TU WIEN Informatics

Video Games as an Attack Vector

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Matthias Glinzner
Matrikelnummer 00726342

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing.in Dr.in techn. Martina Lindorfer, BSc

Wien, 16. August 2025

Matthias Glinzner

Martina Lindorfer

TU WIEN Informatics

Video Games as an Attack Vector

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Matthias Glinzner
Registration Number 00726342

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing.in Dr.in techn. Martina Lindorfer, BSc

Vienna, August 16, 2025

Matthias Glinzner

Martina Lindorfer

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Matthias Glinzner

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 16. August 2025

Matthias Glinzner

Danksagung

Großer Dank ergeht an Martina Lindorfer für Betreuung, Rat und Unterstützung.

Acknowledgements

I would like to thank Martina Lindorfer for her advice, input, and support.

Abstract

This work examines cyber attacks leveraging security flaws in video games installed on consumer systems. While there exist numerous scientific works on other aspects of cybersecurity, this specific scenario – video games as an entry point for malicious actors – seems curiously absent from research. What makes this especially puzzling is the fact that the video game industry is one of the fastest growing markets of the last ten years. The thesis' premise is the assumption that detailed study of security mechanisms and -flaws of video games might shed much needed light on their significance regarding the underlying system's integrity.

To show this, I first describe possible attack scenarios and give examples for each of them; I then combine them to give an overview of the potential attack surface of video games. To this end I chose a holistic approach: In addition to engines (a game's core component), the focus is gradually shifted to also contain other elements essential to a game's infrastructure. These include distribution platforms and developers, cheats and cracks (together with their ecosystems), as well as the modding scene. Further, I formulate a threat model in accordance with the aforementioned vulnerabilities and give a proof of concept. The goal is to utilize this practical example to lend real weight to earlier theoretical findings.

The work's result is that video games add significantly to a system's attack surface. Given examples show that vulnerabilities in games are more than the sum of their parts' security flaws; in combining otherwise harmless components, possibilities for exploits present themselves as a result of the interaction between them. Additionally, the ease with which a working prototype for breaching a game's security can be produced is cause for concern. The initial assumption is confirmed: Video games present a considerable risk to consumer systems and should figure more prominently within the field of security research.

Kurzfassung

Inhalt dieser Arbeit sind Cyberangriffe auf Verbrauchersysteme, die sich Schwachstellen in Videospielen und deren Infrastruktur zunutze machen. Während zahlreiche andere Aspekte der Cybersicherheit in wissenschaftlichen Arbeiten thematisiert werden, ist dieses Szenario – Videospiele als Einfallstor für bösartige Akteure – auffallend wenig behandelt. Das ist umso verwunderlicher, als die Videospielindustrie eine der schnellstwachsenden der letzten zehn Jahre darstellt. Ausgangspunkt des vorliegenden Werks ist die Vermutung, daß gründliches Studium der Sicherheitsmechanismen und -schwächen von Spielen deren Signifikanz für die Sicherheit des zugrundeliegenden Systems in ein klareres Licht zu rücken vermag.

Zunächst zeige ich mögliche Angriffsszenarien anhand von Beispielen auf und fasse sie schließlich zu einem Überblick der potenziellen, durch Spielesoftware bedingten Angriffsfläche zusammen. Hierbei wähle ich eine holistische Herangehensweise: neben der Kernkomponente Engine behandle ich in einem sich sukzessive erweiternden Blickwinkel auch essenzielle Infrastrukturelemente. Diese beinhalten Vertriebsplatformen und Entwickler, das Cheat- und Crack-Ökosystem findet ebenso Erwähnung wie die Modding-Szene. Überdies schildere ich ein Bedrohungsmodell, abgeleitet aus der Summe der zuvor aufgezeigten Schwachstellen, und veranschauliche es durch einen Prototypen. Ziel ist, den vorausgehenden theoretischen Beobachtungen durch ein praktisches Beispiel reales Gewicht zu verleihen.

Es stellt sich heraus, daß die durch Videospiele verursachte, zusätzliche Angriffsfläche signifikante Größe hat. Die untersuchten Beispiele lassen erkennen, daß Sicherheitslücken in Spielen mehr als die Summe der Schwachstellen ihrer Teile sind; oftmals treten Kombinationseffekte zutage, die separat betrachtet harmlose Komponenten im Zusammenwirken zum Sicherheitsrisiko werden lassen. Auch scheint die Leichtigkeit, mit der sich ein Prototyp, der bösartigerweise Schwachstellen ausnützt, herstellen lässt, bedenklich. Jedenfalls bestätigt diese Arbeit die Ausgangsvermutung: Videospiele stellen ein zu beachtendes Sicherheitsrisiko für Verbrauchersysteme dar und sollten dementsprechend in der Sicherheitsforschung figurieren.

Contents

CHAPTER 1

Introduction

The video game industry is one of the fastest growing markets in the past ten years. In 2024 its size was estimated at 298.98 billion US-dollars, with a projected growth to reach over 600 billion dollars by 2030 and a compound annual growth rate of 12.2%. Mobile gaming represents the biggest segment, forecast to grow by more than 15% from 2025 to 2030. However, the rise of Esports as a strong competitor for sports viewership shows that traditional gaming platforms (PCs and consoles) are far from being obsolete [1]. The global number of people using a PC for video gaming rose from 1.75 billion in 2020 to a predicted 1.86 billion in 2024. This is accompanied by game developers' preferences: 65% were working on a PC game in 2023 [2]. In terms of operating system, the majority of PC games are developed for Microsoft Windows, followed by macOS and Linux. The distribution platform *Steam* offers 196945 games for Windows (68%), compared with 54577 (19%) and 37614 (13%) for macOS and Linux, respectively [3]. With growing numbers and a wide distribution, PC games pose a potentially underexplored security risk. Video games make for a potent vector of attack:

- they often have extensive permissions,
- they naturally interact with many different subsystems (from peripherals to graphics pipeline to network adapter),
- their infrastructure consists of many additional technologies (like launchers or anti-cheat measures) that can come under attack as well,
- and attacks may go undetected as users are used to crashes and strange behavior.

Installing software on a Windows system requires administrator rights. Unlike standard user accounts, administrator accounts have full control over the device; this includes manipulating files, directories, and services, modifying user accounts and their permissions,

as well as taking control over local resources at any time. Further, they may run applications with elevated permissions, for example to install new software. Administrator rights comprise of the sum of these permissions. Consequently, when trying to gain control over a device, compromising an administrator account or gaining administrator rights is an important step. It is for this reason that Microsoft suggests to "use your local (non-Administrator) account to sign in and then use "Run as administrator" to accomplish tasks that require a higher level of rights than a standard user account. Don't use the Administrator account to sign in to your computer unless it's entirely necessary." [4]

Adhering to these standards, it is possible to install a game with administrator rights and then use a standard user to run it. Unfortunately, many online games incorporate some form of kernel-level anti-cheat solution, making administrator rights a necessity during execution as well [5]. Anti-cheat measures, as the name suggests, are applications meant to prevent cheating in a competitive gaming context. They run in parallel to the game they are meant to protect, either remotely or locally. Different solutions exist, with kernel-level anti-cheat being the most invasive (but also the most powerful) method available. It works by installing a driver directly in the Windows kernel, monitoring not only a game's process but every other running process as well. Kernel drivers have full access to a machine's memory, hardware, and CPU instructions [6]. By observing all processes on a computer, even those running outside the current user's context, kernel-level anti-cheat methods counteract cheating very effectively. Further, they usually load independently during machine startup, decoupled from a game's execution [7]. In many ways, these anti-cheat drivers already exhibit the behavior of rootkits [8].

Paired with private end-users' general tendency to ignore the principle of least privilege this leads to games often being executed in an elevated context for convenience' sake [9]. It follows that these users suffer a higher risk of malware infection and data integrity compromise: Attackers exploiting a video game's vulnerabilities already have administrator privileges in this scenario, giving them a vastly greater range of options to choose from in terms of post-exploitation behavior [10]. The anti-cheat driver `ACE-BASE.sys` is an example for how an attacker might make use of and profit from the privileges associated with the kernel: Tracked as CVE-2024-22830, the driver contains a vulnerability letting a local attacker escalate privileges [11]. Attacks utilizing a kernel-level anti-cheat driver's vulnerability like this are very potent since they combine vast privileges with a high level of stealth. As mentioned, this kind of driver is very similar to a rootkit: it contains features to remain undetected, is loaded on system startup, allows for information exfiltration and network manipulation, and deploys removal-countermeasures [8]. This means that an exploit targeting this driver gives an attacker

- maximum privileges,
- a stealthy, hard-to-remove foothold, and
- a big selection of targets since every game utilizing this driver might come under attack.

This one-to-many relation between anti-cheat and video games is even more pronounced with game engines: Some engines are used to create hundreds of different games. Consequently, an engine's vulnerability propagates to every game using that engine. For example, the *Source*-engine allows for (sometimes undocumented) command line parameters to be passed via its own browser protocol. This feature was actively exploited in the past to write arbitrary files on a victim's system or connect them to malicious servers [12]. Another vulnerability in Source's remote console protocol allowed spying on player's IP addresses as well as remote code execution [13]. It is important to note that these examples affected every game created with the Source engine.

Examples like the above motivate this thesis, in that they show how video games' unique structure not only provides attackers with exploitable targets but also with a means of achieving persistence.

Historically, video games have been plagued by a multitude of vulnerabilities owing to their multi-faceted nature: Memory overflow bugs and denial-of-service exploits, arbitrary file load and remote code execution [14, 12]. Further, side-channel attacks as the one mentioned above were discovered, utilizing hidden console functionality, both local and on a game's server [12, 13]. Recent examples of games exploited this way include *Grand Theft Auto 5* [15] and *Apex Legends* [16].

Video games are not isolated entities with clear boundaries within a system's context; instead, they exist as part of a whole ecosystem of interconnected technologies. Examples include launchers and the aforementioned anti-cheat solutions, mods and online communities. All of these have come under attack over the course of video games' history: Steam could be exploited for privilege escalation [17] and remote code execution [12]. Privilege escalation is also possible by exploiting the anti-cheat driver `ACE-BASE.sys` [18]. Mods were targeted both for introduced vulnerabilities (e.g. *Cultist Simulator* allowing arbitrary DLLs as part of its mod packages [19]) as well as supply chain attacks (e.g. *Slay the Spire* [20]). Finally, numerous instances of player data being leaked exist, affecting companies such as *Nintendo* [21] and *Sony* [22].

All of the examples given so far show that video games can serve as potent vectors of attack. Unfortunately, the video game industry also suffers from the fact that, due to games' complexity, testing and quality assurance inevitably fail to eliminate (sometimes critical) bugs until after a game's release [23]. Flaws in video games frequently serve as a prominent mechanism in the *Speedrunning* scene, showing how common their appearance is [24]. Some even become beloved icons of a game among fans, for example *Pokémon*'s *MissingNo.* [25].

This thesis focuses on video games as a vector of attack against private PCs. The chapter after this introduction is dedicated to giving background information on different components that make up the video game ecosystem – both internal and external – and exploring the academic state of the art; it covers vulnerability detection and cyber threat research as overarching concepts as well as the video game-related topics cheating and anti-cheat, copy protection and piracy, modding, and launchers and engines, with data

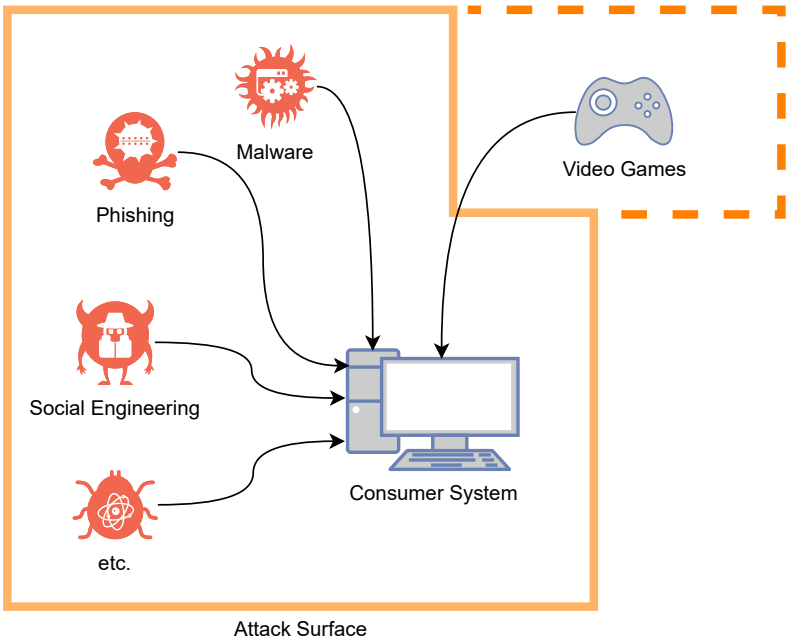

Figure 1.1: How do video games contribute to a system's attack surface?

leakage and privacy concluding the survey and serving as a conceptual bridge between video games and their developers.

The third chapter describes each of these aspects in greater detail, focusing especially on the analysis of their contribution to a system's attack surface. This is done by accumulating published examples of vulnerabilities and analyzing their impact on the underlying system. Scientific publications, as well as forum entries and blog posts, will be used as sources, thus creating an effective overview of video games' attack surface as a whole.

In the subsequent, fourth chapter a threat model is constructed and a proof of concept utilizing *DLL search order hijacking* is shown. The goal is to answer the following research questions.

1.1 Research Questions

- **RQ1.** *Which exploits exist that leverage vulnerabilities of video games to target the underlying system?* It is important to distinguish between attempts to hack a game for the sake of gaining an advantage within that game (*Cheating*) and the idea of using a game's vulnerability to gain illegitimate access to a whole system, in

the same way, e.g., remote code execution could be achieved by exploiting flaws in certain non-video game consumer applications. This thesis is focused on the latter.

- **RQ2.** By examining video games' attack surface, *is it possible to reproduce known vulnerabilities or finding new ones, thus showing a potential lack of focus on the kind of security flaw described above?*

The fifth and final chapter contains a conclusion and suggestions for future work, following up different possibilities emerging from the research done for and results shown in this thesis.

CHAPTER 2

Background and Related Work

Very few academic sources have analyzed video games for potential security risks. The relative absence of games in work environments – where security mechanisms are a higher priority – may help explain this lack of attention. However, in addition to the damage caused to individuals whose machines are vulnerable, compromising a person's private machine can easily lead to compromising their work account.

This chapter aims to give an overview of sources that do exist, covering important aspects both within and without the immediate video game ecosystem. Aside from related sources, it also introduces core concepts that either shape video games' attack surface or are used to analyze it; the first of the latter is software vulnerability detection.

2.1 Vulnerability Detection

Trying to find vulnerabilities in software is a time consuming task. Attackers are in a slightly better position, since they only have to find one exploitable vulnerability, whereas defenders would like to find – and mitigate – all possible exploits within a given piece of software. To this end efforts have been made over the last years to automate as much of this process as possible. The recent rise of the artificial intelligence paradigm has given a boost to these efforts.

Vulnerability analysis falls into one of two categories:

1. Static Vulnerability Discovery

2. Dynamic Vulnerability Discovery

2.1.1 Static Vulnerability Discovery

Static methods do not require program execution for analysis. Instead, they use an abstract domain over which the program is modeled. Two different approaches can be observed [26]: modeling the program as a control-flow graph or as pure data. These methods all suffer from two disadvantages, namely their lack of replayability and their limited semantic insight.

The first means that results have to be verified by hand since static analysis methods do not provide information on how a vulnerability was triggered. The second concerns static techniques' false positive rate: while they usually give good results regarding the absence of vulnerabilities, their over-approximation makes detection of existing flaws unreliable. This is due to static analyses' operating on simplified data domains for performance reasons [26].

