gta-5-source-code-reportedly-leaked-online-a-year-after-rockstar-hack/ (visited on 01/24/2026).

[14] Jussi's blog. *Plouton - A System Management Mode cheat framework*. 2025. URL: https://jussihi.kapsi.fi/2025-03-14-smmcheat (visited on 06/09/2026).

[15] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. *"Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution"*. In: *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. USENIX Association, 2018.

[16] Elie Bursztein. *"I am a legend: Hacking hearthstone using statistical learning methods"*. In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. 2016.

[17] Huixia Cao, Yunkun Li, and Zhaohao Liang. *"BEAT: Behavior Evaluation and Anomaly Tracking, Game Bot Detection Framework in RPG Games"*. In: *2023 6th International Conference on Algorithms Computing and Artificial Intelligence*. ACM, Dec. 22, 2023.

[18] Angela Carrera-Rivera, William Ochoa, Felix Larrinaga, and Ganix Lasa. *"How-to conduct a systematic literature review: A quick guide for computer science research"*. In: *MethodsX* (2022).

[19] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. *"SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems"*. In: *2020 IEEE Symposium on Security and Privacy (SP)*. 2020.

[20] Chris Chambers, Wu-chang Feng, Wu-chi Feng, and Debanjan Saha. *"Mitigating Information Exposure to Cheaters in Real-Time Strategy Games"*. In: *Proceedings of the 15th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, June 13, 2005.

[21] Ying-Chieh Chen, Jing-Jang Hwang, Ronggong Song, G. Yee, and L. Korba. *"Online Gaming Cheating and Security Issue"*. In: *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*. IEEE, 2005.

[22] Minyeop Choi and Gihyuk Ko. *"BotScreen: Trust Everybody, but Cut the Aimbots Yourself"*. In: *32nd USENIX Security Symposium (USENIX Security 23)* (2023).

[23] YeonJun Choi, SungJune Chang, YongJun Kim, HunJoo Lee, WookHo Son, and SeongIl Jin. *"Detecting and Monitoring Game Bots Based on Large-Scale User-Behavior Log Data Analysis in Multiplayer Online Games"*. In: *The Journal of Supercomputing* (Sept. 2016).

[24] Yang-Wai Chow, Willy Susilo, and Hua-Yu Zhou. *"CAPTCHA Challenges for Massively Multiplayer Online Games: Mini-game CAPTCHAs"*. In: *2010 International Conference on Cyberworlds*. IEEE, Oct. 2010.

[25] Victoria Clarke and Virginia Braun. *"Thematic analysis"*. In: *Encyclopedia of critical psychology*. Springer, 2014.

[26] Victoria Clarke and Virginia Braun. *"Thematic analysis"*. In: *The Journal of Positive Psychology* (2017).

[27] Sam Collins, Alex Poulopoulos, Marius Muench, and Tom Chothia. *"Anti-Cheat: Attacks and the Effectiveness of Client-Side Defences"*. In: *Proceedings of the 2024 Workshop on Research on Offensive and Defensive Techniques in the Context of Man At The End (MATE) Attacks*. CheckMATE '24. Association for Computing Machinery, 2024.

[28] Mia Consalvo. *Cheating: Gaining Advantage in Videogames*. The MIT Press, 2007.

[29] A.B. Corman, S. Douglas, P. Schachte, and V. Teague. *"A Secure Event Agreement (SEA) Protocol for Peer-to-Peer Games"*. In: *First International Conference on Availability, Reliability and Security (ARES'06)*. IEEE, 2006.

[30] Intel Corporation. *Intel Processors Supporting Intel SGX*. 2026. URL: https://intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions-processors.html (visited on 02/05/2026).

[31] CyberSecurityNews. *Unpatched Marvel Game RCE Exploit Could Let Hackers Take Over PCs & PS5s*. 2025. URL: https://cybersecuritynews.com/unpatched-marvel-game-rce-exploit/ (visited on 01/24/2026).

[32] dblp. *dblp Computer Science Bibliography*. 2026. URL: https://dblp.org/ (visited on 02/05/2026).

[33] DeepL. *DeepL Translator*. URL: https://deepl.com (visited on 06/02/2026).

[34] Delve. *Open, Axial, and Selective Coding in Qualitative Research: A Practical Guide*. URL: https://blog.delvetool.com/blog/openaxialselective (visited on 06/02/2026).

[35] Damien Desfontaines and Balázs Pejó. *"SoK: Differential privacies"*. In: *Proc. Priv. Enhancing Technol.* (2020).

[36] Christoph Dorner and Lukas Daniel Klausner. *"If It Looks Like a Rootkit and Deceives Like a Rootkit: A Critical Examination of Kernel-Level Anti-Cheat Systems"*. In: *Proceedings of the 19th International Conference on Availability, Reliability and Security*. ARES '24. Association for Computing Machinery, 2024.

