Anti-Cheat: Attacks and the Effectiveness of Client-Side Defences

Sam Collins
University of Birmingham
Birmingham, UK
sxc1327@student.bham.ac.uk

Alex Poulopoulos
University of Warwick
Coventry, UK
alex.poulopoulos@gmail.com

Marius Muench
University of Birmingham
Birmingham, UK
m.muench@bham.ac.uk

Tom Chothia
University of Birmingham
Birmingham, UK
t.chothia@bham.ac.uk

Abstract

This paper studies game cheats as one of the most widespread forms of man-at-the-end (MATE) attacks. We analyse 80 representative web sites selling game cheats, finding an active market, with cheats selling for up to hundreds of dollars a month. We conservatively estimate the combined revenue from selling these MATE attacks to range between $12.8M and $73.2M annually. To find out how game companies attempt to stop these attacks, we survey game cheat forums and experiment with the client-side anti-cheat solutions of 11 different popular competitive multiplayer shooter titles. We create a classification of defense techniques used by these anti-cheat solutions and create a benchmark to assess their technical sturdiness. Our findings suggest that prices for cheats are closely correlated to the technical sturdiness of the anti-cheat software they have to overcome. This correlation exceeds any other factor including, perhaps surprisingly, the popularity of the game targeted by the cheat. This shows that strong defenses against MATE attacks have a measurable real world impact.

CCS Concepts

• **Security and privacy** → *Software reverse engineering*; *Domain-specific security and privacy architectures*; *Software security engineering*; **Software and application security**;

Keywords

Game cheats, Anti-cheat, Man-at-the-end attacks

ACM Reference Format:
Sam Collins, Alex Poulopoulos, Marius Muench, and Tom Chothia. 2024. 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), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3689934.3690816

1 Introduction

Enforcing that a third party is behaving as expected is a difficult problem in cyber security. Technologies such as attestation, trusted execution, and DRM all address this issue in different ways. These systems consider the attacker to be a "Man-at-the-end" (MATE) with some level of control over the software and hardware being used [3]. In this paper we study the relevance, technical aspects, efficacy, and impact of widely deployed, yet often overlooked, category of software protecting against MATE attacks: game anti-cheats.

Over the last decades, video games have secured their place in the mainstream media landscape with a multi-billion dollar market expected to further grow in the foreseeable future [1, 21]. A significant share of this market are competitive online multi-player games, including widely known titles such as Fortnite or Counter Strike. While most people play these games fairly, a significant minority cheat, disrupting the experience for other players. To counter cheating and catch offending players, most competitive titles employ *anti-cheat* systems, which are either customised in-house solutions or third party products shared by multiple games [7].

To understand the relevance of anti-cheats, we survey cheat selling websites. We find cheats available for all games surveyed, sold as downloadable software packages on a subscription model, with prices ranging from $10 to $240 a month. Based on traffic data from similar-web and a standard e-commerce model, we analyse the activity of 80 cheat selling sites in Europe and North America. From this we conservatively estimate the number of people buying cheats on these websites as 30,000 - 174,000 per month. This represents a large number of people paying a significant amount of money to buy MATE attack software.

To find out how anti-cheats work we carried out a survey of game cheat discussion forums. These are public forums where experts discuss the technical details of anti-cheats and how cheats can counter them. Our survey results in a taxonomy of MATE defense techniques that are currently being deployed by game companies.

Additionally, we find that most of the leading Windows anti-cheat systems (unlike the cheats) run Microsoft signed, kernel-level code, with the most advanced anti-cheats mimicking the behaviour of bootkits to verify the integrity of the operating system's kernel starting from early boot [47]. While virtualization and introspection would be a powerful way to cheat, modern games often require high performance GPUs, meaning that they cannot effectively be fully virtualized on standard hardware. Despite this, cheats are easily available for all leading games, indicating that none of the current anti-cheat systems completely prevent cheating.

This leads us to define a novel sub-variant of the MATE attacker model, in which the attacker is the end user, running on a trusted operating system, without kernel level privileges. The attacker's

CheckMATE '24, October 14–18, 2024, Salt Lake City, UT, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1230-2/24/10
https://doi.org/10.1145/3689934.3690816

goal is to run a program while reading or altering its memory and function, which would typically be done by downloading distributed cheat software. The target program, and protections, may run with kernel level privileges. This is distinct from the general MATE attacker model in which the attacker may be assumed to have a higher level of control over the hardware and OS.

Based on our insights, we develop a series of grey-box tests to benchmark the strength of anti-cheat systems. We apply these to 11 popular competitive first and third person shooter games. Empirically confirming the results from our survey, we find that the most effective defenses run in the kernel from boot time and take steps to continually ensure kernel integrity. Weaker methods run at the user level and use process scanning and anti-debug techniques.

Last, to assess the impact of strong anti-cheat technologies, we analyze the prices for cheats for particular games across the most popular cheat selling sites. We find that while strong anti-cheats do not prevent the existence of cheats, their technical sturdiness is correlated with increased attacker costs and higher cheat downtime. We do not find a correlation between cheat price and factors such as game age or popularity. This indicates that in the case of game cheats strong MATE defenses are a driving factor in the cheating economy, increasing the price for available cheats and literally increasing the costs for attackers.

In summary, the contributions of this paper are:

- We show that there is a substantial market of game cheats, using a range of MATE attack methods.
- We survey cheat discussion forums to find what MATE defense techniques are used by anti-cheat solutions.
- We establish benchmark criteria to rank the strength of anti-cheats, and use this to test the most widely used anti-cheat systems.
- We show that technically sturdy anti-cheat solutions directly increase the market cost of cheats, so increasing the cost for attackers.

We provide our code for benchmarking anti-cheat strength and the raw data we collected from cheat selling sites at: https://github.com/SamCollins1327/Anti-Cheat_2024.

2 Background

2.1 MATE attacks

Man-At-The-End (MATE) attacks describe a strong attacker model in which a malicious actor has (physical) access to the victim device [3]. In contrast to a Machine-in-the-Middle attacker model, which places an adversary on the network and allows them to intercept and modify exchanged network traffic, MATE attackers are assumed to be able to tamper with the hardware or software of a target system without restriction. MATE attackers play a crucial role in the areas of Software Protection and Digital Right Management. In these scenarios, defenses have to consider "all-powerful" attackers who have control over the system on which the assets to protect are run. Thus, many defences against MATE attacks are not expected to prevent attacks completely, but rather to slow or deter an adversary, for instance via code obfuscation, deployment of anti-debug techniques, or remote attestation [8, 19, 22].

One category of MATE attacks are video game cheats, as acknowledged by the literature (e.g., [2, 4, 8, 14, 36]). However, these papers mention cheats as potential example for MATE *attacks*. In contrast, in this work, we set out to understand the role of anti-cheat solutions (i.e., the *defences*) used by modern video games.

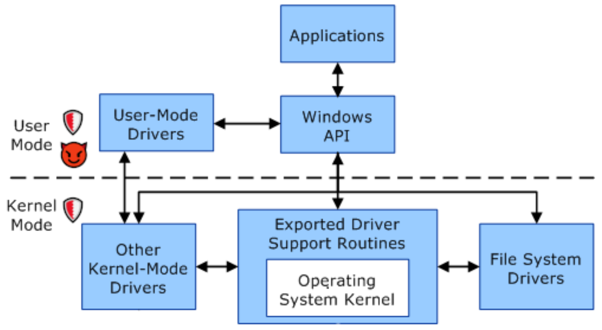

Figure 1: A diagram illustrating the communication between the user-mode and kernel-mode on Windows, adapted from [38]. Anti-cheat solutions may run both in user and kernel mode, while modern cheats begin their execution in user mode.

2.2 Windows Internals & Privilege Levels

The majority of cheats target games running on Windows and need to bypass the protections offered by the anti-cheat and the operating systems itself. A processor running windows supports two different access modes: usermode and kernel mode. Applications operate in usermode, while core system operations and device drivers operate in the kernel (i.e., *ntoskrnl.exe*[1]). The relationship between the two is illustrated in Figure 1.

User-mode applications are self contained, each with their own virtual address space in physical memory. Applications run in isolation, and can only interact with each other through specific means such as pipes, sockets, and Remote Procedure Calls (RPC). Therefore one process cannot access the memory of another without the correct handles or permissions. Moreover, if a process crashes, it crashes in isolation.

Kernel-mode processes run with fewer restrictions and share a single virtual address space. Kernel operations include management of system resources, such as the processor and memory, file system operations via File System Drivers, and device drivers. Since the virtual memory space is shared, one kernel program may write to the memory of another; if a program should crash, the entire system will crash in the infamous Blue Screen of Death (BSOD).

An important feature of Windows kernel security is Driver Signature Enforcement (DSE), introduced with 64-bit Windows Vista. This requires that any driver running in kernel mode must be signed in order to load, preventing malicious or poorly written code from entering kernel space. DSE can be disabled manually by booting into *test mode*, allowing drivers to be self signed and tested. This

[1]ntoskrnl.exe includes both the Windows executive and the so called "undocumented kernel". Their distinction is not important in the context of this work.