2.1.2 Dynamic Vulnerability Discovery

Unlike static methods, dynamic analysis does require the analyzed code to be executed; this can be done using either a physical machine or an emulated environment. These methods fall into one of two categories: They are either executed concretely or symbolically. Results produced this way are highly replayable, although their semantic insight can vary [26].

Dynamic concrete execution means that a program is executed in a minimally-instrumented environment, computing and using data as it would under normal working conditions. This requires test cases to be provided, which makes this approach difficult for unknown code or input. Dynamic symbolic execution on the other hand means that a program is executed in an abstract domain consisting of symbolic variables; an application's state is tracked via the states and constraints of those variables [26]. This leads to a high degree of semantic insight since for any path reached during program execution the corresponding variable constraints give the input necessary to arrive at that specific state.

Both approaches incorporate the principle of fuzzing in their methods: By providing malformed input to an application, crashes are provoked dynamically. In case no crash occurs, the input is mutated randomly and tried again [26].

Vulnerability detection is an important tool for the broader, more general subject of cyber threat research. The term pertains to the analysis of every aspect of misuse of digital technology, including video games.

2.2 Cyber Threat Research

There is extensive work in the area of cyber threat research probing vulnerabilities of non-video game consumer applications and systems [27]. Many works exist that focus on specific categories of threats, like malware [28] or ransomware [29]. The methods employed include static and dynamic analyses, machine learning models as well as trained

neural networks [30]. Further, thorough benchmarks for various tools exist [31, 26] which can serve as a guide when choosing a specific technology.

Research focusing on the gaming industry is usually concerned with threats targeting developers and key infrastructure like servers. For example, Stephen Mohr [32] et al. cover *ZeniMax Media* as a case study, focusing on the software company's security concerns. This study was done in 2011. More recent publications by Kornélia Sára Szatmáry [33] or Amer Ibrahim [34] similarly analyze threats faced by video game developers, like DDoS attacks and data breaches.

However, little research has been conducted looking at how malicious actors can use video games specifically to penetrate private machines. Hatim Bennani takes a look at this possibility as part of their Master's studies, a literature review concerned with different aspects of cyber threats targeting the gaming ecosystem [35]. A dissertation, written by Jonathan K. Vanover, describes ways of hardening private systems against gaming-related threats [36].

Other sources often take the form of blog entries or forum posts. For example, *Imperva Threat Research* posted about cyber threats in the gaming industry, mostly focusing on attacks targeting developers and infrastructure [37]. The same topic was reported on a few years prior by *KELA* [38], emphasizing threat actors' use of darknet resources.

Sealingtech published a short article about cyber threats faced by gamers [39]. Similar articles exist by *Kaspersky* [40] and *Flashpoint* [21]. Another paper from 2016 looks at *PlayStation 4*'s security as well as different PC games [41].

After the more general topics of vulnerability detection and cyber threat research, the following sections' focus will lie on video game-specific aspects, the first of which is cheating.

2.3 Cheating

The technical aspects of cheats and cheating in video games are usually addressed by research focusing on their prevention (i.e. anti-cheat solutions). However, numerous works concerned with the psychological aspects of cheating exist:

Sung Je Lee et al. try to answer the question of what motives for cheating exist [42]. A study done in 2005 by Mia Consalvo is concerned with players' definition of cheating behavior and its rationalization [43]. A similar approach is adopted by Vivian Hsueh Hua Chen et al., narrowing the studied technology to online games [44].

Players' perception of cheating depending on context and goals within a game's ecosystem is described by Arianna Boldi [45] et al., the influence of social structures on that perception is the topic of a work by Jeremy Blackburn [46] et al.

Other works aim to draw parallels between cheating and various other behavior, for example academic performance [47] or cybercrime [48].

Finally, Santiago Pontiroli gives a technical overview of video game cheating terminology, tools, and trends [49]; one of the latest being the incorporation of AI into the cheat creation tool chain [50].

As mentioned, a topic that is inseparably intertwined with cheating is its prevention. Anti-cheat solutions are sophisticated pieces of technology with their own niche of research, problems, and vulnerabilities.

2.4 Anti-Cheat Solutions

A majority of the research analyzing video game security considers anti-cheat systems with an eye towards preserving the internal integrity of the games.

Recent publications, for example, compare (anti-)cheating technologies or develop new methods for detecting cheats. While Samuli Lehtonen [6] gives a high-level view of anti-cheat methods for video games, both Christoph Dorner [8] et al. and Anton Maario [51] et al. focus on kernel-level tools specifically. All three analyze the technology's invasiveness, albeit without further exploring the possible increase in attack surface. Ethical concerns like invasiveness and privacy issues of anti-cheat software have also been raised on various online platforms, for example by Dan Alder in an article written for *Levvvel* [7]. Further, the strong bias towards the Windows operating system introduced by many (mandatory) anti-cheat solutions has led to debate especially within the Linux community, giving rise to projects such as *Are We Anti-Cheat Yet?* [52]; it is described as "A comprehensive and crowd-sourced list of games using anti-cheats and their compatibility with GNU/Linux or Wine/Proton."

Cheat detection is often examined through the lens of machine learning and artificial intelligence; ranging from vision-based data fed into a neural network to detect cheating in first-person shooters [53] to training a machine learning model to evaluate user inputs [54] or metadata [55] as a detection mechanism. A different approach is suggested by José Pedro Pinto [56] et al., treating user input as a multivariate time series and using a convolutional neural network to find suspicious behavior.

In terms of "classical" cheat detection, efforts are being made to extract threat intelligence from existing cheat binaries [57]. Chenxin Sun [58] et al. even propose a method to counteract AI-aided cheating relying on game visuals. Another study by Blake Bryant et al. examines network protocols used by video games and their security [59].

Conceptually closely related to cheat prevention is copy protection; like other digital goods, video games often struggle with piracy – copy protection's illicit counterpart.

2.5 Copy Protection

Video game developers, like other creators, face the problem of how to protect their intellectual property. Different forms of copy protection and digital rights management

for games exist; research is done on their performance (e.g. by Mir Mohammad Azad [60] et al.), and their legal aspects (e.g. by Oleksandr Mihus [61]). An article by Andrew V. Moshirnia describes psychological aspects and how copy protection mechanisms could be improved without alienating consumers [62].

A different approach is proposed by Davide Quarta et al.: By utilizing trusted execution environments, code confidentiality is ensured, removing the need for additional digital rights management methods [63].

The topic of copy protection software – similar to anti-cheat solutions – is also being discussed heavily among the online community, with articles highlighting its turbulent history [64], as well as potential issues for consumers [65].

2.6 Piracy

Many sources cover digital piracy, often with a focus on analyzing technical or economical details. Examples include works by Charles W. L. Hill [66] and Arun Sundararajan [67]. A different category of studies more focused on video games is dedicated to the analysis of malware spread via game hacks (both real and fake) and pirated software. Svrana Kumar [68] et al. give a survey of malware infecting personal computers as a result of (pre)installed pirated applications. A similar attack vector is the so-called *Game Hack Scam* [69]. It works by deceiving victims into paying for unwanted subscriptions or downloading malware with the promise of providing cheats or hacks for a specific video game they are playing. Another facet are analyses of malware families and their distribution, as done for example by Markus Kammerstetter [70] et al. These studies have in common their focus on how malware is spread and the fact that the malware itself has nothing to do with video games.

Aside from technical analyses, the topic of piracy is also being studied in terms of legal and psychological aspects. Shivam Kaushik writes about the history and legality of digital piracy, also touching upon video games [71]. Sigi Goode et al.'s research asks what motivates people who create software cracks [72].

It is only a small step from piracy to modding, a video game-specific topic that originated in the hacking scene and resides in a legal gray area.

2.7 Modding

Many aspects of modding have spawned research projects, among them questions of economics (e.g. how the video game market profits from mods [73], or which economic trends among modders can be observed [74]) and legal aspects (as described by Erika Weisdorfer for example [75]).

Joanna Curtis et al. not only take a look at motivations behind modding and its legality, but also mention possible threats as a consequence of malicious intent when creating

a mod [76]. The topic of mods posing an exploitable threat to cloud gaming is being explored by Guannan Liu et al., who also provide proof of concept [77].

More or less detailed reports on current exploits can again be found within the online community: Examples for affected games include *Tales of Maj'Eyal* [78], *Cultist Simulator* [19], *Dota 2* [79], and *Slay The Spire* [20].

Moving closer to the core technologies of video games, the next sections cover launchers and engines, but also take into account the relationship between a game's developer and customer data.

2.8 Launchers and Engines

In a presentation published in 2013, Luigi Auriemma [80] et al. examine the potential vulnerabilities of game engines and their consequences for users. The authors also analyze *Electronic Arts*' *Origin* launcher in greater detail and show how it can be used as a vector of attack against a user's machine [81]. They published a presentation about vulnerabilities concerning the Steam launcher as well [12].

Both researchers and hobbyists post actively about current exploits and vulnerabilities in online blogs; examples include incidents involving Steam and *Valve* [17, 82, 83, 13], or games like *Dark Souls 3* [84], *Apex Legends* [16], and *Grand Theft Auto 5* [15].

2.9 Data Leakage and Privacy

Jacob Leon Kröger et al. published an article in 2023, describing different aspects of the video game industry that affect end users' privacy [85]. They mention the vast amounts of data collected by the gaming industry as a whole and compare the situation and practices to those of social media platforms. Ratiros Phaenthong et al. published a similar study concerning Android games [86]. Another study from 2021 examines privacy concerns of online gaming communities in contrast with gamers' desire to take part in the community and express themselves [87].

Concrete examples of data breaches within video games or video game companies include *Sony* [22, 88] and *Bethesda Softworks* [89].

All aspects mentioned so far play an important role when considering attacking and exploiting video games. The next chapter aims to combine them into a comprehensive analysis of the potential attack surface presented by video games when seen as the sum of their aforementioned parts.

CHAPTER 3

Video Game Attack Surface

Video games have a very diverse attack surface. This is due to their internal complexity, the technologies that are part of their supply chain, and the involved parties' (economical) interests.

Unlike most regular consumer software, video games are real-time systems. This means that their internal state is constantly being updated, independent of user input. These updates take the form of physics calculations (to correctly place all objects on the screen), keeping track of various statistics, and in general steering the game environment's behavior (i.e. taking care of all the elements that are not directly influenced by player input). In contrast, many consumer applications only update their state whenever a mouse action or keystroke occurs [90].

Updates of a game's state occur quite frequently, with 60 times per second being the norm [91]; many games offer unlimited updates as an option, depending on a user's hardware setup. As a consequence, small errors when handling data can quickly accumulate and lead to unexpected behavior. Paired with the vast amount of different systems that have to correctly work together, it is no surprise that video games are often times error-prone and bug-ridden despite extensive quality assurance measures [92].

Broadly speaking, a game can be described as three sub-systems working together: the application layer, communicating with the operating system and hardware; the logic layer, responsible for the internal state and its change over time; and the game view, presenting this state as graphics and sound [90].

Components that make up these sub-systems include kernel drivers, process and input handling, memory access, network communications, sound drivers, and the graphics pipeline [90]. In terms of attack surface, every single component adds to an application's potential vulnerabilities.

By just looking at their structure, video games are an entirely different category of software than regular consumer applications. While a game's vulnerabilities, when looked at isolated, are the same as any other piece of code's, the fact that they can occur in so many different places makes video games unique in that regard.

Another aspect to consider is online gaming: Having a networking protocol makes it possible for an attacker's payload to be delivered directly via the game, instead of, for example, in the form of an email attachment or as a download [93]. Moreover, the servers required to host the game are a potential target for malicious actors, offering the possibility to affect many clients at once with a single successful infection [21].

With the rise of the *Malware-as-a-Service* paradigm, a whole industry of professional malware coders was created: people whose day job consists of finding and exploiting weaknesses in – among other things – software [28]. Considering the gaming industry's revenue it is not surprising that a good portion of these actors' attention is focused on a tailor-made business model: *Cheats-as-a-Service* [49].

This has two implications: firstly, that there is no guarantee that all the vulnerabilities found in games by cheat manufacturers are used exclusively to create cheats – especially since not every security flaw that permits cheating can be leveraged to attack the system a game is running on, and vice versa. Secondly, the cheating ecosystem poses a significant supply chain to be exploited [68].

Because of the legal gray area surrounding video game cheats, the entry barrier for many players to use them is not very high. And while there are (mostly) no enforced consequences to cheating (in most of the world there is no legislation to that effect), most people are aware of the fact that by cheating they stop being "customers in good standing". As a result there exists a supply chain for games (the cheat industry) that can be attacked and exploited by hackers without repercussions since the cheats' creators have no legal basis to operate on, i.e. "cannot call the cops" – and neither can the players who use those cheats [70, 69].

It is worth noting that in the scenario described above, i.e. a situation where players' machines get compromised by a cheat program they installed, the cheat's creator, seller, and the party exploiting a vulnerability therein might be three different entities [70].

A similar attack vector is pirated software; only instead of downloading and installing semi-legal cheats, players download and install fully illegal cracked games. The principle of using this situation for supply chain attacks is the same as with cheat programs [70].

These two – cheats and cracks – are not the only modifications to games that exist outside of regular updates and patches. Another huge segment of video gaming is the modding scene [94].

Video game mods are player-created modifications of a game. These can range from simply tweaking a few assets, to creating custom levels, to basically writing a whole new game with different graphics, mechanics, and content. Some games even provide specific tools to facilitate creating and sharing modded content. Unfortunately, these

modifications can introduce vulnerabilities to an otherwise stable and well-protected game; given that their creators often times have limited programming experience, the chances for that happening are quite high. Furthermore, there is also the possibility of an entire mod being created by an attacker [79].

As mentioned, video games are very complex programs and, as a consequence, quite volatile. Players are used to frequent crashes and glitches. This also plays a role in their assessment of system sanity: While most users would be suspicious of a constantly crashing mail client, a game doing the same thing might just be an indication of some hardware issue or misconfiguration by the user. This goes doubly for "unofficial" modifications, therefore malware might be detected later in this context than on a machine with no games installed [76].

So far, video games' attack surface consists of the combined vulnerabilities of their software components as well as those of their third-party modifications (both legal and illegal). These – cheats, cracks, and mods – form a special part of a game's supply chain. A closely related category of software constitutes anti-cheat and anti-piracy measures.

It certainly is in the interest of actors creating cheats and cracks to also evade any countermeasures against their products. This means that the same efforts that are made to find games' vulnerabilities exist also to find security flaws in commercial anti-cheat and anti-piracy software. These are, unlike games, of a more general and therefore wide-spread nature: The same cheat prevention can be used for hundreds of games. Combined with usually high privileges, anti-cheat and anti-piracy software make for very desirable exploit targets [8].

In terms of their ecosystem, two kinds of application that shape video games' unique attack surface and that share the characteristic of one product affecting many games are also worth mentioning: The first one are game engines, the second one are launchers.

Many modern games are produced using game engines [95]. From a programmer's point of view they provide valuable tools to facilitate various aspects of production. Unfortunately in terms of vulnerability they constitute a single point of failure when taking into account all the games that are based on a single engine: If there is an exploitable flaw in that engine's framework, every game then shares that flaw. This issue is compounded by the fact that game engines are very much like games in terms of complexity, i.e. not the easiest to debug [96].

A similar argument can be made for the different game launchers that double as library management tools and shops. They, too, provide a shared infrastructure for many games that, if breached, affects all of them [12, 80].

Finally, many modern video games have an online component. This usually requires players to create an account, providing certain personal data (the extent of which varies between games). These data can also come under attack by malicious actors, and data leaked this way can have serious consequences, for example if login credentials are being reused [38].

Another aspect of (online) gaming is that players engage in social interactions with complete strangers. As with all similar platforms, this enables a variety of social engineering tactics most users do not expect and are not prepared for [21].