[37] FACEIT. *FACEIT Anti-Cheat Final Rollout Of TPM 2.0, Secure Boot, IOMMU & VBS Requirements*. 2025. URL: https://faceit.com/en/news/faceit-rollout-of-tpm-secure-boot-iommu-and-vbs (visited on 06/09/2026).

[38] Lu Fan, Phil Trinder, and Hamish Taylor. *"Design Issues for Peer-to-Peer Massively Multiplayer Online Games"*. In: *International Journal of Advanced Media and Communication* (2010).

[39] Wu-chang Feng, Ed Kaiser, and Travis Schluessler. *"Stealth measurements for cheat detection in on-line games"*. In: *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*. NetGames '08. Association for Computing Machinery, 2008.

[40] Stefano Ferretti and Marco Roccetti. *"Game time modelling for cheating detection in P2P MOGs: a case study with a fast rate cheat"*. In: *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games*. NetGames '06. Association for Computing Machinery, 2006.

[41] Stefano Ferretti, Marco Roccetti, and Roberta Zioni. *"A Statistical Approach to Cheating Countermeasure in P2P MOGs"*. In: *2009 6th IEEE Consumer Communications and Networking Conference*. 2009.

[42] MITRE: Center for Threat Informed Defense. *ATT&CK Matrix for Enterprise*. 2025. URL: https://attack.mitre.org/ (visited on 01/21/2026).

[43] MITRE: Center for Threat Informed Defense. *Threat Modeling with ATT&CK v1.0.1*. 2024. URL: https://center-for-threat-informed-defense.github.io/threat-modeling-with-attack/question-2/ (visited on 01/21/2026).

[44] Florent Fortat, Maryline Laurent, and Michel Simatic. "*Games Based on Active NFC Objects: Model and Security Requirements*". In: *2015 International Workshop on Network and Systems Support for Games (NetGames)*. IEEE, Dec. 2015.

[45] Olga Gadyatskaya and Nathan Daniel Schiele. "*A limited technical background is sufficient for attack-defense tree acceptability*". In: *Proceedings of the 34th USENIX Conference on Security Symposium*. SEC '25. USENIX Association, 2025.

[46] Luca Galli, Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. "*A cheating detection framework for Unreal Tournament III: A machine learning approach*". In: *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*. 2011.

[47] Steven Gianvecchio, Zhenyu Wu, Mengjun Xie, and Haining Wang. "*Battle of Botcraft: Fighting Bots in Online Games with Human Observational Proofs*". In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. ACM, Nov. 9, 2009.

[48] GosuGamers. *Call of Duty: WWII pulled from Xbox PC Game Pass amid RCE exploit chaos*. 2025. URL: https://gosugamers.net/entertainment/news/75984-call-of-duty-wwii-pulled-from-xbox-pc-game-pass-amid-rce-exploit-chaos/ (visited on 01/24/2026).

[49] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. "*Cache Attacks on Intel SGX*". In: *Proceedings of the 10th European Workshop on Systems Security*. Association for Computing Machinery, 2017.

[50] Grammarly. *Writing Assistance*. URL: https://grammarly.com (visited on 06/02/2026).

[51] Robert J. Hall. "*Cheating Attacks and Resistance Techniques in GeoGame Design*". In: *Proceedings of the International Academic Conference on the Future of Game Design and Technology*. ACM, May 6, 2010.

[52] Mee Lan Han, Byung Il Kwak, and Huy Kang Kim. "*Cheating and Detection Method in Massively Multiplayer Online Role-Playing Game: Systematic Literature Review*". In: *IEEE Access* (2022).

[53] Jianan Hao and Suiping Zhou. "*SAFE: A Secure Area of Interests Filter for P2P-based MMOGs*". In: *2010 10th IEEE International Conference on Computer and Information Technology*. IEEE, June 2010.

[54] Bill Haskins, Jonette Stecklein, Brandon Dick, Gregory Moroney, Randy Lovell, and James Dabney. "*Error Cost Escalation Through the Project Life Cycle*". In: *INCOSE International Symposium* (2004).

[55] Jiankun Hu and Fabio Zambetta. "*Security Issues in Massive Online Games*". In: *Security and Communication Networks* (Jan. 2008).

[56] Yixuan Hu. "*The Impact of Artificial Intelligence's Automatic Targeting on Traditional Online Games Cheating*". In: *International Journal of Computer Science and Information Technology* (Nov. 24, 2024).

[57] igda. *Developer Satisfaction Survey (DSS)*. 2023. URL: https://igda.org/dss/ (visited on 01/24/2026).

[58] Md Shihabul Islam, Bo Dong, Swarup Chandra, Latifur Khan, and Bhavani M. Thuraisingham. "*GCI: A GPU Based Transfer Learning Approach for Detecting Cheats of Computer Game*". In: *IEEE Transactions on Dependable and Secure Computing* (2020).

[59] Jared Jardine and Daniel Zappala. "*A Hybrid Architecture for Massively Multiplayer Online Games*". In: *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*. ACM, Oct. 21, 2008.