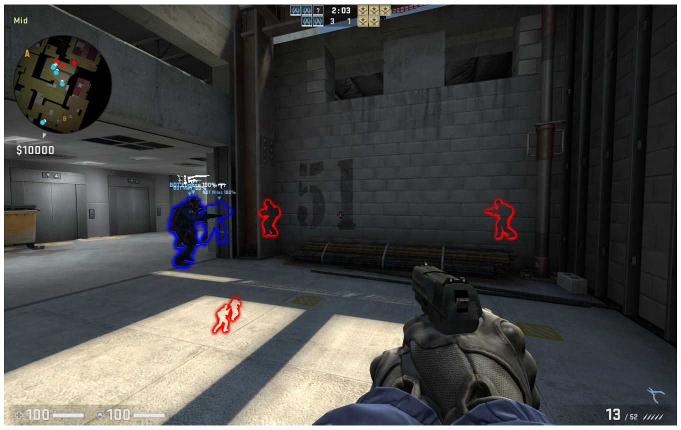

Figure 2: An example of an ESP cheat for the game Counter Strike: Global Offensive, adopted from [34]. The wallhack outlines enemies in red waiting in ambush behind the wall, and shows their locations on the mini-map, giving the cheater a large advantage.

also disables certain functionality related to Digital Rights Management[2]. Additionally, processes can check if test signing is enabled with functions such as `NtQuerySystemInformation`. Core kernel functions and structures are further protected by Windows Kernel Patch Protection (KPP) and will cause a BSOD if hooked or edited.

The restrictions on driver loading and kernel modifications are key security features for modern versions of Windows: Even as fully privileged user, full control over the system is not warranted.

2.3 Cheating in Competitive Shooter Games

Cheat types are highly dependent on the category of game they target. In this study, we focus on online, multiplayer shooter games that are dominated by two classes of cheats: ESPs and aimbots.

Extra-Sensory Perception (ESP) cheats are designed to grant an unfair advantage to players by providing them with supplemental information not ordinarily accessible through the standard player interface. They extract data from the game's memory or rendering pipeline, intercepting properties such as player locations, health, and equipped weapons. This extracted information may subsequently be displayed on both the game map and the player's screen. An especially common subcategory of ESP cheats are so called "wallhacks", which enables the player to see opponents through obstacles such as walls, as shown in Figure 2.

Aimbots automate the aiming process, resulting in instantaneous and precise targeting of opponents. Modern aimbots are highly configurable and allow, e.g., prioritization of opponents based on their health or proximity to the player. Additionally, it is noteworthy that traditional "rage aimbots" which instantly redirect the players crosshair have been largely replaced by "silent aim" configurations. In this case, the cheats are aiming to mimic a natural player's aiming process to avoid detection by other players or AI.

[2]To name some: high-bandwidth digital content protection (HDCP), Protected Media Path (PMP) , Media Foundation Protected Pipeline (MFPP), Encrypted File System (EFS)

Other types of cheats. While ESP cheats and aimbots are by far the most popular for online multiplayer shooter games, further types of cheats exist for this class of game, for instance so called "spinbots" or "triggerbots". However, these type of cheats are notably less widespread, which is why we exclude them from the scope of our study.

2.4 Cheat Development & Game Hacking Methods

Cheats are either 'internal', i.e., injected into the process and modifying game code in-vivo, or 'external', i.e., running in a separate process while reading and sometimes writing the game's memory externally, often providing overlays or synthetic input [20]. Cheat development often relies on traditional binary analysis techniques, described in brief detail in the following.

Static Analysis. is commonly used in cheat development and refers to the analysis of code without running it. Depending on information available, static analysis concerns source code, bytecode, or machine code, and may leverage reverse engineering frameworks (e.g., IDA Pro or Ghidra). The aims of static analysis for cheat development are straight forward: finding useful segments of code, offsets, missing checks, or vulnerabilities. In modern games, code obfuscation, packing, and virtualization make static analysis much more complex.

Dynamic Analysis. One of the more classic cheat development and execution techniques is (dynamic) memory scanning during the game's run time. Games store all their running information in memory, the same as any other process. Common memory scanning tools, such as Cheat Engine[3], allow their users to locate and modify game-critical values (e.g., hit points, enemy locations, and similar) in memory. Protecting against according modifications is a key task for modern client-side anti-cheat solutions. However, preventing *scanning* of memory in the general case is more challenging and often relies on finding and blocking the external process.

Cheat development may also rely on *debugging* the target game, allowing identification of game-critical logic and data with additional transparency. For instance, developers may analyse the stack layout at certain points in execution, set breakpoints, and carefully step through the running code to understand the game's behaviour. However, while more powerful, debugging is also more invasive and, hence, easier to detect and prevent by anti-cheat solutions.

Code Injection & Hooking. Code injection is a common technique used to insert and execute custom logic in the context of the target application. For instance, Dynamic Link Library (DLL) injection, is a classic way to implement and distribute "internal" cheats: relevant game data discovered via static and dynamic analysis can be automatically modified during run-time by an injected DLL. This DLL is then packaged and distributed to the users of a cheat, often along with a "launcher" to handle injection.

In contrast to plain code injection, which adds new stubs of functionality, hooking is a binary alteration technique that involves rerouting particular functions or system calls to change their functionality. An example of hooking in the context of game cheating

[3]https://www.cheatengine.org/

is vTable modification which makes it possible to re-implement (or augment) crucial functions of the game logic to enable the cheat.

3 Attacker Model

We characterise game cheats as a variant of MATE attacks, following our investigation of cheat discussion forums and selling sites in the following section.

First, we assume that the game, anti-cheat, and cheat are all running on some version of Windows OS, since it holds the lion's share of the PC gaming market, with 96.94% of Steam[4] users playing on Windows [6]. The cheater/attacker runs an unaltered copy of Windows, as providing a modified versions of the operating system are impractical for the distribution model of cheat selling sites. The anti-cheat code may be signed by Microsoft allowing it to run in the kernel, inspect any process and hook into system calls. Contrary, game cheats are not signed by Microsoft.

This gives the defender an advantage, and is distinct from the general MATE attacker model, in which the attacker is assumed to have full control over the hardware and software they run [3]. If the kernel-level protection on Windows worked correctly, and the game and anti-cheat were well written, the system should successfully mitigate MATE attacks (i.e., cheats). However, as we will show, this is not the case and we found game cheats available for all studied games.

The attacker's goal is to alter the running software by, for instance, directly reading or writing process memory. The defender aims to protect the confidentially and integrity of the games' memory, with a secondary goal of identifying and banning players attempting to cheat. Anti-cheats do not use trusted hardware, such as the SGX, as this would restrict who can run the game.

4 Analysing the Video Game Cheat Market

In this paper we suggest that game cheats and anti-cheats are an important form of MATE attack; that the attack methods developed by cheaters and the defense methods deployed by companies are state of the art and worthy of further study. Part of the reason for this is the money that can be made from selling cheats; with a clear financial incentive it makes sense for attackers to continually develop better cheats, and to counter these, games companies will need to deploy the best MATE defenses.

To understand the market in game cheats we survey cheat selling sites. In this section we estimate the size of the cheat selling market and in Section 6 we use the prices of individual cheats to help measure the impact that strong anti-cheat defenses have in practice.

We create a data set by scraping representative web sites selling and distributing cheats for popular games, and further analyse their operations as well as the legal aspects of game cheats. Additionally, based on our data, we estimate the market size and revenue generated by game cheats. Overall, our analysis indicates that anti-cheats may have a substantial financial relevance for the modern video game market.

4.1 Methodology and Dataset

Game & Cheat Selection. We focus on player vs player shooter games for PC, such as Call of Duty Warzone and Fortnite, due to their similarity in game mechanics, competitive nature, and commercial success. We selected the 13 most popular games in this genre, excluding the games "Escape from Tarkov" and "PlayerUnknown's Battlegrounds (PUBG)", as we cannot fairly compare them with the other games: The former heavily penalises players for loosing and the latter did not allow us to obtain figures for its PC vs mobile player base. This leaves us with 11 games in total.

Among our selected games, Fortnite has the youngest age demographic and is very popular with children [18]. Even when only considering adult players, 60% of Fortnite players fall within the 18-24 age range [43], although we anticipate this percentage to be even higher among PC Fortnite players. While cheats are not unheard of on consoles, they are notably less common. Consoles pose a greater challenge for cheat development and implementation. We observe that e-sports aspirations and competitive drive are less prevalent among console players - two factors influencing individuals to cheat.

We note that our study focuses solely on client software cheats. Hence, we exclude other methods of cheating, such as cheats requiring customized hardware [37] or doping with performance enhancing drugs [31] and detection methods such as server side AI pattern recognition.

Cheat Distribution Platforms. To construct an actionable dataset we first collated a comprehensive list of cheat sites. We first collected ~20 representative sites by searching google using keywords from our keyword list[5]. Then, we used the SimilarWeb's 'Similar Site' functionality to recursively expand this list until all found similar sites were already part of the data set. We only added cheat websites with more than 5000 hits a month on SimilarWeb and also excluded outdated and unmaintained sites. Furthermore, due to the different legal status of game cheats in China and Korea, and language issues, we excluded these sites from our dataset. This lead to 80 cheat sites, mostly hosted in Europe and North America.