When compared to consumer applications, no single one characteristic of video games is exclusive to them or vastly different to those of the former. It is the combinations of all the factors listed above that make games – and their attack surface – unique.

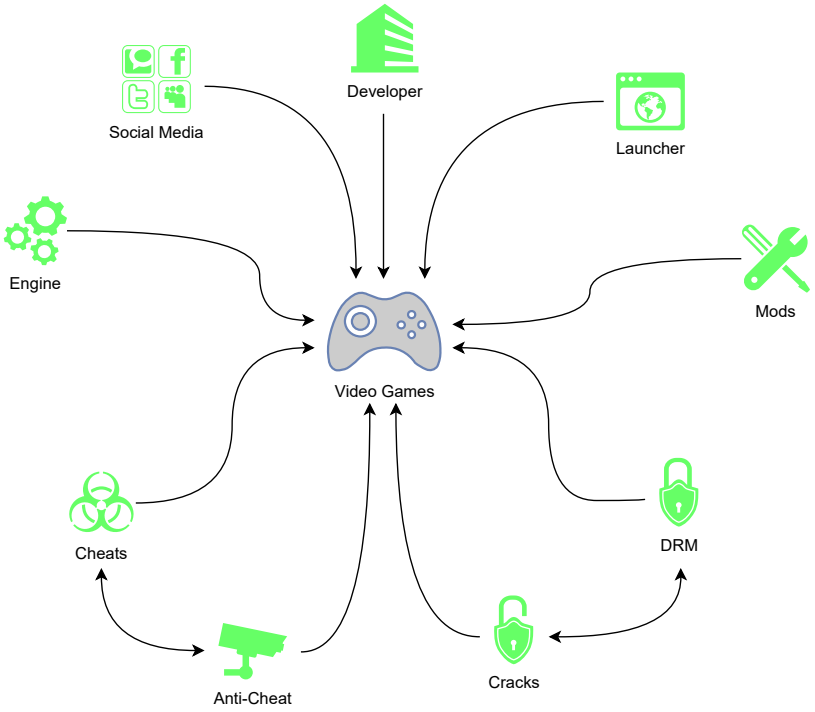

Figure 3.1: Different components of video games potentially contribute to their attack surface.

The following sections explore each of the components described above and shown in figure 3.1, giving an overview of the technologies and their contribution to video games' attack surface.

3.1 Cheating and Anti-Cheat

The phenomenon of cheating in video games adds an additional aspect to the analysis of their potential attack surface. When analyzing regular consumer software from a security standpoint, the important question to be asked is that of exploitability: Is it possible for an attacker to abuse the application for some unintended purpose? This could be due to bugs in the code or some functionality working as intended having an unintended side

effect. In creating program-specific hacks, an attacker usually has a different end goal, like remote code execution or escalation of privileges [93].

In contrast, video games often represent a malicious actor's main target: Bugs and design flaws are being exploited, hacks are being created for the sole purpose of gaining an unfair competitive advantage within the game's context [45]. While this is not an increase of attack surface (it makes no difference if an attacker is looking for ways to exploit a game because they want to cheat or achieve persistence on a user's machine), anti-cheat measures certainly are.

Over the years different technologies to prevent video game cheating have been developed. Depending on their purpose, some are more invasive than others; they range from pure server-side checks to kernel drivers on end-users' machines. Generally, the more invasive an anti-cheat method, the larger the additional attack surface it creates.

3.1.1 Server-Side Anti-Cheat Technologies

As the name suggests, server-side anti-cheat technologies are limited to the server that is hosting a session and therefore minimally invasive. In theory all the calculations necessary to run a game could be done server-side; the problem is that even with modern fiber-optics connections, games based on fast player reaction cannot achieve the same input-output-loop as if they were run locally. Depending on the game mechanics, a compromise in terms of calculation distribution has to be found [6].

Client Data Verification

Client data verification is something every game server does to some extent, if merely for synchronization purposes. The way it works is by calculating relevant results on the server; either in a redundant fashion (i.e. to compare with a result sent by the client) or exclusively [6]. In terms of cheat prevention this is a good way to make sure players cannot behave in an anomalous way. Examples include server-side calculation of positional data or keeping track of various in-game resources.

Unfortunately for some games even light-weight calculations would result in unacceptable lag. Also, exploits giving a player an unfair advantage in terms of knowledge cannot be prevented this way. An example for this is the classic wall hack scenario: To render enemy models, a game client has to know their positional data, even when a player's line of sight is obstructed. No amount of server-side calculations can remove the need to at some point transmit these data to the clients. It then becomes a question of whether the client is able to access these data in an unintended way [6].

Data Obfuscation and Encryption

To protect the data traffic from and to the game server from unauthorized access both obfuscation and encryption techniques are used [6]. A staple like data verification, they make cheating harder but by no means impossible. On one hand the methods used

cannot be too computationally expensive since otherwise lag becomes a problem again. On the other hand, since all of the game client's code resides with the player any attempt at encryption becomes security by obscurity in the long run [6].

Statistical Analysis

Not usually employed in real-time, statistical analysis still offers a solid anti-cheat solution. It is very adaptable to different genres of games, and benefits greatly from AI support.

The general principle is to capture key metrics of player behavior during a game session and then use statistical tools to compare them with a previously established ground truth. If the difference exceeds a certain threshold, a player can be classified as a cheater (and subsequently banned from the game) with an error rate specified in advance [6].

Different metrics can be used to measure a player's behavior. These range from input dynamics (e.g. mouse movement or keystroke patterns) [54] to player statistics (e.g. win percentage or shooting accuracy) [54]. They can then be analyzed "by hand" as part of a player peer review process or with common machine learning algorithms like decision trees or support vector machines [54]. Even more involved analysis techniques have been proposed, for example multivariate time series fed into a convolutional neural network [56].

Although statistical analysis is not (yet) being employed in real-time, the theoretical detection rate is very high (up to 98% in the case of multivariate time series analysis [56]). It is also a minimally invasive technique.

3.1.2 Client-Side Anti-Cheat Technologies

Client-side anti-cheat is run on the player's machine. This makes it quite invasive, since it requires more resources than just the game client would, no matter the concrete technique implemented [6].

On the other hand it is necessary for the prevention of some cheats to utilize client-side solutions. Depending on the desired degree of hardening, different methods exist, with scope ranging from the protected game to the whole client system [6].

File Hashing

File hashing is a very basic method to confirm on startup that a game's files have not been tampered with. By generating a hash value for every relevant game file (this could range from calculating the hash sum for just a single file to the whole locally saved folder), their integrity can be verified by comparing values with those of untampered with samples. It is especially necessary to compensate for any omitted server-side integrity checks for performance's sake. The downsides are that this method itself is not tamper-proof and that most modern cheats do not modify the game files anymore [6].

Code Encryption

One of the problems all applications run on a client face is that with enough effort and time they can be disassembled and their code reconstructed and inspected [6]. This is already problematic for any proprietary software since the functionality contained in its code is what gives it its value. However, it also plays a role for attackers looking to exploit a piece of software: The easier it is to get at its code, the faster a vulnerability can be found.

Code encryption is used to hinder these efforts by decrypting code protected this way only when it is being loaded into memory [6].

Memory Obfuscation

To run smoothly, games have to store information about their different components (e.g. player positions) in memory. These data in their totality make up the current state of the game. It is therefore possible to gain absolute knowledge of all the players' actions by looking at the memory. Since this is rarely part of a game's design, some sort of memory protection is desirable to prevent cheating.

Memory obfuscation techniques solve this problem: by both encrypting variables and randomizing their position in memory [6]. To refine this method, cheat binaries for a specific game and their memory access can be analyzed; by concentrating obfuscation on those parts of memory that get modified most often, these kinds of exploits can be counteracted efficiently [57].

Vision-Based Cheat Detection

The client-side methods described so far are of a more general kind; vision-based cheat detection is more specific. It functions not by verifying code or inspecting memory but by evaluating the frame buffer's final state, i.e. the visual output a player actually sees on their monitor [53]. Naturally, this only works against cheats that modify the cheater's perspective. Examples would be visual overlays displaying cheat data or altering the displayed geometry to enhance a player's field of view [53].

Vision-based cheat detection trains a neural network to evaluate a machine's frame buffer. By not reducing resolution, visual fidelity high enough to detect even small deviations from a "clean" frame is guaranteed. Moreover, this method has a high success rate of detecting cheats even when counter-measures are deployed by a cheater [53].

Interestingly it is not only defensive measures that are utilizing vision-based methods: Cheat developers also discovered the potential of using only frame data for their exploits to avoid detection. Visual cheats – in contrast to memory-access cheating – are very stealthy since they only access data that is legitimately available to a player. They do not provide information that is impossible to have without them but rather enhance a player's sensory capabilities artificially. They also benefit greatly from the advancement of AI technology [58].

It is not surprising that there exists a defense against these kinds of exploits in the form of visual perturbations introduced in the final rendering stage. These have to be pronounced enough to confuse visual cheats yet so subtle that they can not be perceived by players [58].

The same characteristics that make visual cheats hard to detect work in favor of vision-based cheat detection: It is not invasive, efficient (thanks to AI), and highly adaptable [58].

Signature Detection

Signature detection of cheats works basically the same way as general-purpose anti-virus solutions. The machine a game runs on is scanned (beforehand or at runtime) by the anti-cheat tool and suspicious signatures are either reported to a server or dealt with immediately, depending on the tool's invasiveness [6].

These signatures can be anything from process or file names, to hash values [6]. There are obvious privacy concerns associated with this kind of detection: If the whole machine is to be scanned, all of the personally identifiable information could potentially end up on the game company's servers. Further, the permissions needed for thorough scanning might pose a security risk if the anti-cheat tool contains any vulnerabilities. Finally a bias could be introduced as is often seen on Linux machines: Legitimate programs needed to run the game are detected as cheats and game execution is stopped by the anti-cheat [52].

There exist variants of signature detection that are less invasive. Instead of scanning the whole machine, modern cheat characteristics can be leveraged to concentrate scanning on certain processes: so-called *Injectors* [55].

Video game cheats come in many different shapes and flavors, making cataloging them a daunting task. Moreover, they might be hidden anywhere on a machine's file system, leading to the privacy concerns mentioned above. However, most (memory access-based) cheats need to be injected into the game process at runtime. Far fewer injectors exist than cheats and by reducing signature detection to current processes its invasiveness can be reduced [55].

Kernel Drivers

Kernel-level anti-cheat is both the most invasive and powerful method available. Operating with maximum privileges, kernel drivers have full access to a machine's memory, hardware, and CPU instructions [6]. This enables them to counteract cheating by observing all processes on a computer, even those running outside the current user's context.

In addition to these privileges, kernel-level anti-cheat usually loads during machine startup (no matter if the game client is being run or not) and is able to block other drivers deemed malicious from loading [7]. In fact, many parallels can be drawn between the behavior of rootkits and kernel-level anti-cheat [8]:

- **Evasion** - Rootkits contain features helping them remain undetected by intrusion detection systems. Common methods include hiding malicious processes and hooking tactics. Kernel-level anti-cheat employs evasion techniques to make it harder for cheat developers to circumvent its protection.
- **Virtualization** - Rootkits and kernel-level anti-cheat both employ virtualization to protect their code from reverse engineering.
- **Time of execution** - Rootkits frequently load with or before the infected machine's operating system. On one hand this serves the purpose of avoiding detection by the operating system's security mechanisms as well as monitoring tools deployed on the user level. On the other, it makes eavesdropping on other, subsequently loaded processes possible. Exactly this ability of observing other processes is the reason for kernel-level anti-cheat to similarly load as early as possible.
- **Remote access** - Rootkits usually give malicious actors remote access to their victims' machines. This is mostly done to incorporate them into botnets. Kernel-level anti-cheat requires remote access to keep its detection mechanisms up-to-date and allow on-demand loading of additional modules.
- **Information exfiltration** - Rootkits offer functionality to extract data from an infected system. These can then be used as an attacker sees fit; for reconnaissance or financial gain. Kernel-level anti-cheat usually collects information about the system it is running on to uniquely identify it. This can include hardware and software signatures as well as hard drive contents.
- **Network manipulation** - Rootkits are able to communicate over the network by attaching themselves to the network card's drivers. This makes it impossible for user-level monitoring to detect this activity. Kernel-level anti-cheat does it for a similar reason: to have a communication channel that can not be tampered with by a cheater.
- **Removability** - Rootkits have mechanisms in place that make them hard to remove, sometimes making hardware replacement necessary. This is done to achieve a maximum of persistence. Kernel-level anti-cheat operates under similar conditions, even though stealthiness is not strictly necessary; Players are usually aware that they cannot remove a game's anti-cheat without losing the ability to play the game. Still, kernel-level solutions often times hide quite well on users' machines once installed.

This creates a number of problems. First, the kernel driver might interfere with regular programs running on a machine. It could even affect the operating system: In 2019, the Windows 10 beta branch suffered a severe hold-up because no reliably stable versions could be published. The reason was an unspecified anti-cheat driver, causing regular crashes [97]. Second, it poses a significant threat to players' privacy. Third, it might be vulnerable to exploits itself, drastically decreasing a machine's security levels [7].

While all these concerns also hold true for signature detection-based methods, the difference is that kernel drivers are loaded on startup, vastly extending the time frame during which these issues might be relevant.

3.1.3 Attack Scenarios

In terms of exploitation potential, anti-cheat measures clearly add to the attack surface; especially if they are not integrated in the game client but run as an independent process instead. Further, no one method is able to combat cheating effectively on its own, calling for some kind of combined solution. This in turn increases the potential for vulnerabilities even more [11].

Another aspect is the fact that an anti-cheat tool's scope is very limited: It is concerned with securing the game, not necessarily the machine it is run on. This might go so far as ignoring or even introducing security flaws as long as gameplay is not disturbed.

An especially troublesome attack scenario are so-called *BYOVD (Bring Your Own Vulnerable Driver) Attacks* [28]. Kernel drivers by design are a very desirable target for attackers: They have the highest possible privileges within the Windows context, higher even than the administrator role. A malicious actor gaining these privileges would have complete control over the system. Since drivers under Windows are considered a trusted part of the operating system, they can only be loaded if they are digitally signed by Microsoft. This is meant to add an extra layer of security, preventing attackers from loading arbitrary (kernel) drivers [98].

To get around this restriction, it is unnecessary to forge a signature or try to get a malicious driver signed by Microsoft; all an attacker needs is a legitimately signed but vulnerable kernel driver. Prime candidates for such a technique are kernel-level anti-cheat measures [11].

As stated, security considerations for anti-cheat tools are often times limited in scope: It is in the developer's interest to secure them against reverse engineering attempts and similar efforts to circumvent them and enable cheating. However, the underlying system's security is not necessarily on that list of priorities. On the contrary: exposing more functions of the driver can mean a more powerful protection of a game. Further, heavy obfuscation is a common practice, making it harder for security researchers to properly analyze an anti-cheat driver's code. It is therefore not uncommon for malicious actors to install a vulnerable kernel-level anti-cheat driver as part of a post-exploitation BYOVD attack [11].

For example, the anti-cheat tool *Anti-Cheat Expert* uses a kernel driver named `ACE-BASE.sys`. There exists a vulnerability – documented as CVE-2024-22830 [18] – that lets a local attacker escalate privileges. This is possible because the driver offers users functionality to terminate arbitrary processes. Further, it can give any process access permissions to any resource. The driver's tamper-protection is obfuscation and encryption; however, the decryption keys are hardcoded, making it just a matter of deobfuscation

before the driver can be fully exploited by an attacker. A big reason for this design flaw seems to be the assumption that only the specific game under the anti-cheat's protection will ever interact with its functionality – when in practice any process might make use of exposed functions [11].

Games using this technology include multiple free-to-play titles, among them *Mecha BREAK Demo* and *Delta Force*, both with approximately 90.000 concurrent users [99].