[60] Manadda Jaruschaimongkol, Chayakan Chanchai, Sarawut Kotrcha, and Thitirat Siriborvornratanakul. "*Enhanced Bot Game for a Massively Multiplayer Online Role-Playing Game (MMORPG) Using Object Detection*". In: *SN Computer Science* (Aug. 30, 2024).

[61] Albert B. Jeng and Chia Ling Lee. "*A Study on Online Game Cheating and the Effective Defense*". In: *Recent Trends in Applied Artificial Intelligence*. Springer Berlin Heidelberg, 2013.

[62] Alexander Joens, Ananth A. Jillepalli, and Frederick T. Sheldon. "*Trustworthy High-Performance Multiplayer Games with Trust-but-Verify Protocol Sensor Validation*". In: *Sensors* (July 21, 2024).

[63] Aditya Jonnalagadda, Iuri Frosio, Seth Schneider, Morgan McGuire, and Joohwan Kim. "*Robust Vision-Based Cheat Detection in Competitive Gaming*". In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* (Apr. 26, 2021).

[64] Patric Kabus and Alejandro P. Buchmann. "*Design of a Cheat-Resistant P2P Online Gaming System*". In: *Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts*. ACM, Sept. 19, 2007.

[65] Edward Kaiser, Wu-chang Feng, and Travis Schluessler. "*Fides: remote anomaly-based cheat detection using client emulation*". In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS '09. Association for Computing Machinery, 2009.

[66] Anssi Kanervisto, Tomi Kinnunen, and Ville Hautamäki. "*GAN-Aimbots: Using Machine Learning for Cheating in First Person Shooters*". In: *IEEE Transactions on Games* (Dec. 2023).

[67] Ah Reum Kang, Seong Hoon Jeong, Aziz Mohaisen, and Huy Kang Kim. "*Multimodal Game Bot Detection Using User Behavioral Characteristics*". In: *SpringerPlus* (Dec. 2016).

[68] Ah Reum Kang, Jiyoung Woo, Juyong Park, and Huy Kang Kim. "*Online Game Bot Detection Based on Party-Play Log Analysis*". In: *Computers & Mathematics with Applications* (May 2013).

[69] Yong Goo Kang and Huy Kang Kim. "*Quick and Easy Game Bot Detection Based on Action Time Interval Estimation*". In: *ETRI Journal* (Aug. 2023).

[70] Panicos Karkallis and Jorge Blasco. *VIC: Evasive Video Game Cheating via Virtual Machine Introspection*. 2025. arXiv: 2502.12322.

[71] Erin E. Kenneally and David Dittrich. *The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research*. Aug. 2012. URL: https://ssrn.com/abstract=2445102.

[72] Hyunsoo Kim, Jun Hee Kim, Jaeman Son, Jihoon Song, and Eunjo Lee. "*A Framework for Mining Collectively-Behaving Bots in MMORPGs*". In: *Pattern Recognition*. Springer Nature Switzerland, 2025.

[73] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. "*Systematic literature reviews in software engineering - A systematic literature review*". In: *Inf. Softw. Technol.* (Jan. 2009).

[74] Jan H. Klemmer, Juliane Schmüser, Jacques Suray, Jan-Ulrich Holtgrave, Byron Lowens, Florian Schaub, and Sascha Fahl. *Transparency Guidelines for Human-Centered Security and Privacy Research*. Dec. 2025. URL: https://doi.org/10.5281/zenodo.18030545.

[75] Philip Klostermeyer, Sabrina Klivan, Sandra Höltervennhoff, Alexander Krause, Niklas Busch, and Sascha Fahl. "*Skipping the Security Side Quests: A Qualitative Study on Security Practices and Challenges in Game Development*". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. CCS '24. Association for Computing Machinery, 2024.

[76] Denis Kotkov, Gaurav Pandey, and Alexander Semenov. "*Gaming Bot Detection: A Systematic Literature Review*". In: *Computational Data and Social Networks*. Springer International Publishing, 2018.

[77] Sandeep Kulkarni, Minhaj Khan, S. Bulomine Regi, Dipans Verma, and Tejas Tagad. "*Anti-Cheat and Cybersecurity in eSports and Gaming: A Case Study*". In: *International Journal of Inventive Engineering and Sciences* (June 30, 2025).

[78] Young Eun Kwon. "*Unveiling Shadow Markets: A Scalable Anomaly Detection Framework for Gold Farming and RMT in MMORPG Player-Driven Markets*". In: *Companion Proceedings of the Annual Symposium on Computer-Human Interaction in Play*.

CHI PLAY Companion '25. Association for Computing Machinery, 2025.

[79] Samuli Laato, Sampsa Rauti, Lauri Koivunen, and Jouni Smed. *"Technical cheating prevention in location-based games"*. In: *Proceedings of the 22nd International Conference on Computer Systems and Technologies*. CompSysTech '21. Association for Computing Machinery, 2021.