For these 80 sites, we collect site name, average monthly traffic between August and October 2023 inclusive, the average price of a 1, 7, and 30 day cheat (where available), as well as the standard e-commerce conversion values to conduct a wider market analysis, as outlined in Section 4.4. Additionally, we sample 21 sites for a more detailed dataset, including individual cheat prices for the 11 selected games, uptime of cheats, and popularity of the respective games. We list these sites in Table 1 and use the data for a correlation analysis with anti-cheat strength in Section 7.

We publish all collected data as part of our repository[6].

4.2 Operational Aspects of Cheat Distribution Platforms

We found that all surveyed cheat distribution platforms are web sites with professionally organised marketplaces, typically selling a range of cheats for top games. We show a representative example of such a site in Figure 3.

Sales work on a subscription basis, with subscriptions varying from one day trials to three months. Accepted payment methods

[4]The largest PC game distributor, holding ~75-80% of EU and NA market share.

[5]https://github.com/SamCollins1327/Anti-Cheat_2024/blob/main/DataSets/Keywords.csv

[6]https://github.com/SamCollins1327/Anti-Cheat_2024/tree/main/DataSets

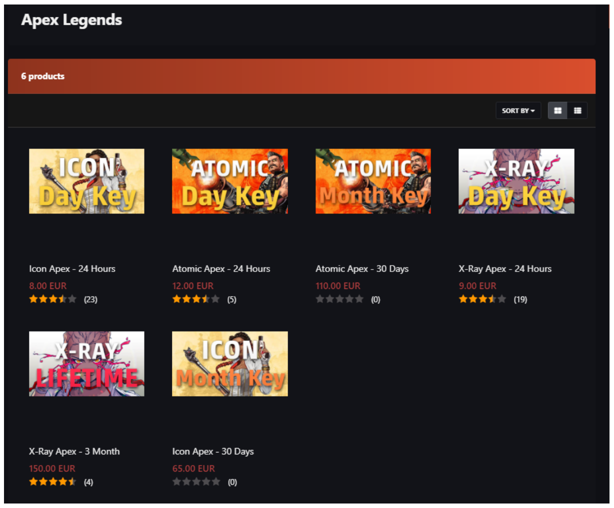

Figure 3: A Screenshot of the site Veterancheats from June 2024, showing available cheats for the game Apex Legends.

typically include credit/debit cards, a wide range of crypto currencies, or third party services such as Paypal or Stripe. Many sites further include a recommendation system for customer reviews, similar to contemporary e-commerce platforms such as Amazon or Ebay. Additionally, the majority of sites also include a status page showing whether a given cheat is currently expected to be working and/or detected by anti-cheat systems.

Almost all cheats sold are bundling different types of cheats together and include both ESPs and aimbots. Cheats mainly differed on how much information the ESP displays on the screen and the claimed reliability of the cheat. For example, some cheats explicitly note that they—allegedly—never had been detected previously.

Some platforms clearly develop and sell their own cheats, while others are pure distributors with multiple cheats of different origin available for the same game. Contrary to the underground economy related to "hacking", fraud, and cyber crime forums [11], we did not find any evidence of scams or malware on popular cheat selling sites. We believe that this is based on financial incentives: Much more money can be made from selling a regular cheat subscription than scamming potential customers. This is re-enforced by the reviewing systems on the cheat distribution platforms, as it enables an easy detection of untrustworthy cheat sources.

4.3 Legal Status of Game Cheats

If game cheats were clearly illegal then the best way to stop them might be to prosecute the cheat providers. If this was the case, technically strong anti-cheat solutions are less important. However, if legal action is not an effective option then games companies must develop and deploy technically strong anti-cheats. Hence, we investigate the legal aspects of cheats and previous lawsuits with cheat developers or distribution platforms as defendants.

Cheating in video games is not a crime in most countries, with the notable exceptions of South Korea and China, which have dedicated laws against using cheat software. Regardless of explicit laws, cheating does violate the terms of service of most games, which allows companies to withdraw services and ban cheating players. The lack of actionable laws in most countries results into cheat sellers operating without restrictions, as shown in the last section. Prices are advertised openly, payment is often taken via credit card, and distribution platforms are available on the public facing internet rather than, e.g., Tor hidden services.

So far, legal action against cheat sellers has mainly been based on claiming that the cheats were a "derivative work" of the targeted game and therefore infringes copyright protections. We could find no record of successful legal action against cheat sellers using laws targeting circumvention of protections, such as the US Digital Millennium Copyright Act (DMCA). The first legal action against developers of a commercially available cheating device (i.e., the Game Genie) was initiated by Nintendo in 1992 for copyright infringement [35]. The court found that the cheats were computer algorithms that did not include any of the plaintiffs code and, hence, did not infringe any copyright. More recently the company Blizzard has sued the—now defunct—cheat seller Bossland on copyright grounds [26]. First in Germany, where a court sided with the seller, ruling that the cheat did not qualify as copyright infringement. Then Blizzard repeated the case in the US where Bossland did not defend themselves and a judge gave a summary judgement against them, awarding Blizzard \$8 million in damages. Following this, Bossland shut down, suggesting that in this particular instance legal action was effective.

In a similar spirit, the games company Bungie successfully sued multiple sites selling cheats for their game Destiny 2, claiming that the cheats were a "derivative work" of their game and therefore infringe their copyright [49]. In a case against Lavi Cheats a judge agreed, awarding Bungie \$6.7 million in damages[7] [49]. However, in a second case against the Aim Junkies cheat site, a different judge dismissed the case, arguing that Bungie failed to demonstrate that the cheats were derivative work [39]. In 2023, Bungie further successfully sued Veteran Cheats and ring-1 [46]. Out of all the sites sued, only ring-1 continues to sell a Destiny 2 cheat, however all of the sites sued remain in business and continue to sell cheats.

Overall, our analysis shows that developing and selling game cheats is legally much safer than, for instance, writing malware, and that legal action against cheat distribution platforms may have some temporary effect on cheat availability, but is not effective in suppressing the existence of cheats altogether. Therefore, there is a clear need for games to be protected by technically strong anti-cheat/MATE defenses.

4.4 Market Size and Revenue of Video Game Cheats

To provide additional indicative data of the financial real-world relevance of a widely available type of MATE attacks, we estimate the size of the market in game cheats.

As no concrete sales figures or public accounts are available, we can only estimate the revenue of cheat sellers. We collect traffic data for each website and apply a standard conversion rates estimate[8]

[7] US law allows derivative work for comical purposes, therefore if the cheats had added elements of parody to the game they may have been exempt from copyright issues.
[8] I.e., the percentage of buyers over the total amount of visitors.

Table 1: Selected sites, their average monthly traffic between August and October 2023, the average price of a 30 day cheat on the site, and the min/max price of a cheat for 11 analysed games.

Site	Avg. mo. Traffic	Avg. 30 Day Cheat Price	Min. Price	Max. Price
Engine Owning	509,720	$13.80	$10.89	$19.59
Sky Cheats	197,463	$92.43	$35.00	$130.00
Battle Log	194,463	$72.84	$19.90	$145.75
Kernaim	189,338	$41.13	$16.50	$60.00
Lavi Cheats	153,429	$71.08	$29.00	$109.00
Interwebz Cheats	144,838	$21.79	$21.79	$21.79
Aimware	135,784	$19.16	$17.24	$22.99
Ring-1	115,353	$54.00	$29.00	99.00
Phantom Overlay	87,528	$32.546	$19.96	$43.24
Clutch Solutions	84,731	$41.89	$22.11	$59.95
AC Diamond	84,410	$81.33	$49.00	$120.00
Private Cheatz	78,166	$99.79	$24.99	$174.99
Aim Junkies	62,063	$44.12	$14.95	$99.95
Time2Win	54,415	$68.84	$29.99	$119.99
Veteran Cheats	45,452	$128.02	$22.11	$254.28
Wallhax	45,086	$15.00	$15.00	$15.00
Chod's Cheats	37,540	$28.96	$19.87	$38.05
Proofcore	36,408	$66.66	$49.99	$99.99
Cheat Army	31,100	$62.36	$17.50	$110.00
Perfect Aim	12,524	$17.87	$6.63	$35.38
Invision Cheats	10,757	$49.75	$11.06	$66.33

for e-commerce sites ranging from 0.7% to 4%, as suggested by Salesforce [15]. Sites sell access to a cheat for, e.g., one day, one week, thirty days, or 90 days. We apply the conversion rate to the mean monthly price of a cheat from each site in the following way:

$$\begin{aligned}\text{Site Monthly Revenue} = &\ (\text{Mean Monthly Price})\\ &\times(\text{Monthly Site Traffic})\\ &\times(0.7\%, 4\%)\end{aligned}$$

Considering the 80 sites of our data set, this model suggests a combined revenue generation between \$1.1M - \$6.1M per month, i.e., \$12.8M and \$73.2M annually. If we consider the number of people buying cheats, we estimate $30,000 - 174,000$ customers per month across all sites. Given that these figures do not include cheats brought via forums, Asian cheat sites, or cheats shared for free, we have a high degree of confidence that the actual number of people regularly cheating in video games is higher than our upper bound.