As an interesting side note the various techniques to prevent cheating mentioned above are a good example for the cat-and-mouse game that is ongoing between attackers and defenders. First the reaction to reading or altering game data in the form of memory obfuscation and server-side checks leads to more sophisticated cheat programs that in turn necessitate process inspection. When processes become too stealthy, kernel-level and vision-based anti-cheat is introduced – attackers subsequently take a page out of the defenders' playbook and develop visual cheats [50].

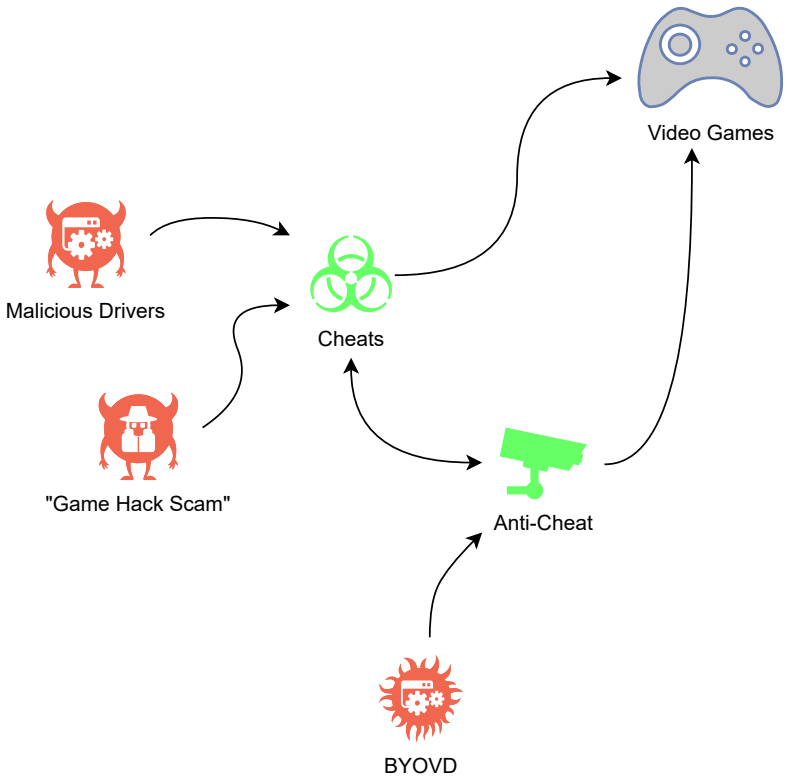

Figure 3.2: Attack vectors associated with cheats and anti-cheat measures.

3.2 Cracks and DRM

Pirated software constitutes a potent attack vector for (not only) private machines. Studies show that even newly bought computers often already come prepackaged with illegally obtained software versions (*Cracks*) containing some sort of malware [68]. The malicious programs range from adware to worms and viruses, trojans, and hacktools. This already bad situation is made worse by taking into account user-initiated installs of pirated software over the lifetime of a system [68].

3.2.1 Copy Protection

Many video games use a form of copy protection or *DRM (Digital Rights Management)* to prevent unauthorized copying and subsequent financial loss for their developer. Over the years, different forms of DRM have been developed, following are the most notable examples [64]:

- **Activation Keys** - Legally purchased games include a key that has to be entered upon installation, unlocking the game.
- **Online Verification** - A game's copy is associated with an online account, preventing more than one simultaneous user. Variants include one-time activation and the need to be online constantly while playing the game.
- **Dedicated DRM Software** - During installation of the game, additional software for verification of ownership is installed on the owner's machine.

Copy protection, while being beneficial for developers, has often sparked controversy among the gaming community depending on its invasiveness; always-online requirements for example make it difficult to play games in areas with bad internet connection or while traveling. Some DRM measures limit a game's number of installations, making it impossible to reinstall it after a certain number of hardware upgrades (since a "use" is associated with hardware ID). Others simply get abandoned by developers after a game's official end-of-support date, leading to older games no longer being legally playable. Finally, dedicated DRM software can lead to performance and compatibility issues (e.g. preventing games to run on certain operating systems) [65].

3.2.2 Attack Scenarios

A study done in 2012 shows that over 50% of available (working) cracks and key generators for proprietary software contain some form of malware, the most popular being trojans [70]. It is also shown that this is not necessarily intended by the original authors of a crack but rather a consequence of the way they distribute their work: By relying on third-party hosting providers to retain anonymity they relinquish a certain level of control over the content they create. This creates opportunities for other actors (not least of all the

hosting providers themselves) to alter files, upload copies of cracks containing malware or faking a working exploit and distributing malicious programs instead [70].

By counting each piece of proprietary software as a candidate to be replaced by an illegal version, adding games to this list of candidates hugely increases the potential for infection on any given machine. Especially since pirating video games to this day is seen by many consumers as a victimless and thus consequence-free crime [71, 72]. The same can be said for hacks and cheats for video games which have evolved to incorporate a very lucrative Cheats-as-a-Service model [49].

This has given rise to a very targeted form of scam: the game hack scam [69]. In it, victims are offered a cracked (and therefore "free") version of or unlimited resources (in-game currency, cosmetics, etc.) and other advantages for a specific game. In return, victims are asked to complete certain tasks; these range from completing online survey forms (to leak personal information) or subscribing to questionable services (e.g. "free" trials to video streaming platforms, requiring a credit card) to installing executable files on their machines. Most of the time, after having fulfilled all the tasks, victims get nothing in return. When they do, it is usually a modified executable containing malware, miners, or adware [69].

This shows that video games form an important part of a machine's attack surface, providing both direct and indirect opportunities to be leveraged by a malicious actor.

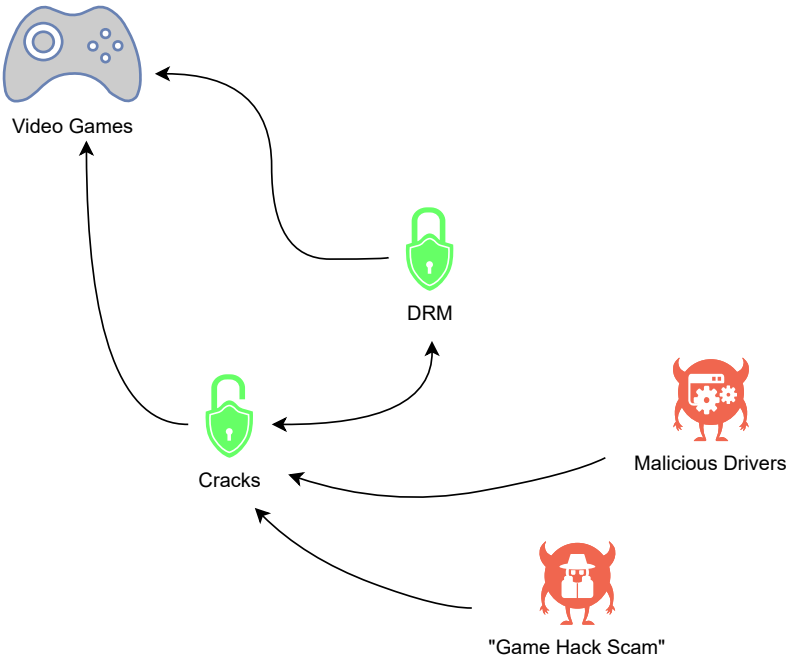

Figure 3.3: Attack vectors associated with cracks and DRM.

3.3 Modding

Video game modifications (*Mods*) are alterations and extensions of a game's content, usually created by fans. Different distribution channels exist, ranging from online forums to officially provided platforms like Steam's *Workshop* program. Depending on the specific game, mod creators have varying control over which part of it they can modify; although generally speaking, it is possible to alter every aspect of a game with enough time invested. It is common for mods to include DLLs and CMD files. For many players, mods are a convenient way to keep their favorite game interesting without spending any more money, since most of the time mods are provided free of charge [76].

Some games have a vast modding community; *Garry's Mod*, for example, has over 1.8 million registered mods on Steam [100]. Unfortunately, from a security standpoint mods are highly problematic. An exploit can have a vast impact and security standards are low. While some distribution platforms enforce certain rules, this is not the norm. Making matters worse, most mod developers are hobbyists, having obtained a skill set that is sufficient to achieve their goal of contributing to their favorite game's community, and nothing more. Understandably, security considerations take on a fringe role. Further, there exist no auditing standards – despite being free, many mods are closed source [76].

3.3.1 Classification of Mods

Modifications can be roughly classified as falling into one of two categories, depending on which aspects of a game they alter: either they change a game's underlying mechanics or their effects are cosmetic. The former requires more technical knowledge on the creators' part and represents a bigger security concern [101].

User Experience Customization

A huge part of any game is its presentation. This ranges from character art to background music, from interface design to control scheme. While changes to these aspects do not alter any gameplay, they still can have a substantial impact on how a player experiences the game [101]:

- **Character Mods** change the appearance of characters (both playable and non-playable). Players thus get the ability to insert their favorite likenesses into a game's narrative or to equip their character with clothes and accessories that more closely resemble their own tastes.
- **Asset Mods** change aspects like texture resolution or sound fidelity. While these modifications can also lead to a tonal shift, most often they are meant to improve a game's quality.
- **Interface Mods** change how game data are presented to the player. This also includes convenience functions like automatic aggregation and summation of core player statistics.

Gameplay Conversion

Mods in this category are characterized by their (partial) altering of gameplay mechanics. They are often created to improve upon existing aspects of a game, to add to the game world, or to create whole new games [101].

- **Rule Mods** change a game's core mechanics. Examples include simulation parameters in a racing game (like friction or drag coefficients) and resource costs or production times in strategy games.
- **Add-ons** provide additional content to a game. This ranges from custom weaponry to whole play areas, complete with additional story lines and cinematics in role-playing games like *The Elder Scrolls V: Skyrim*.
- **Total Conversions** change every aspect of an existing game to create something entirely new. One of the most popular total conversions is the game *Counter-Strike*, a total conversion of the game *Half-Life*.

3.3.2 Legal Aspects and Development Processes

Mod development is very dependent on how a game's developers approach the licensing of their game. Some games are very mod-friendly, making game editors publicly available or even part of their product. Often times mods are being tolerated by adding clauses to the license that state the extent to which a game can be legally modified; the line is usually drawn between cosmetic and gameplay alterations. Finally, some games' licenses forbid any mods outright [101].

This is reflected in the tools necessary to create modifications. Game editors are self-contained, requiring only to become familiar with their use. They usually also contain technical documentation, sometimes even a whole official online community dedicated to them [101].

Very mod-prohibitive games require considerable hacking skills to be modded. Creators need to be able to reverse-engineer the game code and develop tools to then be able to repack their mods in such a way that the game can still be executed. Taking into account modern copy-protection systems, their skill set has to include knowledge on how to circumvent these as well [76].

3.3.3 Attack Scenarios

Adding mod support to a game generally means exposing an interface, sometimes coupled with a specific scripting language. Both create additional points of attack: The interface – if not crafted and tested properly – might allow for unintended side effects; a scripting language might have security flaws of its own [78].

More substantial modifications to a game sometimes go so far as to make code re-writes necessary. These mods are usually made possible by extensive reverse engineering of a

game's code and therefore exist in a legal gray area. This way, entirely new executables are being created and players installing such a mod have to rely on the creator's good faith [19].

To make their code more transparent, some mod creators publish it on platforms like *GitHub*. On one hand this can help avoid security flaws; on the other it opens up new attack vectors in the form of repository poisoning or repository credential stealing.

For example, the game *Tales of Maj'Eyal*, released on Steam in 2010, offers mod support based on LUA. Version 1.4.8 had a vulnerability caused by missing filtering of mod-provided LUA commands: Any command, even file access and process spawning, would be run with the current user's permissions. This makes the game vulnerable to malicious actors achieving remote code execution if they can get their victim to download a specific mod [78].

A different vulnerability was found in the game *Cultist Simulator*: as part of its mod packets, it allows arbitrary DLLs to be included. When a mod is activated via the game's interface, the associated DLLs are loaded, regardless of their contents. This also showcases the fact that many developers' trust in Steam's security checks of community content might be misplaced [19]. When notified of this vulnerability, the developer reacted by requiring players to consent to the risk of downloading mods containing unknown DLLs. From their announcement: "[...] (Players are shown) a notice making you aware of the theoretical risk. This is basically the same as making the user tick a box to say 'I am OK with this' [...] Honestly I think the risk here isn't traffic-accident low, it's hit-by-a-meteorite low." [102]

While the previous examples are of a more theoretical nature (no proof of the mentioned vulnerabilities being actively exploited was found), the next covers a case where mods were uploaded to Steam's Workshop that contained code to exploit a vulnerability. The game affected was *Dota 2*. It used a vulnerable version of the *V8 JavaScript* engine to allow customization of its user interface. This version contained multiple actively exploited vulnerabilities, for many of which existed public proofs-of-concept. In an article from 2023, Jan Vojtěšek describes that they were able to find and report 3 mods in Steam's Workshop that would act as a backdoor to the machine they were installed on. Valve, after patching the vulnerability, stated that "under 200 players were affected" [79].

A final case exemplifying another interesting aspect of video games' attack surface is the game *Slay the Spire*. One of its most popular mods , *Downfall*, came under attack in December of 2023. Attackers were able to breach the developer's Steam account, uploading a malicious version of their mod. When loaded, it would install a data scraper, looking for user logins and passwords. While the payload's execution was not detected by most antivirus solutions, the subsequent connection attempt to a command and control server would usually be blocked – provided an antivirus program was running on a victim's machine [20].

This is an interesting example of supply chain poisoning in the context of video games. In this case, the attackers did not use a very subtle approach; it was only a matter of

time before the developer noticed their breached account and could react accordingly. But a dedicated attacker has other possibilities as well: Jan Vojtěšek describes the idea of just offering to take over development of an abandoned popular mod (of which there exist plenty) instead [79]. This approach comes with some overhead in form of social engineering but might be worth the effort if a big target population can be reached.

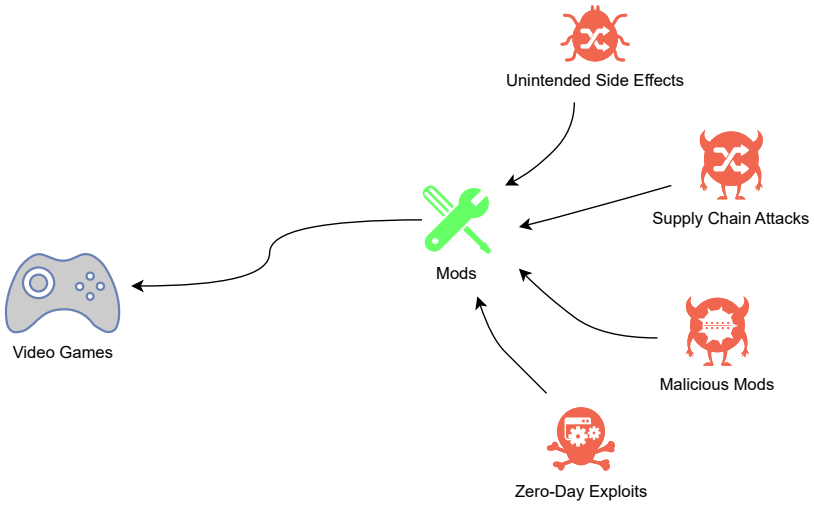

Figure 3.4: Attack vectors associated with modding.

3.4 Launchers

A video game launcher is "a software application that allows gamers to access and manage their game library. It serves as a storefront where users can browse and purchase games, as well as a platform to launch and play games." [103]

Besides offering convenience features to players (e.g. game time tracking or a friends list), launchers also serve as a form of copy protection, as well as a distribution platform for developers. While some launchers offer a variety of games from different developers and publishers, others are used exclusively to distribute a single company's products. They all have in common that they require a user account [104].

From a security perspective, launchers represent a risk in that they act as a single point of failure, which is especially problematic if they offer a big catalog of games [12].