[80] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. *"SoK: Taxonomy of Attacks on Open-Source Software Supply Chains"*. In: *2023 IEEE Symposium on Security and Privacy (SP)*. 2023.

[81] Peter Laurens, Richard F. Paige, Phillip J. Brooke, and Howard 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)*. 2007.

[82] Eunjo Lee, Jiyoung Woo, Hyoungshick Kim, Aziz Mohaisen, and Huy Kang Kim. *"You Are a Game Bot!: Uncovering Game Bots in MMORPGs via Self-similarity in the Wild"*. In: *Proceedings 2016 Network and Distributed System Security Symposium*. Internet Society, 2016.

[83] Wenbin Li, Xiaokai Chu, Yueyang Su, Di Yao, Shiwei Zhao, Runze Wu, Shize Zhang, Jianrong Tao, Hao Deng, and Jingping Bi. *"FingFormer: Contrastive Graph-based Finger Operation Transformer for Unsupervised Mobile Game Bot Detection"*. In: *Proceedings of the ACM Web Conference 2022*. ACM, Apr. 25, 2022.

[84] Ruei-Min Lin, Hwai-Chung Ho, and Kuan-Ta Chen. *"Bot detection in rhythm games: a physiological approach"*. In: *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*. Association for Computing Machinery, 2011.

[85] Chang Liu, Zichun Gao, Zhenyu Liao, Yue Sun, and Xianglong Feng. *"An Exploration of Human Pose Estimation Based Cheating Tools for FPS Video Game and its Defense Solution"*. In: *2024 IEEE 26th International Workshop on Multimedia Signal Processing (MMSP)*. 2024.

[86] Daiping Liu, Xing Gao, Mingwei Zhang, Haining Wang, and Angelos Stavrou. *"Detecting Passive Cheats in Online Games via Performance-Skillfulness Inconsistency"*. In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, June 2017.

[87] Minhua Ma and Andreas Oikonomou. *"Network Architectures and Data Management for Massively Multiplayer Online Games:"* in: *International Journal of Grid and High Performance Computing* (Oct. 1, 2010).

[88] Francis Martinson and Dylan Rangel. *"A Comprehensive Analysis of Game Hacking through Injectors: Exploits, Defenses and Beyond"*. In: *International Journal of Computer Applications* (Sept. 22, 2023).

[89] Ken Matsuhira, Shun Hattori, and Wataru Sunayama. *"Effects of Adversarial Patches Against Object Detection on Combating Cheats in Online First Person Shooter Games"*. In: *2024 Joint 13th International Conference on Soft Computing and Intelligent Systems and 25th International Symposium on Advanced Intelligent Systems (SCIS&ISIS)*. IEEE, Nov. 9, 2024.

[90] Sjouke Mauw and Martijn Oostdijk. *"Foundations of Attack Trees"*. In: *Information Security and Cryptology - ICISC 2005*. Springer Berlin Heidelberg, 2006.

[91] MAXQDA. *MAXQDA Qualitative Data Analysis Software*. 2026. URL: https://maxqda.com/ (visited on 02/05/2026).

[92] Trend Micro. *Ransomware Actor Abuses Genshin Impact Anti-Cheat Driver to Kill Antivirus*. 2022. URL: https://trendmicro.com/en%5C%5Fus/research/22/h/ransomware-actor-abuses-genshin-impact-anti-cheat-driver-to-kill-antivirus.html (visited on 01/24/2026).

[93] Microsoft. *DebugActiveProcess*. Oct. 2021. URL: https://learn.microsoft.com/en-us/windows/win32/api/debugapi/nf-debugapi-debugactiveprocess (visited on 02/05/2026).

[94] Microsoft. *SendMessage*. Aug. 2022. URL: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendmessage (visited on 02/05/2026).

[95] Miro. *Miro*. 2026. URL: https://miro.com/ (visited on 02/05/2026).

[96] Y. Mishima, K. Fukuda, and H. Esaki. *"An Analysis of Players and Bots Behaviors in MMORPG"*. In: *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, Mar. 2013.

[97] Stefan Mitterhofer, Christopher Kruegel, Engin Kirda, and Christian Platzer. *"Server-Side Bot Detection in Massively Multiplayer Online Games"*. In: *IEEE Security & Privacy Magazine* (May 2009).

[98] Daniel Moghimi, Jo Van Bulck, Nadia Heninger, Frank Piessens, and Berk Sunar. *"CopyCat: Controlled Instruction-Level Attacks on Enclaves"*. In: *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*. USENIX Association, 2020.

[99] Christian Mönch, Gisle Grimen, and Roger Midtstraum. *"Protecting Online Games against Cheating"*. In: *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games - NetGames '06*. ACM Press, 2006.

[100] Emerson Murphy-Hill, Thomas Zimmermann, and Nachiappan Nagappan. *"Cowboys, ankle sprains, and keepers of quality: how is video game development different from software development?"* In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. Association for Computing Machinery, 2014.