To verify that the 0.7% - 4% conversion rate figure used in our model applies to this market, we seek out for additional data points which may support or refute our estimations.

First, we investigated the Discord communities for a subset of the sites for indications of active customer count. We note that this approach is challenging as Discord does not allow explicit cheat-selling forums on its platform. Nonetheless, there are various loopholes depending on the discussed content in the community, but well-established Discord servers with accurately role-labeled members are not common. At time of our analysis, the most active Discord server tied to a cheat selling website was Sky Cheats. Upon our first inspection, this Discord had 12,167 total members. At the time, there were 1,826 members online, with 145 explicitly marked as non-customers, leaving 1,681 confirmed customers online. With our 0.7%-4% traffic model we estimate between 1045 and 5971 active customers for Sky Cheats. This puts active customers on Discord within the range of our web traffic estimate. However, many customers may have not been online at the time of our check. Additionally, "Cheat Army" also used Discord at the time of data collection and we observed 1188 active customers which is well within our traffic estimate of 1045-5971. Again we note that cheaters may not have join the Discord server in the first place. Therefore, we assume the true number of active customers to be higher in both cases.

Second, we draw from insights of our legal analysis presented in the last section. A particular interesting case is against one of China's top cheat selling sites. It estimated an income of at least \$350,000 per month [24], which is towards the higher end of our range estimate for the top sites included in our data set. In the case of Bungie vs Veteran Cheats [46], a subpoena against Stripe Inc., a third-party payment company used by Veteran Cheats, showed that 5,848 Destiny 2 cheats had been sold in 19 months from November 2020 to July 2022. This is higher than our estimate, whose upper value would have been 3,699 for this time period.

The size of this market suggests that game cheats will remain one of the most important form of MATE attacks. The financial incentives will mean cheat developers will put a lot of effort into discovering more inventive attack methods, and this will be countered by stronger anti-cheats from the games companies. This provides evidence for our view that the MATE community can learn a lot from studying cheats and anti-cheats.

5 Taxonomizing Anti-Cheat Technology

In this section, we will provide a technical analysis and classification of defenses against MATE attacks, as deployed by client-side anti-cheat engines.

5.1 Methodology

Given the limited amount of prior work on the topic of anti-cheat engines, we base our classification on a variety of rather unconventional sources of knowledge. We not only analysed representative anti-cheat solutions, but also studied blog posts and discussions on online forums on game hacking and cheating. More specifically, we again started with google searches using our list of keywords, but this time to find discussion forums and blogs rather than cheat

Table 2: Selected games, their anti-cheat solutions, and the privilege level of the anti-cheat solution.

Priv. Class	Game	Anti-cheat
User Mode	Counter-Strike 2	Valve Anti-Cheat (VAC)
	Team Fortress 2	Valve Anti-Cheat (VAC)
	Overwatch 2	Blizzard Defense Matrix
	Battlefield 1	FairFight
Kernel Mode	Apex Legends	Easy Anti-cheat (EAC)
	The Finals	Easy Anti-cheat (EAC)
	Fortnite	Easy Anti-cheat (EAC) & BattlEye
	Rainbow 6 Siege	BattlEye
	Battlefield 2042	EA Anti-cheat
	Warzone	Ricochet
Kernel Mode on Boot	Valorant	RIOT Vanguard

selling websites. We then selected five active and timely discussion platforms for further information retrieval: UnKnoWnCheaTs, GuidedHacking, OwnedCore, MPGH, and Secret.club. We searched these sites for the keywords between September 2022 and January 2024 to find relevant posts and discussions. We used the information gathered from these sites to establish a classification of anti-cheat techniques.

We note that our study focuses on *client-side* anti-cheating solutions. Thus, we refrain from assessing technically advanced server-side defenses such as machine-learning based behavioral analysis [5, 44]. Instead, we only consider local defenses and *simple* server-side solutions, such as bans based on hardware or account IDs. The complete list of investigated sites, the key words used for searching, and forum/blog references for each technical method listed are available in our repository.

Limitations: We note that this investigation is based on the expert knowledge available on the cheat forums searched. This provides enough information to bypass most anti-cheats, however it is possible that anti-cheats implementation additional protections not discussed on these forums. Data was only collected up until January 2024, and additional defense strategies may have become part of anti-cheat solutions after this.

5.2 Data Set

We analyse and evaluate the anti-cheat engine used by our 11 selected games (c.f., Section 4.1, in the time period between September and December 2023. We show the analyzed titles and their anti-cheat solutions in Table 2. We note that while some anti-cheats are in-house solutions and only used for particular games (e.g., Vanguard or Ricochet), others are commercially available solutions that are configurable by the game developers. Thus, the same anti-cheat solution can employ different protection measures based on their configuration, which is why we deliberately analyse the same anti-cheat multiple times in case they are used by different games.

We grouped games based on the *invasiveness* of the anti-cheat. The least invasive of anti-cheats run in user space alongside the game. This category includes Counter-Strike 2 (CS2) and Team Fortress 2 (TF2), which both use Valve Anti-Cheat (VAC), alongside Overwatch 2 and Battlefield 1. Of these, Battlefield 1 runs no distinct client-side anti-cheat, and therefore acts as a control in our experiment[9]. Our second category of selected games deploy kernel-level anti-cheat systems. Subsequently, the anti-cheat runs at a higher privilege level than the corresponding game. Notably, Fortnite runs two different solutions, Easy Anti-Cheat which is Epic Games' in-house solution also available as third-party solution for other studios, and BattlEye, a proprietary third-party anti-cheat solution first released in 2004. The third category is the most invasive. Here, the anti-cheat not only runs at kernel level, but is also loaded during system boot and runs constantly on the target system whether the game to be protected is played or not. At the time of writing, we only found Valorant, running RIOT Vanguard, deploying this technique[10]. Notably, this anti-cheat approach received significant backlash upon its initial introduction (e.g., [48]) due to security and privacy concerns.

5.3 Classification of Anti-cheat Techniques

Based on our survey of anti-cheat solutions, we classify the prevalent techniques deployed by classic and modern anti-cheat systems for identifying and catching cheating players. This classification will aid us to derive practical experiments to assess the effectiveness of popular anti-cheat systems. This also provides a list of MATE defense methods that are used by the games industry.

Player Identification. Most Anti-cheat systems aim to uniquely identify the current player to implement remediation strategies. If a cheater is caught, most game providers want to exclude the player from subsequent games.

P1 - Account ID. The most straightforward approach to identify players is based on their account ID. This way, players (and cheaters) can be identified across different systems. Account based suspension will lead to loosing game progress and can be a major motivator to avoid cheating.

P2 - Hardware ID. To enable more permanent ban types, game providers often take the hardware systems of cheating players into account. Otherwise, dishonest players could simply create new accounts after suspension and get back to playing. To enable device-based bans, modern anti-cheat often gather (and log) serial numbers from different components of the user hardware.

Integrity Checks. Anti-cheating technology aims to verify that the game under protection has not been tampered with. To this end, modern anti-cheat systems commonly deploy a mix of static and dynamic integrity checks.

I1 - File integrity checks are one of the most basic protections anti-cheats offer and aim to prevent static patching of game logic. Common implementations hash the game's files and compare these hashes to a stored or dynamically retrieved list of valid hashes.

I2 - Run-time memory checks. Complementing static file integrity checks, anti-cheats also try to protect the integrity of the process' memory at runtime. This involves both protecting memory regions

[9]The deployed anti-cheat engine "FairFight" provides server-side defenses only
[10]At time of writing Vanguard has now been ported to League of Legends, a popular MOBA title, but a completely different genre to shooters.

with standard OS-functions, like the guarded mutex system Vanguard employs, as well as systematically checking the validity of dynamic memory. These checks work in a similar way to file integrity checks, with periodic hashing and verification.

I3 - Code injection countermeasures. To further enable game integrity, anti-cheat systems may provide defenses against code injection. Particularly common are DLL injection countermeasures to protect against simple cheating approaches which leverage the Windows API for loading a DLL. Anti-cheats may register callbacks for or hook functions used for code injection (e.g., `LoadLibraryA()`), continuously check all loaded modules, search executable space for manually mapped modules, or debug the game to check for newly started threads.

Game Hardening. Games (and anti-cheats themselves) commonly use measures to complicate analysis of the game, which aims to slow down or prevent the development of cheats.

H1 - Obfuscation is a common technique used throughout the field of systems security to complicate reverse engineering. It aims to make code appear deliberately unclear and confusing while maintaining the functionality of the software. Common obfuscation techniques involve, for instance, packing, compression with non-standard algorithms, encryption, control-flow flattening, or instruction misalignment.

H2 - Anti-debugging measures formed the backbone of anti-cheat systems before the development of contemporary kernel mode systems in the mid 2000s, and is still prevalent today. Integrity checks and obfuscation makes static analysis and patching a long, arduous, and complex task; dynamic methods with a debugger have therefore always been a viable alternative for code analysis and manipulation. Therefore, anti-cheat systems often attempt to prevent debuggers from attaching, or purposefully crash the game when detecting their presence. Registering callbacks on newly opened handles then downgrading their privileges is a common technique, debug ports are checked, and even hypervisor based debuggers can be barraged with VM-Exits to reveal their presence.