3.4.1 Popular Examples

Steam

Steam was released in 2003 and has the biggest user base to date. It offers approximately 30.000 games, community features, and a dedicated developer tool kit [105].

GOG Galaxy

GOG Galaxy is associated with *GOG.com*, a site dedicated to preserving old video games and providing DRM-free versions. While not as big as Steam, GOG Galaxy has similar features with a focus on cross-platform compatibility [106].

Epic Games Store

The *Epic Games Store* was launched in 2018, following the success of the game *Fortnite*. In addition to video games, the launcher also offers access to different versions of the *Unreal* engine [107].

itch.io

The site *itch.io* specializes in supporting independent developers. Unlike the competition, it offers open revenue sharing and video game related products like comics and music [108].

Ubisoft Connect

Ubisoft Connect is a launcher developed by *Ubisoft* as an exclusive means to distribute their products. It is used both as a store front as well as copy protection [109].

3.4.2 Attack Scenarios

The Steam launcher makes use of a custom URL handler to integrate web-based functionality – so-called *Steam Links*. The URL format is `steam://`, followed by the desired command, e.g. `steam://install/220` opens Steam and installs the game *Half-Life 2* [110]. Possible commands include

- `install` and `uninstall` (install and uninstall games),
- `connect` (connect to a specified IP address),
- `launch` and `run` (run an application),
- `openurl` (open a URL in the default web browser),
- `store` (open the Steam store page for an application).

Valve's official documentation says that each of these commands can either be used with a command box (e.g. Windows' *Run*) or typed directly into any browser. They all first open the Steam launcher if it is not already running. The documentation also warns that some existing commands might be missing from its list [110].

An article by Luigi Auriemma and Donato Ferrante from 2012 [12] explores ways Steam's URL handler might be exploited. A simple attack scenario might look like this:

1. The victim is presented with a harmless-looking link hiding Steam browser protocol commands.

2. Clicking the link executes vulnerable local Steam functionality with attacker-controlled parameters.

Depending on the specific browser, the first stage might be more or less easy to achieve. In 2012, the following behavior for different products was observed:

Browser	**Behavior**
Chrome	Warning with a detailed description of the URL and the program to call.
Safari	Direct execution without warnings.
Internet Explorer	Warning including the URL, in case of IE9 a possible additional warning ("protected mode") without any detail.
Firefox	No URL visualized, only request for confirmation (no warnings).
Opera	Warning with only 40 characters of the URL visualized.

Table 3.1: Browser behavior for Steam browser protocol [12].

Additionally, many other applications are capable of parsing external protocol handlers. They usually open the default system browser but this is not always the case. Examples include document readers and code editors [12].

At this point, different assumptions regarding the victim can be made. Ideally, Steam is opened. If this is not the case, it is going to be started, no matter if the malicious link was processed silently or not. Therefore, the ideal scenario is one where Steam is running and the victim's browser executes Steam links without warnings. But even if neither is the case, it is still reasonable to assume that a user might be persuaded to follow a link that then opens Steam. A prime example for this would be an invite to a multiplayer lobby [12].

Once at this stage, there are different ways to exploit the browser protocol's options. One revolves around the usage of `retailinstall` – this command is undocumented and allows installing or restoring games from a local directory [12]. It has a parameter to specify the path which contains the installation/backup files. While this is typically meant to be a location on the user's machine, it also accepts network folders on a remote host. When `retailinstall` is called, Steam loads two files: `splash.tga` and `sku.sis`, with the TGA (image) file getting displayed immediately. Unfortunately, the function processing image files is vulnerable to integer overflow if the file is malformed. An appropriately crafted payload can lead to a heap-based buffer overflow, resulting in the execution of malicious code as part of the Steam process [12].

An alternative exploitation approach would be to run an application in a way that lets an attacker control its behavior. The commands `steam://run/` and `rungameid/`

both allow arguments to be passed which are used as command line parameters for the launched application. At the time of writing this no longer works for *Source*-engine games: Both commands mentioned can be used to start a game but additional arguments are ignored. Games based on a different engine still take command line parameters but Steam shows a warning that requires user interaction (see Figure 3.5). Still, since most games take some sort of launch parameters, the principle makes for a potent attack vector; Steam's catalog included roughly 100000 games at the end of 2024 [111].

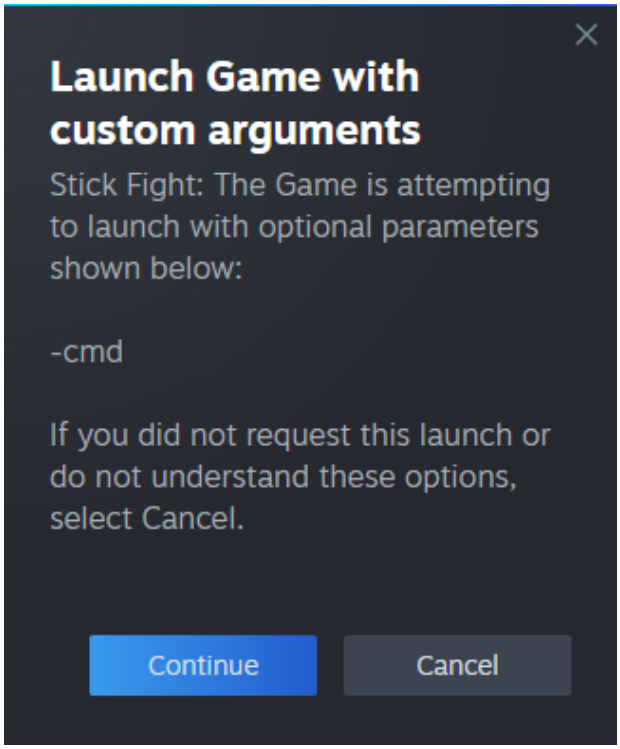

Figure 3.5: Steam warning users of command line parameters.

Another launcher that had vulnerabilities disclosed over its life time is Electronic Arts' Origin platform. Similar to Steam, Origin has its own browser protocol. What sets it apart is the possibility to chain multiple game IDs together when attempting to launch games. A valid Origin URL could look like this: `origin://LaunchGame/<id1>,<id2>,...`. Origin would attempt to launch every game in the list, giving attackers a potent tool for no-look exploitation; they could just chain every potentially vulnerable game in one URL [81].

What makes matters especially interesting is that games published by Electronic Arts always need the Origin launcher to run in the background, even when they are sold via a different platform like Steam. (All of these games' store pages contain the following information: "Incorporates 3rd-party DRM: EA online activation and EA app software installation and background use required." This example was taken from the game *EA*

SPORTS FC 25 [112].) As another example, the game *Mass Effect Legendary Edition*, released in 2021, when bought on Steam requires Origin to run and is built using *Unreal Engine 3*. Aside from any potential vulnerabilities particular to this game, it might be targeted by attackers exploiting security flaws in

- Steam,
- Origin,
- Unreal Engine 3,

showing impressively how a game's composite nature impacts the width of its attack surface.

So far, the potential exploits utilized launchers' (and games') functionality intended by their creators to achieve unintended side effects. A different category is the following exploit, taking advantage of Steam's installation process to escalate privileges locally. Also, unlike previous examples, the only application targeted in this case is the launcher itself.

The issue was reported by Daniel Gebert on July 02, 2020 as CVE-2020-15530 [17] and according to *OpenCVE* is still open at the time of writing, with the last status update received in November 21, 2024 [113]. Gebert's description of the vulnerability reads: "A local attacker can use the steam client updater/installer to execute malicious executables in an elevated context via the installation process. This also results in persistence via a hijacked service executable which will run as NT AUTHORITY\SYSTEM." [17]

The way Steam is set up, normal users are allowed to modify its installation folder, including deleting parts or all of it. The consequence is always a fresh install, since there exists no repair function. Steam's installer can only be executed in an elevated context, i.e. has to be given the rights to make changes to the device. During installation, two sub-folders and two files are created; their path is configured to allow user modification (as described earlier). One of the sub-folders contains an executable `SteamService.exe`. A copy of this file is also placed under `\Program Files (x86)\Common Files\Steam` – a folder controlled not by the user but by SYSTEM [17].

`SteamService.exe` is used by the *Steam Client Service*, a process that only runs when Steam is started. While it is executed as NT AUTHORITY\SYSTEM, it can be controlled, i.e. started or stopped, by normal users. This is problematic, since this discrepancy in access rights is what makes an exploit possible. If an attacker with user rights were able to replace `SteamService.exe` with a malicious version, they would gain elevated privileges on the victim's machine [17].

One way to achieve this is to take advantage of how Steam's installation works; the (simplified) steps are [17]:

1. The sub-folders and files mentioned above are created in a destination of the user's choosing.

2. `SteamService.exe` is launched with the `/Install` flag within the setup process's elevated context.

3. `SteamService.exe` is copied to `\Program Files (x86)\Common Files\Steam`.

Assuming an attacker has compromised a user's account, they can first perform a fresh install of Steam (enforced by the deletion of an existing folder, if necessary). Making use of opportunistic locks [1], the installation process can then be interrupted between steps 1 and 2. During this interruption, the freshly created `SteamService.exe` is replaced with a malicious executable of the same name in the installation folder. This is possible because this destination is under the user's – and therefore, attacker's – control. Upon continuing, the malicious file is launched with elevated privileges [17].

Normally, it would now be copied to `\Program Files (x86)\Common Files \Steam`, where it will always be called whenever Steam runs. Since this is part of the original `SteamService.exe`'s installation process, an attacker does not have to include this in their malicious version; however it might raise suspicion if the service is not working as expected. Also it would be a passed-up opportunity to achieve persistence since the Steam Client Service is called frequently, especially if Steam's default setup of being executed on system start is not changed during installation [17].

Launchers also represent an attractive target for attackers because of how customers perceive their function within the gaming ecosystem: It is generally assumed that games being sold have been developed by reputable companies and that a thorough process of quality assurance (including security-related issues) is in place. If an attacker were able to abuse this reputation, by either hijacking an existing game's code base or even by publishing their own product to infect buyers' machines, every customer using the launcher would become a potential target [82].

In 2023 several game developers' accounts on Steam were compromised and their games got updated with malware. While only a few users were affected (less than 100, according to Valve), the underlying issue is severe; Steam, as most current launchers, requires games to be up-to-date to be played, making it impossible for users to revert to the last uninfected version on their own. As a consequence and to increase developers' security, multi-factor authentication was added as a requirement to partnered accounts [82].

Recently, in March of 2025, the game *Sniper: Phantom's Resolution* was taken down from Steam's store page because it contained information stealing malware. It is important to note that in this case no developer got hacked but instead an attacker registered a

[1] "An opportunistic lock (also called an *oplock*) is a lock placed by a client on a file residing on a server. Client applications directly request opportunistic locks only when the lock is intended for a file on the local server." [114]

legitimate account and published their malware directly via Steam. Further, this abuse of the platform was initially not detected by Valve but first disclosed by the gaming community [83]. This is the second case of a legitimately published game on Steam containing malware in 2025; in February, *PirateFi* was taken down for similar reasons [83].

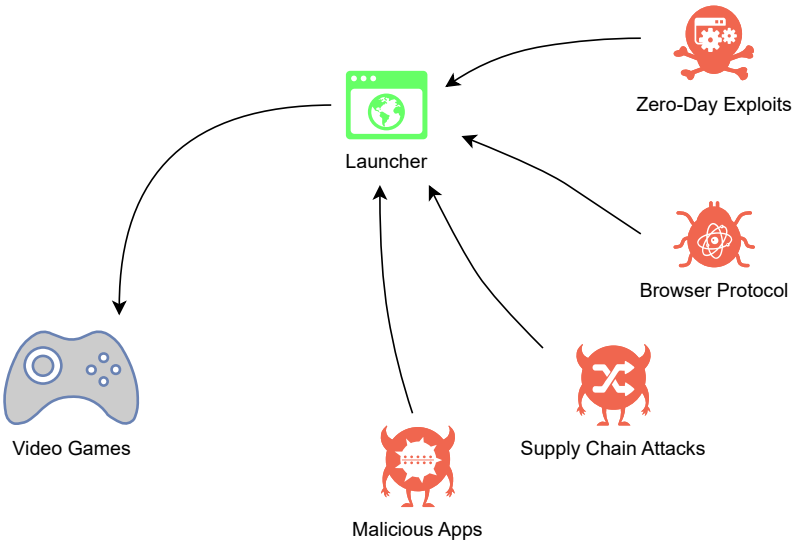

Figure 3.6: Attack vectors associated with launchers.

3.5 Game Engines

Game Engine is the collective term for all subsystems that, working together, make up a video game. This includes the application and logic layers as well as the game view. By definition, every game has an engine; in the simplest case the two are identical. In many cases it makes sense for a developer to reuse an engine's components for different projects, leading to multiple games being based on the same engine. Consequently, there exist game engines that are created not to implement one specific game but as general-purpose tools; they are meant to be utilized in the creation of as many different games as possible [90].

Game engines' many interdependent parts and high degree of code complexity pose a challenge to quality assurance, especially in terms of security auditing. The fact that an engine's vulnerability affects every game based on it further increases a potential attack surface [80].

3.5.1 Engine Components

Application Layer

The application layer is responsible for communications between a game and the system it is run on; it manages three subcategories [90]:

- **Devices** - Devices include input and output devices, the file system, and random access memory.
- **Operating System** - The operating system provides a game with thread and network access, as well as dynamic-link libraries.
- **Game Lifetime** - Game lifetime management is concerned with initialization, the main game loop, and shutdown. Video games, being real-time systems, require constant state updates even when no user interaction is being provided; this is called the *Main Game Loop*.

Logic Layer

The logic layer defines a game's mechanics and rules. It also tracks the current game state and changes thereof. Its most important components are [90]:

- **Data Structures** - Different data structures are used to represent different aspects of a game. Their interaction defines the game state.
- **Physics** - In its most abstract form, a game's physics simulation is responsible for state changes that depend on the passage of time. Therefore, almost all games need this component.
- **Events** - Interaction between a game's mechanics (e.g. damage model and aerodynamics simulation in a racing game) is usually event-based. In general, one aspect of the game state's changing triggers events that propagate this change to the whole state.
- **Command Interpreter** - The command interpreter changes the game state according to a received input. Inputs are sent from different game views and the interpreter translates each one into a corresponding update to the state.

Game View

"A game view is a collection of systems that communicates with the game logic to present the game to a particular kind of observer. [90]" This observer might be a human player, requiring video and audio output; it might also be an AI agent or a server, in which case different forms of output are required [90].

3.5.2 Popular Examples

Unity

Unity is a game engine with emphasis on versatility and cross-platform development. Its scripting language is C#. The latest engine version is Unity 6, popular games include *Among Us*, *Pokémon Go*, and *Ori and the Blind Forest* [115, 116].

Unreal Engine

First launched in 1998, Unreal was developed for the game of the same name [117]. It provides tools for game development, as well as animation and visualization. Using C++ as its scripting language since version 4, the current version is 5.5 [118]. Popular games include *Fortnite*, *Rocket League*, and the *Borderlands* series [119].

Source

The Source engine was created by Valve in 2004, its latest iteration is *Source 2*, released in 2015. It is heavily used in the modding community despite its age and several different branches exist. The scripting language is C/C++, the most popular games made with Source are *Half-Life* and its mod *Counter-Strike* [120].

CRYENGINE

CRYENGINE was released in 2002 as a tool to create the game *Far Cry*. It emphasizes physics and lighting simulations, the current version number is 5. Popular games include Far Cry, *Hunt: Showdown*, and the *Crysis* series [121, 122].

3.5.3 Attack Scenarios

In terms of adding to the attack surface, game engines are similar to launchers in that a vulnerability affecting an engine typically affects every game associated with it. In an article published in 2013, Luigi Auriemma and Donato Ferrante compile a list of vulnerabilities in contemporary engines [14]. They later added to that list as part of a conference talk they gave [80]. A summary of their findings can be seen in table 3.2.