[101] Anh Nhu, Hieu Phan, Chang Liu, and Xianglong Feng. *"A Comprehensive Defense Approach Targeting The Computer Vision Based Cheating Tools in FPS Video Games"*. In: *2023 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. IEEE, Nov. 17, 2023.

[102] Yuseung Noh, Seonghoon Jeong, and Huy Kang Kim. *"Trading Behind-the-Scene: Analysis of Online Gold Farming Network in the Auction House System"*. In: *IEEE Transactions on Games* (Sept. 2022).

[103] NVIDIA. *ShadowPlay*. URL: https://nvidia.com/en-us/software/nvidia-app (visited on 02/05/2026).

[104] GDC Festival of Gaming c/o Informa Group. *2026 State of the Game Industry Report*. 2026. URL: https://reg.gdconf.com/2026-SOTI/ (visited on 02/05/2026).

[105] OpenAlex. *OpenAlex*. 2026. URL: https://openalex.org/ (visited on 02/05/2026).

[106] Julia Orlova, Anton Stepanov, Anton Vinogradov, Lubov Orlova, Anna Baldycheva, and Andrey Somov. *"Estimation of In-Game Player Behavior by Measuring Key Gameplay Parameters"*. In: *2024 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE, May 20, 2024.

[107] Jeni Paay, Jesper Kjeldskov, Daniele Internicola, and Mikkel Thomasen. *"Motivations and Practices for Cheating in Pokémon GO"*. In: *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services*. ACM, Sept. 3, 2018.

[108] Seonghyun Park, Adil Ahmad, and Byoungyoung Lee. *"BlackMirror: Preventing Wallhacks in 3D Online FPS Games"*. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Association for Computing Machinery, 2020.

[109] Perplexity. *Web search engine*. URL: https://perplexity.ai (visited on 06/02/2026).

[110] José Pedro Pinto, André Pimenta, and Paulo Novais. *"Deep Learning and Multivariate Time Series for Cheat Detection in Video Games"*. In: *Machine Learning* (Dec. 2021).

[111] Serafim Pinto. *A guide to identifying cheating in online video games in 2026*. 2026. URL: https://gamesindustry.biz/a-guide-to-identifying-cheating-in-online-video-games-in-2026 (visited on 06/12/2026).

[112] Daniel Pittman and Chris Gauthierdickey. *"Match+Guardian: a secure peer-to-peer trading card game protocol"*. In: *Multimedia Syst.* (June 2013).

[113] Christian Platzer. *"Sequence-based bot detection in massive multiplayer online games"*. In: *2011 8th International Conference on Information, Communications and Signal Processing*. 2011.

[114] Aaron Portnoy and Ali Rizvi-Santiago. *"Walking on Water: A Cheating Case Study"*. In: *IEEE Security & Privacy Magazine* (May 2009).

[115] James Prather, Robert Nix, and Ryan Jessup. *"Trust Management for Cheating Detection in Distributed Massively Multiplayer Online Games"*. In: *2017 15th Annual Workshop on Network and Systems Support for Games (NetGames)*. IEEE, June 2017.

[116] OBS Project. *Open Broadcaster Software*. URL: https://obsproject.com (visited on 02/05/2026).

[117] Xianyang Qi, Jiashu Pu, Shiwei Zhao, Runze Wu, and Jianrong Tao. *"A GNN-Enhanced Game Bot Detection Model for MMORPGs"*. In: *Advances in Knowledge Discovery and Data Mining*. Springer International Publishing, 2022.

[118] Leo Thomas Ramos and Angel D. Sappa. *"A Decade of You Only Look Once (YOLO) for Object Detection: A Review"*. In: *IEEE Access* (2025).

[119] Elissa Redmiles and Ben Stock. *USENIX Security '26 Call for Papers*. 2025. URL: https://usenix.org/conference/usenixsecurity26/call-for-papers/ (visited on 01/21/2026).

[120] Rosslin John Robles, Sang-Soo Yeo, Young-Deuk Moon, Gilcheol Park, and Seoksoo Kim. *"Online Games and Security Issues"*. In: *2008 Second International Conference on Future Generation Communication and Networking*. IEEE, Dec. 2008.

[121] Jan Sablatnig, Fritz Lehmann-Grube, Sven Grottke, and Sabine Cikic. *"Dealing with Multiple Truths in Online Virtual Worlds"*. In: *International Journal of Digital Crime and Forensics (IJDCF)* (2009).

[122] Mohamed Sabt. *"Battling The Eye: Exploring the Anti-Cheat Techniques of BattlEye"*. In: *Proceedings of the 2025 Workshop on Research on Offensive and Defensive Techniques in the Context of Man At The End (MATE) Attacks*. Association for Computing Machinery, 2025.

[123] Bruce Schneier. *"Attack Trees"*. In: *Dr Dobb's Journal* (Apr. 1999). eprint: https://schneier.com/academic/archives/1999/12/attack_trees.html.