H3 - Process Scanning. Scanning external running processes serves a dual purpose: identification of suspicious tools and the detection of cheats themselves. Most anti-cheat systems will enumerate through all running processes, introspect UI windows, and their associated child windows. The goal is to uncover external processes running cheats, reverse engineering tools, and cheat overlays.

Kernel-level protection. For additional hardening, detection, and protection capabilities, many modern anti-cheat systems moved to implementing kernel mode drivers, placing the anti-cheat at a higher privilege level than the game (and classic cheats). This makes altering game code, injecting DLLs, and even running memory scanners significantly more complex for cheat developers. On top of deploying the techniques outlined above, which can be enforced from kernel-level, kernel anti-cheats may additionally verify the integrity of kernel-space on the user's system.

K1 - Detecting test mode. If not properly mitigated, cheat developers may put themselves back on a level field with kernel anti-cheats using a strikingly simple technique: Enabling test signing in Windows. This feature allows the testing of driver code and provided a straightforward way for cheat developers to load "signed" kernel modules. Nowadays, most kernel-level anti-cheats engines will prevent the protected games from running if test signing mode is enabled.

K2 - Driver verification. Similar to recent cases in malware development [30], cheat developers have started to abuse Microsoft signed drivers with known vulnerabilities. Faults in these drivers allow cheat developers to manually map code into the kernel and, ultimately, bypass anti-cheat systems. As a countermeasure, anti-cheats may keep a list of known vulnerable drivers which they will block from loading, or refuse to open the game if loaded.

K3 - Detecting driver injection. To bypass anti-cheats running with kernel-level privileges, cheats will often require to run code at the same privilege level. The most straight-forward way to obtain kernel-level code privileges is the insertion of additional kernel modules containing the code required to carry out the cheat. Kernel anti-cheat systems commonly rely on the operating system's signing and verification services. If the driver is not signed then it cannot be loaded. Preventing the injection of unsigned drivers is often a multi-levelled approach and can include techniques outlined in *K2* and *K4.* Despite the clear need to find pre-mapped kernel code, preventing code injection altogether is the more viable approach for anti-cheat solutions, if possible.

K4 - Malicious code identification. A core limitation of most anti-cheat systems is that they can only execute verification tasks and integrity checks while they are loaded. While a single anti-cheat engine has moved to an "always-on" approach, where it is loaded at operating system boot time, most engines are still only started together with their corresponding game. Hence, if a cheat bypassing kernel-level protections is injected before the start of the game, verification attempts may fail, as the corresponding malicious drivers may be unloaded at time of verification. As a result, anti-cheats may continuously scan the kernel for executable code which isn't part of any properly loaded module, along with threads not backed by a signed module. This is often amended by scanning for signatures of known cheat drivers in the kernel space.

6 Technical Sturdiness of Anti-cheat Systems

Based on our classification of anti-cheat techniques, we can now design experiments to assess the technical sturdiness of representative anti-cheat systems. This allows us to understand the prevalence of MATE defenses in the video game industry and to create a ranking over the analysed anti-cheat solutions.

6.1 Test Definition & Benchmarks Used

We develop 14 experiments to benchmark the protection mechanisms popular anti-cheat systems deploy. Our experiments allow us to evaluate the individual protections in isolation and with levels of transparency commercial cheats cannot provide. Our tests aim to cover the most crucial *building blocks* of cheats and allow us to fairly compare the effectiveness of solutions; more exhaustive testing of individual implementations remains a topic for future work.

Most of our experiments map directly to one of the anti-cheat features discussed in subsection 5.3. For instance, to assess whether an anti-cheat carries out file integrity checks (*I1*), we modify relevant game files. The notable exceptions to this one-to-one mapping are the experiments around *H3 - Process Scanning* and *I3 - Code injection countermeasures*. In both cases, we develop additional tests to assess the feature with additional granularity. For *H3*, we run two distinct experiments with different processes opened, and for *I3*, we test different code injection techniques. Furthermore, we also refrain from testing *I2 - Run-time memory checks* directly, since we test the protections placed on accessing memory such as anti-debug.

In our experiments, we define a cheat attempt as prevented when either the attack fails immediately (e.g., handle permissions are stripped), or when it is detected by the anti-cheat and an observable reaction was handed out within the same game session (i.e., a crash or a ban). We consider the first type of prevention as *proactive*, and the latter type as *reactive*. Additionally, some games may detect a cheating attempt but deliberately delay punishment for the offending party to obfuscate which behaviour was picked up by the anti-cheat.

We provide full technical details for the individual experiments in Appendix A and the experiment code in the repository accompanying this submission[11].

6.2 Results

We carried out all experiments for each game in two scenarios: *(1)* in the game menu and, *(2)* during the ongoing game. For *(2)*, we allowed two full game lengths where applicable for the anti-cheat to react. As long our test accounts were not banned, we re-run the experiment whenever we were kicked from a match or the game crashed. This allowed us to test whether a second, subsequent detection merited in an identical result or an escalated reaction.

We showcase the results of the individual experiments and a categorical ranking of resulting anti-cheat strengths in Table 3. Our ranking consists of three primary divisions of strength, further subdivided by smaller differences recorded. Despite the direct detection/no-detection results from the experiments, we also consider the severity of response, which directly maps to the cost for a player upon detection. For example, during our experiments we observed that Overwatch 2 bans accounts for lighter offences than other titles and requires a new phone number per account, adding an additional hurdle for (repeating) cheaters.

As shown in Table 3, we observe only two systems in the top division, Valorant and Fortnite. For both games our experiments suggest full or partial prevention and detection of tested attack strategies. Additionally, Valorant deploys more severe responses and is, hence, ranked higher. Within the middle division, Battlefield 2042 and Rainbow 6 Siege deploy the strongest anti-cheats, with the only difference being imperfections in the process scanning approach of the latter. The Finals and Apex Legends reacted similarly to our experiments. This is unsurprising, as both games run the same third-party anti-cheat solution (i.e., Easy Anti-Cheat) and only showcase minor diversions, which we attribute to slightly different configuration of the solution. Between these two ranks Overwatch 2. Although this game does not rely on kernel-level detection capabilities, it performed well in our experiments and reacted with severe responses (e.g., it already issued bans for simple process injection attempts). At the lower end of the middle division, we find Warzone running the in-house solution Ricochet. Interestingly, despite being a kernel-level anti-cheat system, it was unable to detect any of our kernel-based experiments. Nonetheless, outside of detecting kernel-level cheating patterns, Ricochet detected most attack attempts. However, we found that it does not employ a strict ban policy and does not even institute hardware ID based bans[12].

The final subdivision constitutes three games: Counter-Strike 2 (CS2), Team Fortress 2 (TF2), and Battlefield 1. CS2 and TF1 both run an identical anti-cheat, developed in-house by Valve. They each cover a small base of protections, preventing the most basic injection and providing file integrity checks. Online sources suggest that CS2 uses a server-side machine learning algorithm to catch some cheats based off suspicious in-game behaviour, notably starting back in 2017 with spinbot detection [33]. However, we could not observe this in action during our experiments, which is why we rank both games at the same level. Battlefield 1, however, seems to perform almost no client-side anti-cheat detection whatsoever. From our experiments, we find that it only employs basic anti-debug functionality and file integrity checks. We suspect this is due to Battlefield 1's anti-cheat mostly relying on server-side statistical analysis [9], which, similar as for CS2, was not triggered by our experiments.

Limitations. We note that our ranking methodology is not free from limitations. First we do not account for signature scanning of cheats due to ethical concerns related to buying and running cheats in live game sessions. Therefore, anti-cheats that only ban if the detected code is a known cheat are inherently disadvantaged for our experiments. However, our experiments do account for signature scanning of known cheat tools like injectors and memory scanners.

Second, by focusing our study on client-side anti-cheat techniques, we exclude supplementary server-side features such as machine learning driven behavioral analysis or statistical thresholding. However, we argue that similar to how aimbots have evolved into 'silent aim', cheat developers can pick up on behavioral analysis trends and modify cheats so that detection based on such means as described above is less likely.

Lastly, our classification and experiments do not account for anti-cheats with robust report systems, outsourcing the cheat detection to other, human players. While such systems will not stop the cheater at the time of cheating, these systems may lead to a ban at a later point in time which may be bypassed with a Hardware ID spoofer or simply a new account. We argue that report-based solutions, while providing psychological benefits to benign players, are a somewhat weaker method for actual cheat detection and prevention.

7 The Effect of Technical Sturdiness of Anti-Cheats

Based on the insights provided from our market analysis, as well as the classification and ranking of anti-cheat solutions, we now analyse whether technical strong anti-cheats stops cheating or influences cheat availability or price. This directly provides us with

[11]https://github.com/SamCollins1327/Anti-Cheat_2024/tree/main/Anti-CheatTesting

[12]Cheat forums suggest that Ricochet does, however, use hardware ID to place suspected cheaters into "low trust" lobbies.