Furthermore, Unreal Engine 3 had a vulnerability that allowed loading of arbitrary files from any Windows remote WebDAV or SMB share via command line parameters [12]. This in theory affected many popular games in 2012, among them *Batman: Arkham City*, *Dishonored*, *Borderlands 1 & 2*, and *Mass Effect 1-3* [123].

Despite their age, these vulnerabilities serve as a good example of how the attack surface of video games is expanded significantly by their components. And while many of the above issues have been patched in the meantime, security flaws in engines are by no means a thing of the past. Valve's Source engine, basis for many best-selling games on Steam, is an example of that.

Engine	Vulnerability Type	Affected Component
Unreal Engine 3	Array Overflow	Unreal Engine Control channel, processing custom commands.
	Invalid Read Access	Customized engine version with remote administration support.
	Adjacent Memory Overwrite	Customized engine version with remote administration support.
	Stack-based Overflow	Customized engine version with remote administration support.
	DoS via Out-of-memory Error	Customized engine version with custom opcodes.
idTech 4	Remote Code Execution	Customized engine version mishandling data read.
	DoS via Endless Loop	Customized engine version mishandling data read.
	DoS via Stack-based Overflow	Network protocol
CryEngine 3	Heap Overflow via Fragmented Packets	Network protocol
	Memory Corruption via Fragmented Packets	Network protocol

Table 3.2: Game engine vulnerabilities.

As mentioned earlier, Source accepts a plethora of command line parameters [124]. Among them are specific commands altering the so-called *console variable* of a game on startup. This is a way of utilizing the development console all Source-based games share. Of special interest in this context are the following commands [12]:

- `con_logfile` (sets log filename),
- `echo` (displays user-defined text in the console),
- `con_log` and `condump` (writes text displayed in the console to the log file).

With these, it is possible to first write arbitrary content to the console (using `echo`) and then saving this content to an arbitrary local file (using `con_logfile` and `condump`) [12].

A concrete attack scenario could look like this [12]:

1. The victim is sent a game invite link by an attacker, using hiding techniques to omit the added console variables if desired. This link would be of the form `steam://run/<id>//+echo <payload> +con_logfile <filename> +condump`.

2. The game specified by `<id>` starts normally.
3. During startup, `+echo <payload>` writes the file contents an attacker wants on their victim's machine to the console, while `+con_logfile <filename>` changes the logging destination to the desired location for the malicious file.
4. `+condump` then writes the payload, creating an arbitrary file on the victim's machine.

An example would be a BAT file in the user's startup folder that executes whenever the user logs into their machine [12].

A similar issue – reported as CVE-2021-30481 in 2019 – persisted until 2021, when Valve finally patched the Source engine in response [13]. The exploit allowed for remote code execution and made use of the *Source RCON Protocol*. From the official documentation: "The Source RCON Protocol is a TCP/IP-based communication protocol used by Source Dedicated Server, which allows console commands to be issued to the server via a "remote console", or RCON. The most common use of RCON is to allow server owners to control their game servers without direct access to the machine the server is running on." [125]

By adding RCON-specific parameters to a Steam Link, attackers were able to get players' games to connect to any server of their choosing. This already makes it possible for an attacker to find out players' IP addresses. The remote code execution exploit made use of how clients would process `SERVERDATA_SCREENSHOT_RESPONSE` RCON packets from the server [13].

Clients treat `SERVERDATA_SCREENSHOT_RESPONSE` packet data as a ZIP archive containing a file named `screenshot.jpg`. Once received, the archive is unpacked to disk. Neither location nor file name can be altered but an attacker is able to send these packets without the victim requesting them directly, just by including the appropriate parameters in a Steam Link [13].

Unfortunately, the library used for unpacking had an integer underflow vulnerability: When processing a ZIP file, it would look for the screenshot's local file header within the central directory file header as per ZIP format specifications. This information would then be used to calculate the offset at which the screenshot was located amidst the surrounding data. By specifically altering the archive's central directory field containing this information, an attacker was able to underflow the offset calculation, leading to memory corruption; worse, since the entirety of data within the archive would reside in memory as a continuous block, addresses containing attacker-controlled code could reliably be pointed to [13].

The whole exploit could be facilitated even further by altering a player's key-bindings via command line parameters which is also possible with Steam Links. By adding the command to request screenshot packets from the attacker's server to a frequently used key, the exploit's success was virtually guaranteed, even if it did not work on the first attempt [13].

More recent examples of engines' vulnerabilities being exploited include remote code execution in the games *Dark Souls 3* [84] and *Apex Legends* [16], as well as player accounts being corrupted by malicious actors in the game *Grand Theft Auto 5*'s online mode [15].

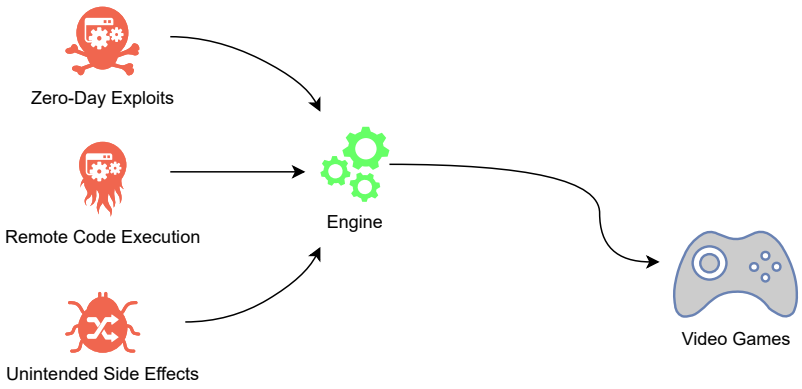

Figure 3.7: Attack vectors associated with engines.

3.6 Data Leakage and Social Engineering

Most modern games are in some way associated with an online account. Distribution platforms require a login, as do launchers; to play online, players need an account – no matter if the game is a standalone program or part of a streaming service. The same goes for support requests. Furthermore, there is social media: Engaging with a developer or game studio, or just with the community, means that individuals' accounts on those platforms become part of the data involved in gaming [85].

In the past, accounts of all kinds have always been a sought-after target for attackers. Almost by definition, a successful compromise includes gaining control of a victim's credentials (if only to achieve persistence on a specific machine). Account data therefore are very valuable; in addition to their direct use to an attacker, they can be sold in underground marketplaces as well. The customer base is multifaceted [21]:

1. Gamers are often interested in buying accounts. The reasons vary from just wanting access to a game for a lower price to owning desired digital goods associated with an account (e.g. cosmetics, statistics, or achievements).

2. Cheat developers also have a demand for working game accounts. A risk of testing and deploying cheats is getting the initial test account banned. Therefore, a cheap supply of such accounts is necessary.

3. Malicious actors use leaked login credentials for various credential stuffing attacks.

From a victim's point of view it is not only a question of becoming vulnerable or losing the account if a malicious actor gets hold of login credentials. Data privacy concerns also play an important role [85].

3.6.1 Ransomware and Data Theft

So-called *Ransomware* is malware that encrypts a victim's files, making them unreadable and thus unavailable; the idea being that only the malicious actor responsible knows the password to decrypt the data. Ransomware usually comes with an accompanying ransom note, asking for money in exchange for the password to make the data available again [29].

Recently, cybercriminals combine ransomware attacks with data exfiltration for double extortion: The victim has to pay money for the encryption key and then is being threatened with the public release of their sensitive data, unless they pay again. Finally, these data are oftentimes sold on underground markets regardless of the victim paying ransom or not [126].

The gaming industry combines large volumes of sensitive data (including user data but also corporate secrets) with very high revenues; making it a prime target for cybercriminals. The potential to compromise additional targets by acquiring user data compounds the issue [37].

3.6.2 Attack Scenarios

Being focused by malicious actors, supply and demand for initial network access for gaming companies is strong; Attack scenarios include industrial espionage and ransomware; customer data (e.g. game logins) is a bonus [38]. Gaming companies that have leaked data through being attacked in the past include *Nintendo*, *Riot Games*, *Rockstar Games*, *Bandai Namco*, *Bethesda Softworks*, and *Sony* [21, 127, 22].

Sony suffered data breaches on multiple occasions. And while most recent incidents concern research and employee data, in 2011 the theft of private data of 77 million *PlayStation Network* users and 25 million *Sony Online Entertainment* users was reported [22].

These data also included *PlayStation 3* device IDs, unique hardware identifiers for these consoles. As a consequence, some customers got banned from Sony's online service. The reason for these bans is presumed to be hackers engaging in illegal activities, utilizing spoofed device IDs they found within the breach. This shows how malicious actors can not only leverage stolen data but also harm legitimate customers; especially since the PlayStation 3 is still being bought and sold frequently on second-hand markets [88].

It is not always malicious actors responsible for a data breach either; Bethesda Softworks accidentally leaked personal data of *Fallout 76* players in 2018. Answers to customer-support requests were forwarded to the wrong recipients. While the root cause was never publicly disclosed, it resulted in users receiving arbitrary support tickets. These contain email and home addresses, severely compromising user privacy [89].

Aside from stealing account data directly from a company, phishing strategies are being employed by cybercriminals to gain possession of user accounts as well. Common lures include fake promotions for in-game gifts, free items or skins, as well as access to beta tests. Many of these campaigns are also tried against players' social media accounts [40].

This is not the full extent of social engineering tactics surrounding video games though; another common strategy are scams offering aforementioned digital goods for low prices. Players who fall victim to these are asked for their credit card information and after money is transferred, they never receive any goods. Finally, users often get tricked into downloading malware via the offer of free games. Most of the time these downloads do not even contain a cracked version of the game either [40].

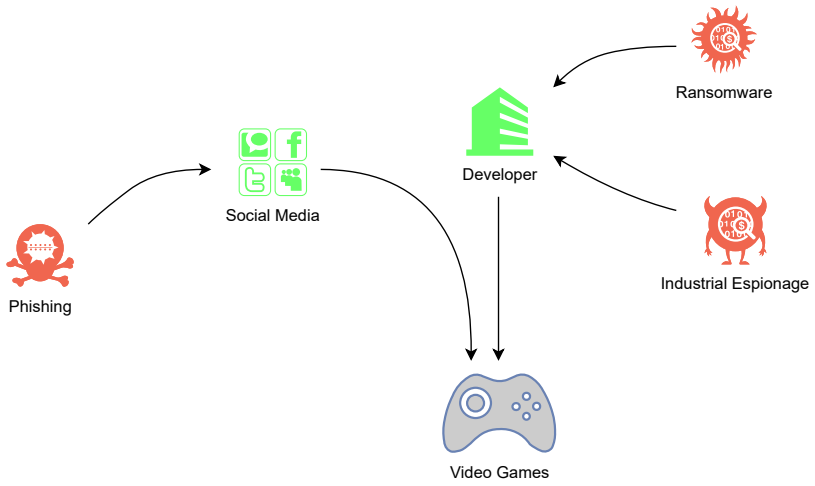

Figure 3.8: Attack vectors associated with data leakage.

3.7 Summary

This chapter shows that threats to video games and their associated infrastructure are manifold and that having gaming software installed on a system definitely increases its attack surface. The severity of vulnerabilities differs; some might be easily avoided by a conscientious user (for example the various flavors of scams and malware associated with cracks and cheats) while others – like different kinds of supply chain attacks and data leaks – are impossible to exert control over. Here the responsibility lies with the companies selling or distributing the software.

Figure 3.9 aims to visualize these different threats; whenever multiple variations of an attack vector exist, another corresponding symbol is added.

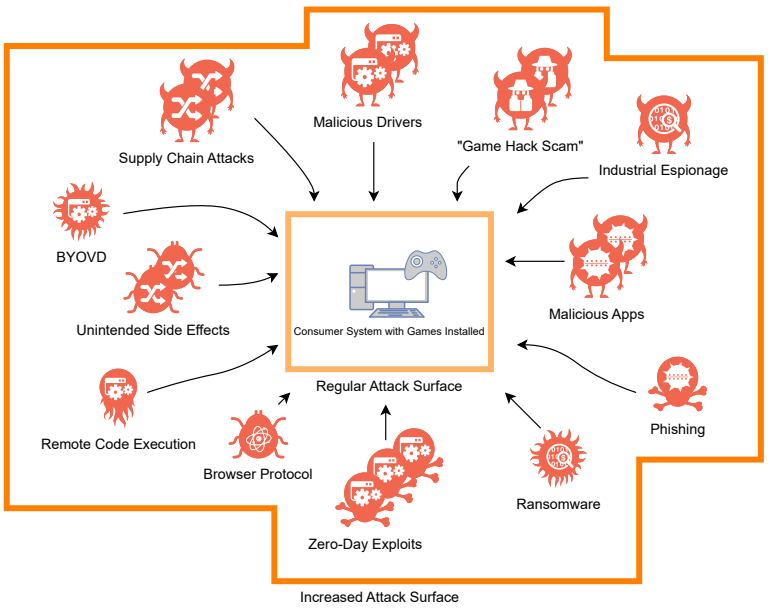

Figure 3.9: Increase of attack surface when games are present on a machine.

3.7.1 Research Question 1

Which exploits exist that leverage vulnerabilities of video games to target the underlying system?

This question can be answered in light of the previous findings. As described, exploits exist for the following targets:

- **Cheats:** Reports exist of scams that promise cheat programs to victims; accepting such an offer usually leads to the installation of malware [69].
- **Anti-cheat solutions:** An instance of a BYOVD attack utilizing Anti-Cheat Expert's kernel driver was reported [11].
- **Cracks:** A study shows that more than half of available cracks contain malware, either added by the creators or as part of the packing process [70]. Similar scams as the one described for cheats above exist for cracks as well [69].
- **Mods:** Examples for vulnerabilities introduced by side effects [78, 19] and zero-day exploits exist [79]. Additionally, an example for a supply chain attack was reported [20].
- **Launchers:** Exploits abusing side effects in custom browser protocols were reported [12, 81], as well as an attack taking advantage of mishandled privileges [17].

Further, reports exist of compromised developer accounts [82] and intentionally uploaded malware disguised as a game [83].

- **Engines:** Reports of numerous zero-day exploits exist, plaguing different engines [14, 80]; some of them leading to remote code execution [84, 16] or player bans [15], others to arbitrary file writes [13].
- **Supply chain:** Multiple developers have reported data breaches compromising player data [21, 127, 22]. There is also evidence of phishing campaigns targeting gamers specifically [40].

To answer the second research question, the next chapter introduces a threat model and follows it with a proof of concept for an exploit.

CHAPTER 4

Threat Model and Proof of Concept

4.1 Overview

The attack surface created by the presence of installed video games on a machine can be associated with a threat model. It is the combination of all the possible threats faced by users of such a machine. Video games consist of many interdependent parts, all of which can be vulnerable to attacks and thus contribute to the following model.

1. **Threats posed by the core application:** Installing a video game on a machine, just like any other application, can introduce vulnerabilities inherent to its engine. These consist of zero-day exploits and unintended side effects. Depending on the vulnerability's severity, an attacker might be able to extract system information, deploy malware, or take over the machine. The only way to avoid these dangers is to refrain from installing and playing the game.

   Another aspect in this category concerns attacks on a game's developer: Data collected about consumers might put their machines in jeopardy as well if they are leaked. Again, the only way to circumvent this possibility is to not play the game.

2. **Threats posed by optional requirements not under the user's control:** Depending on the game and its developer, some vulnerabilities might be introduced by components that are not strictly necessary to play the game but are used for distribution, cheating countermeasures, and authentication.