[124] Semantic Scholar. *Semantic Scholar*. 2026. URL: https://semanticscholar.org/ (visited on 02/05/2026).

[125] Michael D. Schroeder and Jerome H. Saltzer. *"A hardware architecture for implementing protection rings"*. In: *Commun. ACM* (1972).

[126] Salman Shaikh, Hugo Bertin, Marc Dacier, and Yérom-David Bromberg. *"A Framework to Enable Cheating Detection in Multiplayer Online Games"*. In: *Cybersecurity*. Springer Nature Switzerland, 2025.

[127] Huaping Shen, Sajal K. Das, Mohan Kumar, and Zhijun Wang. *"Semi-Lock: An Efficient Cheat-Proof Synchronization Mechanism for Peer-to-Peer Game Systems"*. In: *Embedded and Ubiquitous Computing*. Springer Berlin Heidelberg, 2006.

[128] Ty Sherback. *I asked a local AI to build me a cheat for a popular sandbox game, and only Claude Code could finish the job*. 2026. URL: https://xda-developers.com/i-asked-a-local-ai-to-build-me-a-cheat-for-a-popular-sandbox-game-and-only-claude-code-could-finish-the-job/ (visited on 06/12/2026).

[129] Marcos A. Simplicio, Mateus A.S. Santos, Rodrigo R. Leal, Marco A.L. Gomes, and Walter A. Goya. *"SecureTCG: A Lightweight Cheating-detection Protocol for P2P Multiplayer Online Trading Card Games"*. In: *Security and Communication Networks* (Dec. 2014).

[130] Uday Pratap Singh, Bersha Kumari, Dinesh Goyal, and Mohd Afrid. *"Exploring Novel Machine Learning Approaches for Cheat Detection in Competitive Gaming"*. In: *Proceedings of the 5th International Conference on Information Management & Machine Intelligence*. ACM, Nov. 23, 2023.

[131] Jaeman Son and Hyunsoo Kim. *Human-AI Collaborative Bot Detection in MMORPGs*. 2025. arXiv: 2508.20578 [cs.AI].

[132] Donna Spencer. *Card Sorting: Designing Usable Categories*. Rosenfeld Media, LLC, 2009.

[133] Ruan Spijkerman and Elizabeth Marie Ehlers. *"Cheat Detection in a Multiplayer First-Person Shooter Using Artificial Intelligence Tools"*. In: *Proceedings of the 2020 3rd International Conference on Computational Intelligence and Intelligent Systems*. ACM, Nov. 13, 2020.

[134] Daniele Spina, Andrea Bottino, Davide Cavagnino, Leonardo Chiariglione, Maurizio Lucenteforte, Marco Mazzaglia, and Francesco Strada. *"AI Server-Side Prediction for Latency Mitigation and Cheating Detection: The MPAI-SPG Approach"*. In: *2024 IEEE Gaming, Entertainment, and Media Conference (GEM)*. IEEE, June 5, 2024.

[135] Statista. *Games - Worldwide*. 2026. URL: https://statista.com/outlook/amo/media/games/worldwide/ (visited on 01/24/2026).

[136] sumo logic. *What are TTPs?* 2026. URL: https://sumologic.com/glossary/tactics-techniques-procedures (visited on 06/11/2026).

[137] Chenxin Sun, Kai Ye, Liangcai Su, Jiayi Zhang, and Chenxiong Qian. *"Invisibility Cloak: Proactive Defense Against Visual Game Cheating"*. In: *33rd USENIX Security Symposium (USENIX Security 24)* (2024).

[138] Willy Susilo, Yang Wai Chow, and Rungrat Wiangsripanawan. *"Protecting Peer-to-Peer-Based Massively Multiplayer Online Games"*. In: *International Journal of Computational Science and Engineering* (2015).

[139] Kornélia Sára Szatmáry. *"Cybersecurity of the Gaming Industry"*. In: *2024 IEEE 22nd Jubilee International Symposium on Intelligent Systems and Informatics (SISY)*. IEEE, Sept. 19, 2024.

[140] Jianrong Tao, Yu Xiong, Shiwei Zhao, Runze Wu, Xudong Shen, Tangjie Lyu, Changjie Fan, Zhipeng Hu, Sha Zhao, and Gang Pan. *"Explainable AI for Cheating Detection and Churn Prediction in Online Games"*. In: *IEEE Transactions on Games* (June 2023).

[141] Jianrong Tao, Yu Xiong, Shiwei Zhao, Yuhong Xu, Jianshi Lin, Runze Wu, and Changjie Fan. *"XAI-Driven Explainable Multi-view Game Cheating Detection"*. In: *2020 IEEE Conference on Games (CoG)*. IEEE, Aug. 2020.

[142] Jianrong Tao, Jiarong Xu, Linxia Gong, Yifu Li, Changjie Fan, and Zhou Zhao. *"NGUARD: A Game Bot Detection Framework for NetEase MMORPGs"*. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, July 19, 2018.