Table 3: Results of our anti-cheat strength benchmark, with games ranked by anti-cheat effectiveness.

Title	P1	P2	I1	I3-1	I3-2	I3-3	H1	H2	H3-1	H3-2	K1	K2	K3	K4	Rank
Valorant (K+)	◖	◖	◖	◖	◖	◖	◖	◖	◖	◐	◖		●	●	1.1
Fortnite (K)	◖	◖	◖	◖	◖	◖	◖	◖	◖	◖	◖		◖	●	1.2
Battlefield 2042 (K)	◖	◖	◖	◖	◖	◖	◖	◖	◖	◖	◖		◖		2.1
Rainbow 6 Siege (K)	◖	◖	◖	◖	◖	◖	◖	◖	◖		◖		◖		2.2
The Finals (K)	◖	◖	◖	◖	◖	◖	◖	◖	◖	◖	◖	◐		●	2.3
Overwatch 2 (U)	◖	◖	◖	◐	◖	○	◖	◐							2.4
Apex Legends (K)	◖	◖	◖	◖	◖	◖	◖	◖		◖	◖			●	2.5
COD Warzone (K)	◖		◖	◖	◖	◖	◖	◐		◖					2.6
Counter-Strike 2 (U)	◖		◖	◐											3.1
Team Fortress 2 (U)	◖		◖	◐											3.1
Battlefield 1 (U)	◖		◖					◐							3.2

Anti-cheat responses: ●: reactive prevention and banning of offending player, ◖ : proactive prevention, ◐: reactive prevention via crash
○: detection of offense with delayed ban, blank: no detection/prevention
Anti-cheat privilege: K+: kernel mode on boot, K: kernel mode, U: user.

insights whether the MATE defenses, as deployed by anti-cheats, prevent attacks or yield increased attacker costs.

7.1 Methodology

To assess the effect of technically strong MATE defenses for stopping game cheats, we gathered relevant data and calculated a set of pairwise correlations across multiple features, we consider any correlation above 0.5 to be *significant*.

$$Correlation(X, Y) = \frac{\sum(x - \overline{x})(y - \overline{y})}{\sqrt{\sum(x - \overline{x})^2 \sum(y - \overline{y})^2}}$$

We consider the technical anti-cheat strength, as measured in Section 6. We normalize their strength to a value between 0.1 and 0.9, as none of the studied systems is perfect or completely ineffective. Second, we investigate the price of each cheat. We collect this data by analyzing 21 cheat sites described in Section 4 and record the individual price of each cheat targeting a game included in our data set. We note that while some sites only offer one cheat per game, others offer multiple options with varying prices. We first accounted for each cheat entry individually, however the variation between individual sites caused significant interference in the correlations. Thus, we use the mean price of the cheats for a given game across all inspected site for our correlation analysis instead.

We further gathered information about cheat downtime, as explained below, and we consider the popularity of each game, using data provided by activeplayer.io. As a popularity metric, we chose monthly players and weight this value where necessary to discount non-PC players (e.g., for Fortnite, which is widely played on PlayStation and Xbox). Finally, we consider game age, in months since release as of March 2024, and cheat availability. To account for the fluctuations in available cheats per game per site, we measure the total amount of available cheats for a given game, rather than how many sites sell a cheat for that game.

7.2 Cheat Downtime

A primary function of an anti-cheat is to prevent cheating directly. Therefore, we collected data on the downtime of cheats, i.e., times at which a specific cheat is unavailable due to increased detection rates or changes introduced by the targeted game or its anti-cheat. Many cheat sites deploy status pages to inform customers whether cheats are in maintenance or run a higher risk of detection than usual. We developed scraper for 9 prevalent sites and monitored cheat up- and downtime for a period of 37 days [13]. We disregard cheats for which we have fewer than 7 days of information for.

Table 4 contains the average uptime of cheats for each game, along with the number of available cheats combined over the 9 sites. In this case, lower cheat uptime represents more downtime and, hence, hints towards a more effective anti-cheat solution. We want to note that the results for Team Fortress 2 are not conclusive, as we could only monitor a single cheat. For other titles monitored, with the exception of Rainbow 6 Siege, at least one cheat was up for 100% of the time. Aside from this we noticed a substantial overlap between the uptime of a cheat for a given title and the technical strength of its anti-cheat solution, as determined by our ranking in subsection 6.2.

Table 4: Measured combined uptime for cheats over a period of 37 days.

Game Title	Uptime (Mean)	Uptime (Median)	Uptime (St Dev)	#Cheats
Team Fortress 2	100%	100%	N/A	1
Battlefield 1	98.0%	100%	4.4%	3
Battlefield 2024	87.7%	100%	41.5%	9
Counter Strike 2	86.2%	97.1%	29.6%	11
The Finals	83.4%	94.1%	25%	17
Rainbow 6 Siege	83.3%	88.6%	15.4%	16
Apex Legends	77.2%	97.1%	34.8%	25
Overwatch 2	73.8%	100%	40.9%	10
COD Warzone	72.4%	100%	45.3%	15
Fortnite	69.8%	85.3%	30.5%	29
Valorant	50%	52.9%	37.2%	17

[13] Individual cheat statuses can be found at https://github.com/SamCollins1327/Anti-Cheat_2024/blob/main/DataSets/status_results.txt

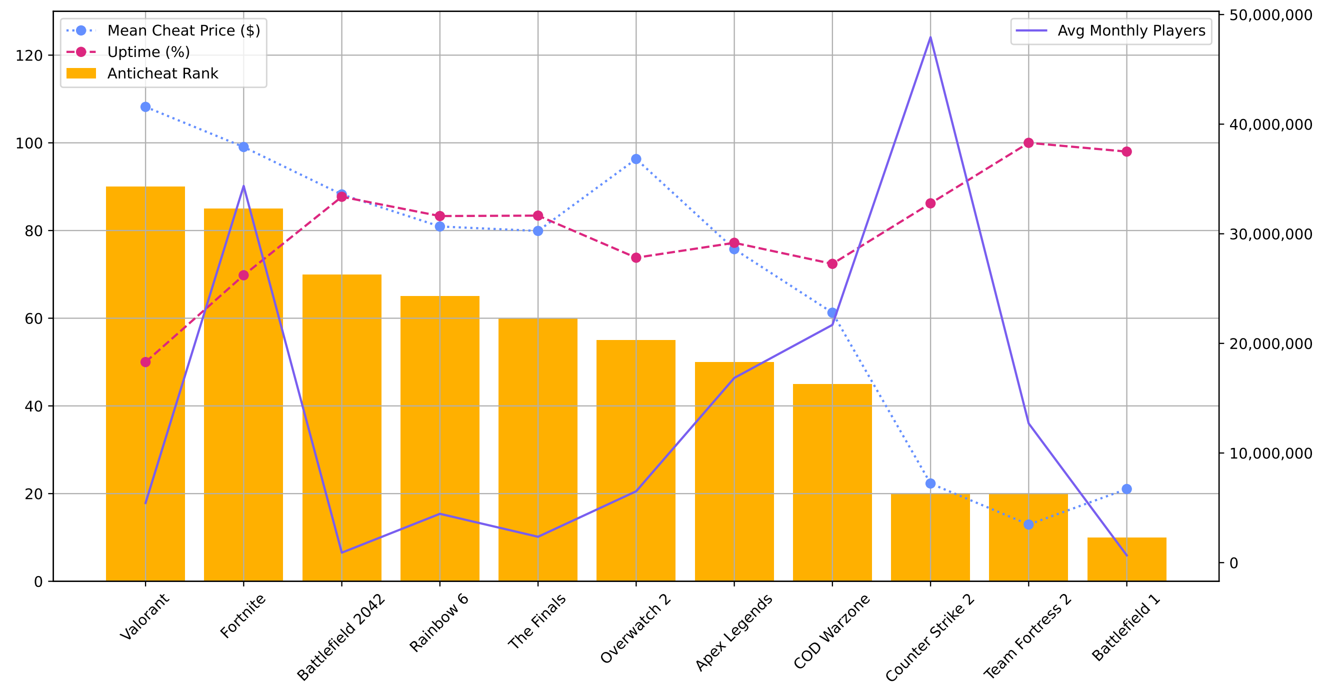

Figure 4: Uptime, mean price, popularity, and anti-cheat rank for analyzed games. Games are ordered by anti-cheat rank.

Table 5: Full set of correlation data. Correlations with an absolute value above 0.5 are highlighted in bold.

	Price	Anti-cheat	Uptime %	Players	Game Age	Availability
Price	1					
Anti-cheat	**0.948**	1				
Uptime %	**-0.771**	**-0.765**	1			
Players	-0.101	0.023	-0.373	1		
Game Age	0.443	-0.297	0.414	-0.145	1	
Availability	-0.398	0.358	-0.49	0.476	-0.364	1

We also observe clear events in our data, which often coincide with game or anti-cheat updates. For instance, representative events we observed are hotfix 1.4.1 and patch 1.5.0 for The Finals. For the first, half of the cheats monitored went down within a day of the patch. A few days later with patch 1.5.0, which mentioned upgrades to the anti-cheat policy, five cheats went temporarily offline from a period of two to seven days before being brought online again.