   Launchers, serving as both libraries and shops, are external, stand-alone applications that usually need to run in the background while a game is played. As such, they introduce their own set of possible vulnerabilities; zero-day exploits and unintended

side effects (like misuse of associated browser protocols) are one aspect. The other concerns their function as distribution platforms: Supply chain attacks targeting legitimate developers can potentially affect every player of a game. Such an attack might modify game files to contain malware which is then automatically pushed to users' machines via the launcher. Similarly, malicious actors might straight up publish a game containing arbitrary vulnerabilities, taking advantage of consumers' goodwill towards a distribution platform.

Anti-cheat measures, common in competitive online games, are especially problematic if they introduce kernel-level drivers with vulnerabilities; these might be used in BYOVD (Bring Your Own Vulnerable Driver) Attacks. Depending on the vulnerability, they might lead to a complete compromise of the machine.

Many games also require a user account as an authentication mechanism. These accounts can fall victim to social engineering attacks, leading to further exploits down the line.

Sometimes the threats in this category can be avoided: by choosing to buy a game's variant not requiring a specific launcher or account creation. Unfortunately, in many instances consumers do not have a choice, especially on the topic of anti-cheat. As before, the only way to completely avoid these dangers would be to avoid playing the game.

3. **Threats posed by user-controlled content:** Many games can be augmented with community-created modifications. Crossing the borders of legality, users my also choose to augment them with cheats to gain a competitive advantage or to downright steal them by acquiring a cracked version.

   Cheats and cracks both pose very similar threats. Due to their illegal nature and that of the communities providing them there is a good chance for both to contain malicious code regardless of their functionality. Further, even if the original creators' intent was "honest" (i.e. providing a cheat or crack that does nothing more than advertised), the distribution networks might add malicious code upon (re)packing. Finally, illicit software like this is often part of scams that promise cheats or a crack for a game but actually supply their victims with unrelated malware instead. Since this kind of software is installed by the user directly, any form of exploit is possible. Avoiding the associated threats is straightforward: Neither cheats nor cracks are necessary to play a game so users might just skip them altogether.

   Mods are different, being not only tolerated by developers but often times openly endorsed and encouraged. Unfortunately they can suffer from the same vulnerabilities as the games they augment; these again include unintended side effects and zero-day exploits, only in this case pertaining to whatever technology was used to create the mod. Like games, mods might fall victim to supply chain attacks as well. Also it is a very real possibility for malicious actors to create their own mods containing exploitable vulnerabilities. Being optional, the easiest course of action to avoid threats posed by game modifications is to avoid mods altogether; however,

trustworthy communities exist that provide well-vetted code, making it possible for users to keep risks to a minimum.

As with potential vulnerabilities, different motivations for attacking a system exist, corresponding with different methods of attack [128]:

- **Novices/Cyberpunks** - Characterized by their low to medium skill level, these hackers are motivated by curiosity and recreational reasons. They use tools provided by others, sometimes with slight modifications. In a video game context, hacking an opponent out of revenge or exploring the possibilities of modding a game might be done by a novice/cyberpunk. Examples given in this thesis include abusing Steam's browser protocol or a mod's unchecked DLL loading.
- **Professionals/Nation States** - This category includes highly skilled individuals, working either as mercenaries or for a nation state. They actively research exploits and create their own tools. These actors certainly have the means to analyze and effectively exploit any vulnerabilities they might find in video games. BYOVD attacks and sophisticated privilege escalation (e.g. exploiting the Steam launcher's installation process) signify their modus operandi.
- **Petty Thieves/Digital Pirates** - Individuals in this category include criminals who move their illegal activities online (e.g. scammers and fraudsters), as well as copyright infringers who illegally duplicate copyrighted material. Their motivation is of a financial nature and their skill level varies. Petty thieves might engage in activities like the game hack scam, while digital pirates actively work on circumventing any form of copy protection.
- **Insiders** - This group comprises of disgruntled employees motivated by financial gain and ideology. Not necessarily hackers at all, they use existing privileges or insider knowledge to gain access to a system. Data leaks often times are caused by members of this group.

This threat model – the combination of threats and attackers/motivations for attack – shows video games to be attractive targets for a variety of individuals. Not only is there a large potential for exploits and security flaws, video games also may come under attack for different motivational reasons, increasing their attack surface even further.

For a proof of concept, the threat model is narrowed down to exclude aspects considered out of scope: analysis of engine and launcher code (because of its complexity), any form of social engineering (which would require carefully chosen test subjects or a developer's cooperation), as well as illegal undertakings like creating a malicious game or mod.

The technique chosen is *DLL search order hijacking* because it exemplifies how easily a game's process can be abused to achieve arbitrary code execution. It works by placing a malicious library on the target machine which is then loaded at a program's start (in

this case a video game). This ensures that whatever code from the library is executed has the same privileges as the program that called it. The DLL serves as a placeholder for arbitrary malicious code as described in the threat model; the question of delivery, i.e. how the malicious library is placed on the target, is assumed out of scope. However, the threat model outlines several possibilities:

- The file can be hidden in a mod package.
- The file can be distributed as part of a cheat or crack.
- The file might be placed via a social engineering technique (e.g. a scam offering cosmetic items).
- The file can be published to a poisoned repository.
- The file can be made part of a game's update using the developer's credential following a data leak.

A very promising form of delivery – abusing side effects of the Steam Browser Protocol – does not work anymore at the time of writing, as was already mentioned previously.

4.2 DLL Search Order Hijacking

The Windows operating system offers the possibility for modules (i.e. programs) to load certain needed functions at load time or run time. This is achieved using so-called *Dynamic-Link Libraries* or *DLL*s. From the official Microsoft documentation: "Dynamic linking allows a module to include only the information needed to locate an exported DLL function at load time or run time. Dynamic linking differs from the more familiar static linking, in which the linker copies a library function's code into each module that calls it." [129]

The advantages of this dynamic linking are [130]:

- System memory needed is reduced since multiple processes loading the same DLL share a single copy.
- Changing a DLL function does not require a change within the applications using it, as long as the signature does not change. This facilitates code refactoring.
- DLLs are independent of an application's programming language, making them a flexible tool.

Unfortunately, from a security perspective there exists a notable disadvantage: the dynamic-link library search order. Because multiple versions of a DLL may exist in different folders on one machine, it is necessary to specify which version takes priority

when a program is executed. This can be done manually by specifying a full path; however, it is common to rely on the operating system's predefined search order, taking advantage of its flexibility. It is as follows [131]:

1. DLL Redirection.
2. API sets.
3. SxS manifest redirection.
4. Loaded-module list.
5. Known DLLs.
6. The package dependency graph of the process (Windows 11 version 21H2 and later).
7. The folder from which the application loaded.
8. The system folder.
9. The 16-bit system folder.
10. The Windows folder.
11. The current folder.
12. The directories listed in the "PATH" environment variable.

Many applications rely on certain DLLs to already exist on the machines they are being installed on. Therefore, to minimize folder size, these DLLs are not included in the installation path; instead, when the executable is invoked, the search order described above makes sure that when they are not found in the installation folder they are then loaded from one of the locations that is searched after that (most commonly the system folder).

If this is the case, an attacker identifying a program relying on Windows' dynamic-link library search order may try to place a malicious DLL directly in the program's folder. Because it then takes priority it will be loaded upon the application's execution and its code executed with the same privileges as the invoking executable. Moreover, the executable (being a legitimate program) may be signed and thus bypass any endpoint security measures altogether [132]. The *MITRE ATT&CK Techniques* catalog further states: "If a search order-vulnerable program is configured to run at a higher privilege level, then the adversary-controlled DLL that is loaded will also be executed at the higher level. In this case, the technique could be used for privilege escalation from user to administrator or SYSTEM or from administrator to SYSTEM, depending on the program. Programs that fall victim to path hijacking may appear to behave normally because malicious DLLs may be configured to also load the legitimate DLLs they were meant to replace." [133]

DLL search order hijacking is suited well to exploit video games not only because of their privileges: Crafting a malicious DLL also requires an attacker to have knowledge about the target's folder structure as well as potentially exploitable DLLs, i.e. libraries that are expected by the game to exist on the victim's system already. When installing a video game via the Steam launcher, its installation folder always has a certain path structure, making it easy for an attacker to find where to put their payload. Further, since game installations usually are not customizable, an attacker is able to extensively test their exploit on their own machine, being confident to find similar conditions on their victim's system.

4.3 Proof of Concept

As a proof of concept two different DLLs were implemented, targeting two different applications:

1. The Steam launcher,
2. The game *Into the Breach*, launched via Steam.

In both cases the process of identifying and creating the exploit is the same. The operating system used is Windows 11.

In a first step, *Task Manager* is used to inspect the applications' privileges. This is done via the "Security" tab of the "Properties" dialogue.

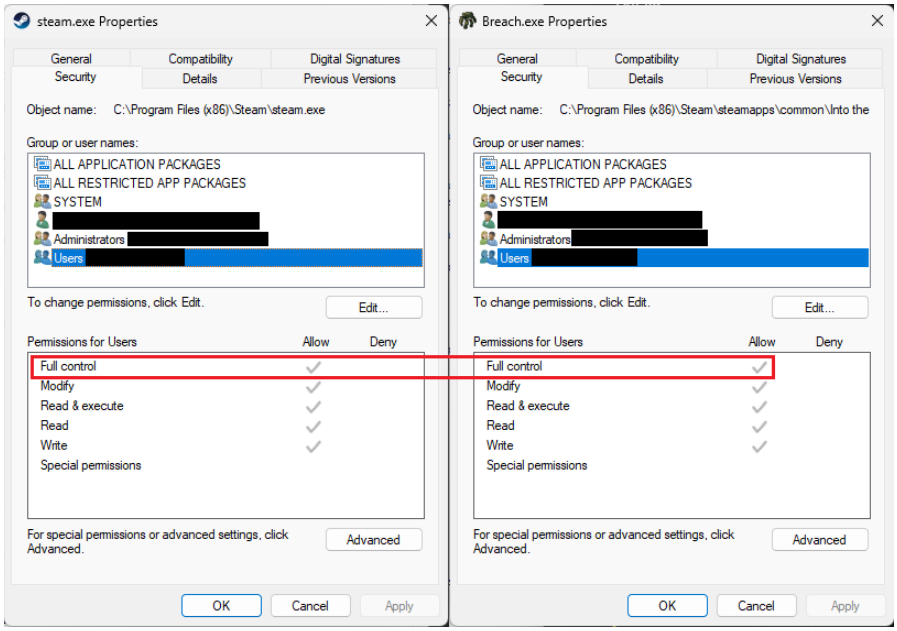

Figure 4.1: User Permissions for steam.exe and Breach.exe.

As can be seen in Figure 4.1, both applications grant users full control which, from an attacker's point of view, is desirable. Next, *Process Monitor* is configured by adding some filters; the caption's result should include processes named "Steam.exe" and "Breach.exe", paths shown should end with ".dll" and the result should be "NAME NOT FOUND". The last filter makes sure that only DLLs that are called by the process but are not present in the installation path are shown.

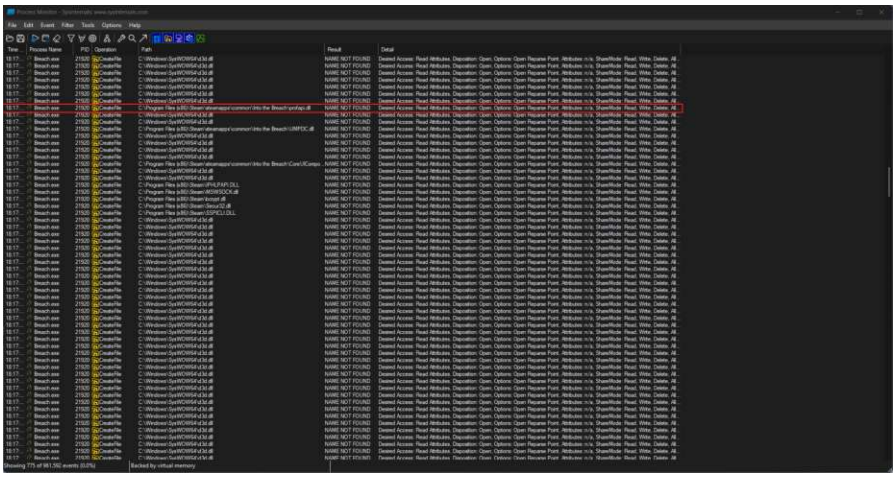

Figure 4.2: Results of Process Monitor Filtering.

Figure 4.2 shows an example output with the aforementioned filters configured. Attackers usually focus on such DLLs as are not found in the installation folder, for example the highlighted "profapi.dll". For this proof of concept two very simple DLLs were compiled. Both utilize the `DLLMain` function; once to spawn a message box that reads "PoC!" and once to open a command line window. Naming the resulting DLL "profapi.dll" and placing it in the installation folders of Steam and Into the Breach yields promising results: in both cases the injected code is executed, the latter leading to a command line window ready to execute arbitrary commands with the current user's rights (figures 4.3, 4.4).

Figure 4.3: Message Box on Execution of Into the Breach.

4.4 Summary

This proof of concept shows that creating an exploit for a game is as simple as finding a suitable DLL. Since path structures tend to be similar for games installed on different machines, placement is not a problem once the file is on the victim's PC. As for the

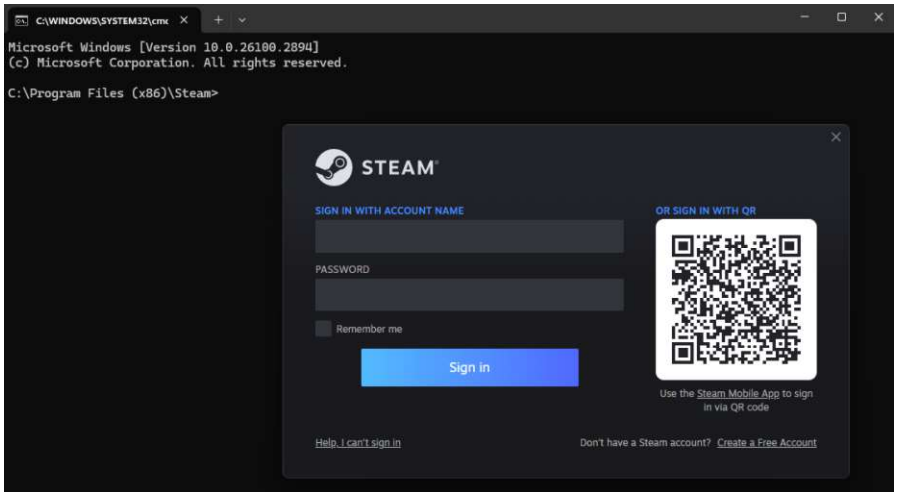

Figure 4.4: Command Line Window on Execution of Steam.

question of payload delivery, plenty of options exist; depending on the resources and capabilities of the attacker some simple social engineering might suffice to infect the first machine.

Video games' large attack surface combined with the relative ease with which an exploit can be created leads to the conclusion that video game security should not be taken lightly; neither by consumers nor developers.

4.4.1 Research Question 2

Is it possible to reproduce known vulnerabilities or finding new ones [...]?

As the above shows it is possible to find vulnerabilities. Unfortunately, trying to recreate Steam's browser protocol vulnerability was not possible due to it having been patched appropriately, making silent placement of files impossible.

The previous chapters show that video games may be exploited in many different ways due to their large and divers attack surface. This leads to many possibilities in terms of follow-up research. The next (and final) chapter will describe these possibilities as well as recapitulate key findings pertaining to vulnerabilities and exploitability of video games.

CHAPTER 5

Conclusion and Future Work

The main goal of this thesis is to show that video games pose a considerable security risk to the system they are running on and that this fact warrants more attention from both developers and researchers. Games are complex pieces of software with many interconnected parts and stakeholders; it is this complexity that makes analysis hard and mitigation even harder.