[143] Luan Bui The and Van Nguyen Khanh. *"GameGuard: a windows-based software architecture for protecting online games against hackers"*. In: *Proceedings of the 1st Symposium on Information and Communication Technology*. SoICT '10. Association for Computing Machinery, 2010.

[144] Yuan Tian, Eric Chen, Xiaojun Ma, Shuo Chen, Xiao Wang, and Patrick Tague. *"Swords and Shields: A Study of Mobile Game Hacks and Existing Defenses"*. In: *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, Dec. 5, 2016.

[145] Igor Tomicic, Petra Grd, and Markus Schatten. *"Reverse Engineering of the MMORPG Client Protocol"*. In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, May 2019.

[146] tulach.cc. *Using VBS enclaves for anti-cheat purposes*. 2024. URL: https://tulach.cc/using-vbs-enclaves-for-anti-cheat-purposes (visited on 06/09/2026).

[147] David Urbina, Yufei Gu, Juan Caballero, and Zhiqiang Lin. *"SigPath: A Memory Graph Based Approach for Program Data Introspection and Modification"*. In: *Computer Security - ESORICS 2014*. Springer International Publishing, 2014.

[148] ValveSoftware. *ValveSoftware/GameNetworkingSockets*. 2026. URL: https://github.com/ValveSoftware/GameNetworkingSockets (visited on 02/05/2026).

[149] The Verge. *Apex Legends devs confirm esports hacking incident*. 2024. URL: https://theverge.com/2024/3/18/24104666/apex-legends-postpones-algs-competition-hack-concerns/ (visited on 01/24/2026).

[150] Maxime Véron, Olivier Marin, Sébastien Monnet, and Zahia Guessoum. *"Towards a Scalable Refereeing System for Online Gaming"*. In: *Multimedia Systems* (Oct. 2014).

[151] VideoGamesChronicle. *Dark Souls servers taken down due to an exploit 'that could let someone take over your PC'*. 2022. URL: https://videogameschronicle.com/news/dark-souls-servers-taken-down-due-to-an-exploit-that-could-let-someone-take-over-your-pc/ (visited on 01/24/2026).

[152] Yilin Wang, Sha Zhao, Shiwei Zhao, Runze Wu, Yuhong Xu, Jianrong Tao, Tangjie Lv, Shijian Li, Zhipeng Hu, and Gang Pan. *"PU-Detector: A PU Learning-based Framework for Real Money Trading Detection in MMORPG"*. In: *ACM Trans. Knowl. Discov. Data* (Feb. 2024).

[153] Yu-Chih Wei, You-Xin Lai, Hai-Po Su, and Yu-Wen Yen. *"Detecting Online Game Malicious Chargeback by using k-NN"*. In: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2020.

[154] Adam Wierzbicki. *"TRUST MANAGEMENT WITHOUT REPUTATION IN P2P GAMES:"* in: *Proceedings of the International Conference on Security and Cryptography*. SciTePress - Science and and Technology Publications, 2006.

[155] Adam Wierzbicki and Tomasz Kaszuba. *"Practical trust management without reputation in peer-to-peer games"*. In: *Multiagent Grid Syst.* (2007).

[156] Jiyoung Woo and Huy Kang Kim. *"Survey and research direction on online game security"*. In: *Proceedings of the Workshop at SIGGRAPH Asia*. WASA '12. Association for Computing Machinery, 2012.

[157] Craig Wright. *"Security Policy Overview"*. In: *The IT Regulatory and Standards Compliance Handbook*. Syngress, 2008. Chap. 6.

[158] Amir Yahyavi, Kévin Huguenin, Julien Gascon-Samson, Jörg Kienzle, and Bettina Kemme. *"Watchmen: Scalable Cheat-Resistant Support for Distributed Multi-player Online Games"*. In: *2013 IEEE 33rd International Conference on Distributed Computing Systems*. 2013.

[159] Amir Yahyavi and Bettina Kemme. *"Peer-to-Peer Architectures for Massively Multiplayer Online Games: A Survey"*. In: *ACM Computing Surveys* (Oct. 2013).

[160] Amir Yahyavi, Jeffrey Pang, and Bettina Kemme. *"Towards Providing Security for Mobile Games"*. In: *Proceedings of the Eighth ACM International Workshop on Mobility in the Evolving Internet Architecture*. ACM, Oct. 4, 2013.

[161] Roman V. Yampolskiy and Venu Govindaraju. *"Embedded noninteractive continuous bot detection"*. In: *Comput. Entertain.* (Mar. 2008).

[162] Jeff Yan and Brian Randell. *"A systematic classification of cheating in online games"*. In: *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*. NetGames '05. Association for Computing Machinery, 2005.

[163] Jeff Yan and Brian Randell. *"An Investigation of Cheating in Online Games"*. In: *IEEE Security & Privacy Magazine* (May 2009).