7.3 Results

We show the results of our correlation analysis across the different features in Table 5. We further visualize core parts of this analysis in Figure 4. The correlations imply various relationships of different strengths; we more closely explore two strongest correlations evident. First we note a significant positive correlation (0.948) between the price of a cheat and the strength of the anti-cheat. This result suggests that robust anti-cheat systems do in-fact increase the price of cheats (i.e., the actual cost for an attacker). Our intuition is that a technically strong anti-cheat is harder to bypass, leading to greater development costs, and, therefore, a higher end price.

Second, we observe a strong negative correlation (-0.765) between anti-cheat strength and cheat uptime. This suggests that as the strength of an anti-cheat increases, the less time cheats for that title are up and running. In other words: strong anti-cheats make cheating more expense for the attacker and less likely to be functioning at all times.

8 Discussion

Anti-cheats as MATE defenses. In our analysis of anti-cheat solutions, we found widespread use of integrity checking methods (such as file and memory integrity checks) and hardening methods (such as obfuscation). Such methods have been widely discussed in previous work on MATE defenses (e.g., [2, 4, 8, 14, 36]) and our study provides evidence that they are used, and useful, in practice.

Beyond these methods we found that many anti-cheat systems use signed code running at the windows kernel level, giving them an advantage over unsigned cheat code. We also found that cheats had to be easy to distribute meaning that, for instance, requiring low level alterations to the OS is not an option for the attacker. This leads to a more restrictive MATE attacker model than models considered in past work. This attacker model, and the use of kernel level protections may be useful when trying to stop MATE attacks in other areas.

We further found wide spread use of attempts to identify and ban cheaters, which is another approach to defend against MATE attacks which has not widely been studied before. Hardware bans increase the risk of trying out possible attacks, and they also make studying the system and developing cheats more difficult, which is overall increasing the costs for an attacker.

Effectiveness of kernel-level anti-cheats. We showed a strong correlation between the use of kernel level protections and the price and the downtime of cheats, suggesting that kernel level protections provide the most effective defense. However, one user level anti-cheat (for Overwatch 2) scored well in our bench marking, and has high priced cheats, suggesting that other factors are also important for the strength of the defense.

Threats to validity. In our market analysis we apply the standard e-commerce conversion rate, this being the best estimate for traffic to sales. While game cheats are not a *black market*, such as various sites running on onion domains, it is a *grey* one; cheats are not explicitly illegal, but sites have been successfully sued. This may effect the validity of the standard conversion rate.

Additionally, since we gathered much of our information from user forums and blogs, we cannot grantee that these methods are used in real cheats. Responses from anti-cheats to our grey box testing shows that these methods are at least defended against, further suggesting their use in real cheats.

Last, we deliberately do not account for server side checks in our methodology, such as machine learning driven in-game behaviour analysis. Thus, we cannot access their influence on a technical strength of an anti-cheat. Similar, we use dedicated tests which are not captured by signature scanning. Thus, we only access the strength of individual defences and not the full end-to-end detection capabilities of anti-cheats.

9 Related Work

Cheating in Video Games. Prior work, such as [25, 28, 29], have discussed game cheats at a high level, outlining the kind of cheat methods that could be used in general terms, without going into technical details of the defenses or assessing their effectiveness.

Multiple studies investigate why people cheat in video games, (e.g., [12, 16]), some suggesting that it is good for mental health [41], and other work suggesting that it leads to crime [27], however such speculation is not the focus of our paper.

Pontiroli investigates game cheats from the anti-virus point of view [42] and how they compare to malware. Karkallis et al. [32] investigate the injector tools shared by the cheating community, while Tian et al. [45] study how cheats for mobile games work. Bursztein et al. [13] present an automated map hack system for real time strategy games, and a protection method for games based on distributing game state between players. However, none of these works systematically analyze contemporary anti-cheat systems or the overall cheat market.

Anti-Cheat Solutions. Technical discussion forums discuss the inner workings of anti-cheat systems in great detail. These sites are aimed at people that want to understand the cheat development and anti-cheat bypasses, rather than providing cheats to a mass audience. Such sites contain the best public sources of information about how anti-cheat systems operate.

Some past work has suggested designs for anti-cheat systems based on trusted hardware [10, 23, 40]. Anwar et al. [7] used memory access graphs to detect pointer chains used by cheats and selectively obfuscate key values. Choi et al. [17] have proposed a client side aimbot detection *BotScreen* which leverages a machine learning model to detect abnormal mouse movements. However, as our examination of anti-cheat system shows, these techniques are not used in practice yet. To the best of our knowledge, no prior work extensively analyzed how client-side PC anti-cheat systems work, ranked their relative effectiveness, or investigated their effects on the market for game cheats.

10 Conclusion

In this paper we have studied game cheats as a form of MATE attack. We study the market of game cheats, showing how this substantial grey market operates and estimating its yearly revenue in Europe and North America.

We establish a clear attacker model for PC game cheats on Windows, backed by information gathered from game hacking forums. We use this information to construct grey box tests for the most salient anti-cheat techniques, then apply these tests to 11 of the most popular FPS and Battle Royale games to create a ranking of the technical strength of different anti-cheat solution.

Based on our results, we then analyse the driving factors of cheat prices. Most importantly, our results suggest that the technical sturdiness of anti-cheat solutions has a large effect on the price and uptime of cheats and, hence, a significant real world impact.

References

[1] Video game market. report code 3218, 2023.
[2] Abrath, B., Coppens, B., Broeck, J. V. D., Wyseur, B., Cabutto, A., Falcarin, P., and Sutter, B. D. Code renewability for native software protection. *ACM Trans. Priv. Secur. 23*, 4 (aug 2020).
[3] Akhunzada, A., Sookhak, M., Anuar, N. B., Gani, A., Ahmed, E., Shiraz, M., Furnell, S., Hayat, A., and Khurram Khan, M. Man-at-the-end attacks: Analysis, taxonomy, human aspects, motivation and future directions. *Journal of Network and Computer Applications* (2015).
[4] Akhunzada, A., Sookhak, M., Anuar, N. B., Gani, A., Ahmed, E., Shiraz, M., Furnell, S., Hayat, A., and Khurram Khan, M. Man-at-the-end attacks: Analysis, taxonomy, human aspects, motivation and future directions. *Journal of Network and Computer Applications 48* (2015), 44–57.
[5] Alpha AI. How ai is eliminating cheaters in gaming. https://medium.com/@Alphapack24/how-ai-is-eliminating-cheaters-in-gaming-da706146f7bf#:~:text=Cheaters-,1.,thousands%20of%20cheater%20bans%20daily., 2024.
[6] Alsop, T. Distribution of steam gaming platform users operating systems used as of september 2023.
[7] Anwar, M. S., Zuo, C., Yagemann, C., and Lin, Z. Extracting threat intelligence from cheat binaries for anti-cheating. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)* (2023).
[8] Basile, C., De Sutter, B., Canavese, D., Regano, L., and Coppens, B. Design, implementation, and automation of a risk management approach for man-at-the-end software protection. *Computers & Security 132* (2023), 103321.
[9] Battlefield Wiki. FairFight, 2019. (Online. Accessed 2024-21-02).
[10] Bauman, E., and Lin, Z. A case for protecting computer games with SGX. In *Proceedings of the 1st Workshop on System Software for Trusted Execution* (2016).
[11] Benjamin, V., Li, W., Holt, T., and Chen, H. Exploring threats and vulnerabilities in hacker web: Forums, irc and carding shops. In *2015 IEEE international conference on intelligence and security informatics (ISI)* (2015), IEEE.
[12] Boldi, A., and Rapp, A. "is it legit, to you?". an exploration of players' perceptions of cheating in a multiplayer video game: Making sense of uncertainty. *International Journal of Human–Computer Interaction* (2023).
[13] Bursztein, E., Hamburg, M., Lagarenne, J., and Boneh, D. Openconflict: Preventing real time map hacks in online games. In *Proc. of IEEE Security and Privacy* (2011).

[14] Canavese, D., Regano, L., Basile, C., Coppens, B., and Sutter, B. D. Man-at-the-end software protection as a risk analysis process, 2022.
[15] Chervalie, S. Online shopping conversion rate in selected verticals worldwide in 2nd quarter 2022. *Statista [online]* (2022).
[16] Cho, S. A community-based investigation of competitive cheating. In *CHI PLAY* (2022), Association for Computing Machinery.
[17] Choi, M., Ko, G., and Cha, S. K. {BotScreen}: Trust everybody, but cut the aimbots yourself. In *32nd USENIX Security Symposium (USENIX Security 23)* (2023), pp. 481–498.
[18] Clement, J. Share of children who play fortnite in the u.s. 2018.
[19] Collberg, C., Davidson, J., Giacobazzi, R., Gu, Y. X., Herzberg, A., and Wang, F.-Y. Toward digital asset protection. *IEEE Intelligent Systems 26*, 6 (2011), 8–13.
[20] Čurda, T. Analysis and detection of online game cheating software. Tech. rep., Masaryk University, 2014.
[21] Engelstätter, B., and Ward, M. R. Video games become more mainstream. *Entertainment Computing 42* (2022), 100494.
[22] Falcarin, P., Collberg, C., Atallah, M., and Jakubowski, M. Guest editors' introduction: Software protection. *IEEE Software 28*, 2 (2011), 24–27.
[23] Feng, W.-c., Kaiser, E., and Schluessler, T. Stealth measurements for cheat detection in on-line games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games* (New York, NY, USA, 2008), NetGames '08, Association for Computing Machinery, p. 15–20.
[24] Franceschi-Bicchierai, L. Inside the 'world's largest' video game cheating empire, https://www.vice.com/en/article/93ywj3/inside-the-worlds-largest-video-game-cheating-empire. *Motherboard, Tech by Vice* (2021).
[25] Gjonbalaj, A., Chen, J., Demicco, D., and Prakash, A. Cheating in esports: Problems and challenges. In *2023 IEEE Gaming, Entertainment, and Media Conference (GEM)* (2023), pp. 1–6.
[26] Good, O. S. Blizzard wins $8 million judgment against overwatch cheat maker. In *Polygon* (2017).
[27] Heubl, B. Gaming - hacking. when cheating leads to crime. *Engineering & Technology 15* (2020).
[28] Jeff Yan, J., and Choi, H. Security issues in online games. *The Electronic Library, Vol. 20 No. 2, pp. 125-133.* (2002).
[29] Jeng, A. B., and Lee, C. L. A study on online game cheating and the effective defense. In *Recent Trends in Applied Artificial Intelligence* (Berlin, Heidelberg, 2013), M. Ali, T. Bosse, K. V. Hindriks, M. Hoogendoorn, C. M. Jonker, and J. Treur, Eds., Springer Berlin Heidelberg, pp. 518–527.
[30] Kálnai, P., and Havránek, M. Lazarus & byovd: Evil to the windows core.
[31] Kaluarachchi, C. D., Wickramatunga, C. A., and Dewapriya, D. The dark side of e-sports: The role of player emotions and cyberbullying in moba. *International Conference on Information Systems (ICIS)* (2023).
[32] Karkallis, P., Blasco, J., Suarez-Tangil, G., and Pastrana, S. Detecting video-game injectors exchanged in game cheating communities. In *Computer Security – ESORICS 2021* (2021), E. Bertino, H. Shulman, and M. Waidner, Eds.
[33] Kotwani, B. What is vacnet, a deep learning product of cs:go's overwatch?, https://www.talkesport.com/news/what-is-vacnet-a-deep-learning-product-of-csgos-overwatch/. *TalkEsport* (2020).
[34] Kumar, A. Csgo wallhack golang. https://github.com/aditkumar1/csgo-wallhack-golang, 2022.
[35] Lewis Galoob Toys, Inc. v Nintendo of America, Inc. 964 F.2d 965 (9th Cir.). https://law.justia.com/cases/federal/appellate-courts/F2/964/965/341457/, 1992.
[36] Manikyam, R., McDonald, J. T., Mahoney, W. R., Andel, T. R., and Russ, S. H. Comparing the effectiveness of commercial obfuscators against mate attacks. In *Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering* (New York, NY, USA, 2016), SSPREW '16, Association for Computing Machinery.
[37] Mayra Rosario Fuentes and Fernando Mercês. Cheats, Hacks, and Cyber-attacks. Threats to the Esports Industry in 2019 and Beyond. Tech. rep., Trend Micro, 2019.
[38] Microsoft. Windows kernel-mode vs user-mode. https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode, 2023.
[39] Orland, K. Judge dismisses "insufficient"' copyright claims in destiny 2 cheating case, https://arstechnica.com/gaming/2022/05/judge-dismisses-insufficient-copyright-claims-in-destiny-2-cheating-case/. *Ars Technica* (2023).
[40] Park, S., Ahmad, A., and Lee, B. Blackmirror: Preventing wallhacks in 3d online fps games. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2020), CCS '20, Association for Computing Machinery, p. 987–1000.
[41] Passmore, C. J., Miller, M. K., Liu, J., Phillips, C. J., and Mandryk, R. L. A cheating mood: The emotional and psychological benefits of cheating in single-player games. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play* (2020), CHI PLAY '20.
[42] Pontiroli, S. The Cake is a Lie! Uncovering The Secret World of Malware-Like Chears in Video Games. *Virus Bulletin* (2019).
[43] Rohit Shewale. Fortnite Statistics For 2024 (Active Players, Revenue & More), 2023. (Online. Accessed 2024-21-02).
[44] Spijkerman, R., and Marie Ehlers, E. 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* (New York, NY, USA, 2021), CIIS '20, Association for Computing Machinery, p. 87–92.
[45] Tian, Y., Chen, E., Ma, X., Chen, S., Wang, X., and Tague, P. Swords and shields: A study of mobile game hacks and existing defenses. In *ACSAC '16* (2016).
[46] Vernace, M. Bungie vs. cheaters: The sequel. In *thegamepost.com* (2023).
[47] vmcall. Why anti-cheat software utilize kernel drivers, 2023. (Online. Accessed 2024-22-02).
[48] Wilde, T. The controversy over Riot's Vanguard anti-cheat software, explained, 2020.
[49] Wood, A. Bungie wins another lawsuit against destiny 2 cheat-makers. In *IGN.COM* (2023).

Appendices

A Test Descriptions

In the following, we list our bench mark experiments to assess the presence of different anti-cheat approaches.

Player Identification

P1 - Account Bans. We check that each service bans players if they are detected to be cheating. While this is a baseline test, it is important for completeness. In some cases we were not banned from games even following our "best efforts". Thus, we also include second hand information from the provider as results here.

P2 - Hardware ID. We collect a list of the Hardware ID (HWID) information gathered by each anti-cheat used in its HWID based bans via the Windows Management Instrumentation (WMI) trace functionality. We also consider how detailed the collected information are in our ranking.

Integrity

I1 - File Integrity. We assess weather or not the anti-cheat runs file integrity checks. This should be one of the most basic defenses run. We test this by patching the game binary via IDA and then running the game.

I2 - Run-time memory checks. Omitted from testing in isolation.

I3 - Code injection. Our testing against code injection countermeasures for each anti-cheat solution is subdivided into three tests:

(1) We attempt standard code injection, by allocating memory within the process then calling `LoadLibraryA` on our DLL using `loadlibraryinjector.exe` and `Dll1.dll`.
(2) We attempt manually mapping our code into the game process using `Extreme Injector v3.exe` and `Dll2.dll`.
(3) We try injection via thread hijacking using `Extreme Injector v3.exe` and `Dll3.dll`.

In all cases we injected harmless "hello world" code and ran a game in case injection succeeded.

Game Hardening

H1 - Obfuscation. We manually check if important game files are obfuscated in any way. To do this we use binwalk to measure and record the entropy of the main PE file for each game.

H2 - Debugging. To check for the presence of simple anti debug techniques we attach GDB to the running game process. Additionally, if we can attach Cheat Engine to the game process, we then also attach the Cheat Engine debugger. In all cases the reaction to both debuggers was identical. We note a clear direction for future work in this test case would be to measure reactions to more advanced debuggers, such as scyllahide and titanhide for x64dbg, as well as hypervisor based debuggers like hyperdbg.

H3 - Process scanning. We test weather the anti-cheat is enumerating running processes and open windows. This is further divided into two sub-cases for two common reverse engineering tools: (1) IDA, and (2) Cheat Engine. Thus, we define two subtests:

(1) We open the IDA static analysis tool, load an executable not associated with the game, then run the game. We see if the game opens or not and if any warnings are given. This is to test if the anti-cheat is checking open processes, and flags the use of IDA as a possible threat.
(2) We start Cheat Engine and then open the game. If the game opens we attach Cheat Engine to the game process. Cheat Engine is a dynamic analysis tool for which the game must be running.

Kernel-level protection

K1 - Test Mode. We test if the game will run on Windows with test signing enabled (`Bcdedit.exe -set TESTSIGNING ON`). If allowed, this would allow an attacker to load their own unsigned kernel drivers to arbitrarily cheat in the game.

K2 - Vulnerable Drivers. We load the widely known to be vulnerable `iqvw64e.sys` driver on the target machine before opening and playing the game. We use a modified version of KDMapper (`kdmapper_no_unload.exe`) which does not unload the vulnerable driver post injection. This tests whether the anti-cheat will check loaded drivers for known vulnerable software.

K3 - Kernel Injection. We first open the game, and then use KDMapper (`kdmapper.exe`) and `iqvw64e.sys` to inject an entry-only driver into the kernel (`HelloWorld_EntryOnly.sys`) This checks if the anti-cheat can detect an injection at the kernel level during run time, and if it takes any action regardless the content of the driver.

K4 - Module Detection. Using KDMapper (`kdmapper.exe`) and `iqvw64e.sys`, we inject a more advanced malicious driver into the kernel (`HelloWorld_WithThread.sys`) before starting the game. This driver starts a single thread, closes the command line and unloads `iqvw64e.sys`. With this test, we evaluate weather an anti-cheat can locate the manually mapped driver without the vulnerable kernel driver present or having observed the injection.