[164] Kai Yan, Xianyi Chen, Qi Cui, Haoqin Yuan, and Zhenshan Tan. *"AdvMap: Crafting Adversarial Maps to Counter AI Aimbot in First-Person Shooter Games"*. In: *IEEE Transactions on Games* (Sept. 2025).

[165] Gefei Yang, Yuqi Zhang, Feihong Liu, and Zishuo Gao. *"Machine Learning Anti-Cheating Algorithm and a Test against Computer Vision Aimbot"*. In: *Applied and Computational Engineering* (June 14, 2023).

[166] S. F. Yeung and John C. S. Lui. *"Dynamic Bayesian Approach for Detecting Cheats in Multi-Player Online Games"*. In: *Multimedia Systems* (Sept. 2008).

[167] S.F. Yeung, J.C.S. Lui, Jiangchuan Liu, and J. Yan. *"Detecting Cheaters for Multiplayer Games: Theory, Design and Implementation[1]"*. In: *CCNC 2006. 2006 3rd IEEE Consumer Communications and Networking Conference, 2006.* IEEE, 2006.

[168] Su-Yang Yu, Nils Hammerla, Jeff Yan, and Peter Andras. *"A Statistical Aimbot Detection Method for Online FPS Games"*. In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE, June 2012.

[169] Li Yujian and Liu Bo. *"A Normalized Levenshtein Distance Metric"*. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2007).

[170] Jiayi Zhang, Chenxin Sun, and Chenxiong Qian. *"XGuardian: Towards Generalized, Explainable and More Effective Server-side Anti-cheat in First-Person Shooter Games"*. In: *35rd USENIX Security Symposium (USENIX Security 26)* (2026).

[171] Lexuan Zhang, Yuhan Wei, and Yitong Zhou. *"Leveraging Machine Learning: Advancements in Cheating Detection Strategies for Ensuring Fair Online Gaming Environments"*. In: *Science and Technology of Engineering, Chemistry and Environmental Protection* (Aug. 14, 2024).

[172] Ruiyi Zhang, Tristan Hornetz, Daniel Weber, Fabian Thomas, and Michael Schwarz. *"StackWarp: Breaking AMD SEV-SNP Integrity via Deterministic Stack-Pointer Manipulation through the CPU's Stack Engine"*. In: *USENIX Security*. 2026.

[173] Zhongqqiang Zhang, Hiroaki Anada, Junpei Kawamoto, and Kouichi Sakurai. *"Detection of Illegal Players in Massively Multiplayer Online Role Playing Game by Classification Algorithms"*. In: *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*. IEEE, Mar. 2015.

[174] Sha Zhao, Junwei Fang, Shiwei Zhao, Runze Wu, Jianrong Tao, Shijian Li, and Gang Pan. *"T-Detector: A Trajectory Based Pre-trained Model for Game Bot Detection in MMORPGs"*. In: *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, May 2022.

[175] Shiwei Zhao, Jiaheng Qi, Zhipeng Hu, Han Yan, Runze Wu, Xudong Shen, Tangjie Lv, and Changjie Fan. *"VESPA: A General System for Vision-Based Extrasensory Perception Anticheating in Online FPS Games"*. In: *IEEE Transactions on Games* (Sept. 2024).

A Playbooks

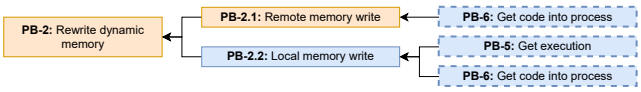


Figure 3: Playbook PB-2: Rewrite dynamic memory.

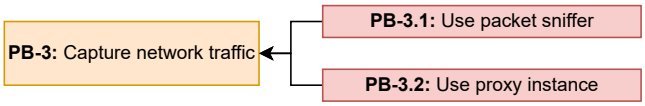


Figure 4: Playbook PB-3: Capture network traffic.

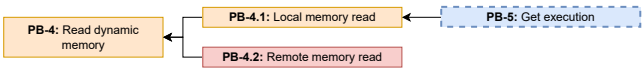


Figure 5: Playbook PB-4: Read dynamic memory.

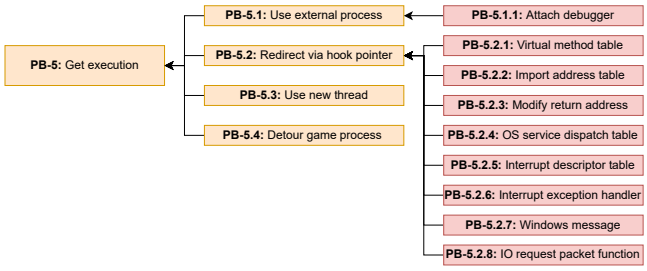


Figure 6: Playbook PB-5: Get execution.

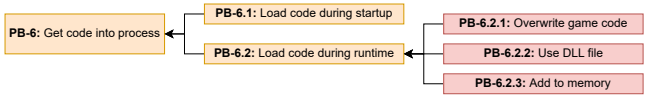


Figure 7: Playbook PB-6: Get code into process.