Two questions (formulated more precisely as research questions in the first chapter) need to be answered in order to show this result:

1. Which exploits happened in the past?
2. How hard is it to create new ones?

The answer to the first question, together with a description of different technologies' interconnectedness, gives a good overview of how video games add to a system's attack surface. It is thus: The increase of attack surface caused by video games is substantial. This is in part due to the aforementioned interplay of technologies and in part due to accompanying factors like lack of research or security standards in general (detailed in chapter three).

Paired with an appropriate threat model, the second question helps to understand the probability with which malicious actors might try to exploit weaknesses in video games. Chapter four provides both the threat model and the answer to the question: The threshold for creating new exploits is relatively low. The fact that not much evidence for breaches utilizing video games exist either means that adversaries concentrate on opportunities that are even easier to exploit or that most incidents go unnoticed.

The conclusion from the two answers is clear: Video games have a large attack surface that can be exploited relatively easily. As stated previously, the fact that few academic

sources exist on the topic makes matters worse. Consequently, numerous opportunities for follow-up research present themselves; the following section gives an overview.

5.1 Future Work

This section follows the same structure as previous chapters, with potential research topics clustered within the appropriate category.

5.1.1 Cheating and Anti-Cheat

The most obvious candidate warranting further research is Anti-Cheat Expert's vulnerable kernel driver documented as CVE-2024-22830. While out of scope for this thesis, creating a proof of concept for an exploit could further the understanding of potential vulnerabilities of anti-cheat solutions operating on the kernel level. It is also of interest how fast (if at all) the vendor reacts to the disclosure of a CVE (more on that later).

Similar to the Malware-as-a-Service principle, there exists a whole economy dedicated to the creation and distribution of video game cheats. The security community could benefit from an up-to-date analysis of this niche; Which actors are involved? What are the financial quantities? What are the distribution channels? It might also be worthwhile to take a look at a cheat developer's work routine as part of a social study. Studying the cheat ecosystem could further provide answers to questions of consumer safety.

Another topic for further work is the hinted-at arms race between developers of cheats and providers of anti-cheat solutions. The current state of the art can provide insights into future trends and the security concerns associated with them. The security standards of anti-cheat measures are also worth looking at in detail.

Finally, the topic of artificial intelligence and its role in both cheat and anti-cheat development is something that should be explored in the future.

5.1.2 Cracks and DRM

The principles of cheat development and cracking games are very similar; the same study concerning the economy driving the cheat market can and should be done for the market of video game cracks. Especially the risks of engaging in illegal activities like acquiring software cracks as a user stand to be made more transparent.

Obviously current DRM measures should also be studied with respect to their security and potential vulnerabilities.

Another study might be conducted to determine what role the damage done by malware disguised as a video game crack plays in the discussion of the benefits of digital rights management.

5.1.3 Modding

Modding, due to its ungoverned nature, poses a big security risk as well and is almost completely absent from the body of security research. It would be interesting to do a survey of distribution platforms for mods, examining details like file types used, security and auditing standards, as well as developer maturity and security knowledge. Also, how do consumers perceive these threats? Which guarantees do users expect? Are they able to formulate educated requests pertaining to the safety of mods?

Another survey might be done on the mod landscape in general: How many mods exist for how many games? How well are they protected against supply chain attacks or repository poisoning? This could also facilitate the assessment of the threat posed by mods and of the steps needed to be taken to alleviate potential problems.

5.1.4 Launchers

Three topics warranting in-depth analysis immediately stand out: various launchers' browser protocols, the issue tracked as CVE-2020-15530, and different launchers themselves.

As stated previously, many launchers introduce their own browser protocol. And while many of the reported vulnerabilities pertaining to same have been patched over time, there still remains a possibility that some overlooked flaws remain to this day. A technical study of different launchers and their browser protocols could help assess this threat.

Steam had a vulnerability reported in 2020, tracked as CVE-2020-15530, that allows a local attacker to install arbitrary files in an elevated context; it is still open at the time of writing, with the last update posted in 2024. Researching this topic might not only confirm an ongoing security flaw within Steam but might also show how other launchers suffer from similar problems.

In general it seems obvious to carefully analyze existing launchers, especially concentrating on their networking components and system privileges. Additionally, a study of enforced security standards for publishers might yield interesting insights. To gauge the threat landscape it might also pay to do a survey of historical data, summarizing exploits and vulnerabilities per launcher, as well as number of machines affected and size of inventory.

5.1.5 Engines

The research done on engines mentioned in earlier chapters was conducted in 2012; a similar endeavor analyzing state-of-the-art technology is in order.

From a technical perspective it might be interesting what can be done with shader files, since many engines let developers write and bundle their own.

Another worthwhile line of inquiry is the question which third-party software is used by different engines and if there are potential supply chain weaknesses in general.

Finally, a study of security features offered to developers by different engines might help assess the degree of security awareness within this particular community.

5.1.6 Data Leakage and Privacy

Since data leaks are not uncommon among game developers, it would be interesting to analyze which insights can be gleaned from the data collected. On one hand leaked data might be used for further exploits, like taking advantage of password reuse or utilizing them in a phishing campaign. On the other hand, players' privacy might be in danger if leaked data allows for tracking their online movements.

In general, a survey of different companies' data collection and usage would help evaluate both privacy and security concerns.

5.1.7 General

The general structure of this thesis might be utilized further researching different gaming technologies; examples include console and mobile games, games for Linux, and – with a narrower focus – gambling applications. It might also be applied to focus exclusively on the topic of cheats or launchers.

Another important aspect of video games is their community and the potential for social engineering that comes with it. Further, even though it was mentioned in conjunction with different aspects of video games already, the general state of video game security and its role and perception in development and marketing is worth to be studied.

It also makes sense to look at things from the perspective of a malware developer: What are the advantages and drawbacks of producing actively malicious video games? What is the potential reach? How about the cost-benefit-analysis?

Additionally, there are many comparisons to be drawn between games and non-gaming software:

- What is the severity of data leaks? How many cases have become public?
- How often do bugs break the application? Are video game users more tolerant of faulty behavior?
- What kind of machine does the software typically run on?
- Do games have different privileges than other software?
- How many different components make up the software? How big is the attack surface?
- What is the average size of the development studio? Do games have smaller development teams?

- Are games being pirated more often than other software? How high are penalty payments usually?
- How many CVEs (or similar) exist? How fast does the industry usually react with patches? How high are bug bounties?
- How safe are cracked versions?
- What is the state of coding and security best practices?

All of these help distinguish games from other software and would go towards further understanding the additional risk of installing video games on a machine.

A final question, warranting detailed research (because it goes beyond the sphere of gaming), is the following: In trying to compromise a user's work account, how much of an advantage does the compromise of that same user's private machine signify? This obviously directly concerns the results of this thesis but can also give valuable insight into best practices when it comes to separating private and work machines in general.

5.2 Final Thoughts

As can be seen, there are many topics related to video games that are worthy of further research. In keeping with its initial goal, this thesis provides evidence that video games are a significant, yet underexplored security risk. By showing video games' divers and – above all – large attack surface, it aims at raising awareness of that fact among the research community, pointing out aspects in need of improvement. It is my hope that this work can serve as both a valuable overview and starting point for future research that follows a similar goal: to improve a beloved field and help make video games more secure.

List of Figures

List of Tables

Bibliography

[1] Grand View Research, "Video game market summary." https://www.grandviewresearch.com/industry-analysis/video-game-market, 2025. [Online; accessed 23-June-2025].

[2] F. Duarte, "The latest pc gaming statistics (2025)." https://explodingtopics.com/blog/pc-gaming-stats, 2025. [Online; accessed 23-June-2025].

[3] SteamDB, "Platforms." https://steamdb.info/instantsearch/, 2025. [Online; accessed 23-June-2025].

[4] Microsoft, "Local accounts." https://learn.microsoft.com/en-us/windows/security/identity-protection/access-control/local-accounts, 2025. [Online; accessed 23-June-2025].

[5] SplicedOnline Team, "Does easyanticheat need admin?." https://splicedonline.com/does-easyanticheat-need-admin/, 2024. [Online; accessed 23-June-2025].

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

[7] D. Alder, "What kernel–level anti–cheat is and why you should care." https://levvvel.com/what-is-kernel-level-anti-cheat-software/, 2023. [Online; accessed 13-February-2025].

[8] C. Dorner and L. D. 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*, pp. 1–11, 2024.

[9] S. Motiee, K. Hawkey, and K. Beznosov, "Do windows users follow the principle of least privilege? investigating user account control practices," in *Proceedings of the Sixth Symposium on Usable Privacy and Security*, pp. 1–13, 2010.

[10] Playgama Blog, "How can running my game as an administrator affect file access and permissions in windows 10?." https://playgama.com/blog/general/how-can-running-my-game-as-an-administrator-affect-file-access-and-per 2025. [Online; accessed 23-June-2025].

[11] Defence Tech, "Cve-2024-22830 vulnerability analysis report." https://www.defencetech.it/wp-content/uploads/2024/04/Report-CVE-2024-22830.pdf, 2024. [Online; accessed 03-March-2025].

[12] L. Auriemma and D. Ferrante, "Steam browser protocol insecurity." http://revuln.com/files/ReVuln_Steam_Browser_Protocol_Insecurity.pdf, 2012. [Online; accessed 13-February-2025].

[13] floesen, "Cve-2021-30481: Source engine remote code execution via game invites." https://secret.club/2021/04/20/source-engine-rce-invite.html, 2021. [Online; accessed 23-February-2025].

[14] L. Auriemma and D. Ferrante, "Game engines: A 0-day's tale." http://revuln.com/files/ReVuln_Game_Engines_0days_tale.pdf, 2013. [Online; accessed 23-February-2025].

[15] B. Toulas, "Gta online bug exploited to ban, corrupt players' accounts." https://www.bleepingcomputer.com/news/security/gta-online-bug-exploited-to-ban-corrupt-players-accounts/, 2023. [Online; accessed 08-April-2025].

[16] A. Chalk, "Easy anti-cheat washes its hands of the apex legends hacking disaster that saw streamer accounts hijacked live: 'there is no rce vulnerability within eac'." https://www.pcgamer.com/games/battle-royale/easy-anti-cheat-washes-its-hands-of-the-apex-legends-hacking-disaster- 2024. [Online; accessed 08-April-2025].

[17] D. Gebert, "Steam - escalation of privileges." http://daniels-it-blog.blogspot.com/2020/07/steam-arbitrary-code-execution-part-2.html, 2020. [Online; accessed 22-February-2025].

[18] CVE, "Cve-2024-22830." https://www.cve.org/CVERecord?id=CVE-2024-22830, 2024. [Online; accessed 01-April-2025].

[19] D. Lodge, "Rce on steam through the cultist simulator game." https://www.pentestpartners.com/security-blog/rce-on-steam-through-the-cultist-simulator-game/, 2023. [Online; accessed 03-March-2025].

[20] Ashwin, "Hackers uploaded malware through a popular game mod on steam." https://www.ghacks.net/2023/12/28/hackers-uploaded-malware-through-a-popular-game-mod-on-steam/, 2023. [Online; accessed 03-March-2025].

[21] Flashpoint, "Combining cybersecurity with gaming: Cheats, insider threats, ransomware, and more." https://flashpoint.io/blog/cybersecurity-gaming-insider-threats-ransomware/#account-takeover, 2022. [Online; accessed 04-March-2025].

[22] C. Reed, "Sony data breaches: Full timeline through 2023." https://firewalltimes.com/sony-data-breach-timeline/, 2023. [Online; accessed 04-March-2025].

[23] A. Truelove, E. S. de Almeida, and I. Ahmed, "We'll fix it in post: What do bug fixes in video game update notes tell us?," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 736–747, IEEE, 2021.

[24] J. Backus, "Players' perception of bugs and glitches in video games: An exploratory study," *arXiv preprint arXiv:2504.15408*, 2025.

[25] S. Baird, "Pokemon finally acknowledges a decades-old glitch beloved by fans." https://www.dexerto.com/pokemon/pokemon-finally-acknowledges-a-decades-old-glitch-beloved-by-fans-2624505/, 2024. [Online; accessed 23-June-2025].

[26] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, *et al.*, "Sok:(state of) the art of war: Offensive techniques in binary analysis," in *2016 IEEE symposium on security and privacy (SP)*, pp. 138–157, IEEE, 2016.

[27] A. Ansaria, "Evaluation of the cyber security models implemented across common attack vectors: A review of literature," *World Journal of Advanced Engineering Technology and Sciences*, vol. 11, no. 1, pp. 064–068, 2024.

[28] C. Patsakis, D. Arroyo, and F. Casino, "The malware as a service ecosystem," in *Malware: Handbook of Prevention and Detection*, pp. 371–394, Springer, 2024.

[29] C. Beaman, A. Barkworth, T. D. Akande, S. Hakak, and M. K. Khan, "Ransomware: Recent advances, analysis, challenges and future research directions," *Computers & security*, vol. 111, p. 102490, 2021.

[30] A. C. Eberendu, V. I. Udegbe, E. O. Ezennorom, A. C. Ibegbulam, T. I. Chinebu, *et al.*, "A systematic literature review of software vulnerability detection," *European Journal of Computer Science and Information Technology*, vol. 10, no. 1, pp. 23–37, 2022.

[31] R. Amankwah, P. K. Kudjo, and S. Y. Antwi, "Evaluation of software vulnerability detection methods and tools: a review," *International Journal of Computer Applications*, vol. 169, no. 8, pp. 22–27, 2017.

[32] S. Mohr and S. S. Rahman, "It security issues within the video game industry," *International Journal of Computer Science & Information Technology*, vol. 3, no. 5, p. 1, 2011.

[33] K. S. Szatmáry, "Cybersecurity of the gaming industry," in *2024 IEEE 22nd Jubilee International Symposium on Intelligent Systems and Informatics (SISY)*, pp. 000441–000446, IEEE, 2024.

[34] A. Ibrahim, "Guarding the future of gaming: The imperative of cybersecurity," in *2024 2nd International Conference on Cyber Resilience (ICCR)*, pp. 1–9, IEEE, 2024.

[35] H. Bennani, "Cybersecurity, cybercrime and the video gaming industry," Master's thesis, Utica University, 2022.

[36] J. K. Vanover, *Exploring the cyber security improvements needed by internet game users to reduce cyber security threats*. PhD thesis, Colorado Technical University, 2018.

[37] G. Sharadin, "Leveling up security: Understanding cyber threats in the gaming industry." https://www.imperva.com/blog/understanding-cyber-threats-in-gaming/, 2024. [Online; accessed 06-April-2025].

[38] A. Zoosman and V. Kivilevich, "Darknet threat actors are not playing games with the gaming industry." https://www.kelacyber.com/blog/darknet-threat-actors-are-not-playing-games-with-the-gaming-industry/, 2021. [Online; accessed 04-March-2025].

[39] U. Altafullah, "Gaming industry cybersecurity risks and how to prepare for them." https://www.sealingtech.com/blog/gaming-industry-cybersecurity-risks-and-how-to-prepare-for-them/, 2022. [Online; accessed 05-April-2025].

[40] Kaspersky, "Gaming-related cyberthreats in 2023: Minecrafters targeted the most." https://securelist.com/game-related-threat-report-2023/110960/, 2023. [Online; accessed 04-March-2025].

[41] L. Chen, N. Shashidhar, D. Rawat, M. Yang, and C. Kadlec, "Investigating the security and digital forensics of video games and gaming systems: A study of pc games and ps4 console," in *2016 International Conference on Computing, Networking and Communications (ICNC)*, pp. 1–5, IEEE, 2016.

[42] S. J. Lee, E. J. Jeong, D. Y. Lee, and G. M. Kim, "Why do some users become enticed to cheating in competitive online games? an empirical study of cheating focused on competitive motivation, self-esteem, and aggression," *Frontiers in psychology*, vol. 12, p. 768825, 2021.

[43] M. Consalvo, "Gaining advantage: How videogame players define and negotiate cheating.," in *DiGRA Conference*, 2005.

[44] V. H. H. Chen and J. Ong, "The rationalization process of online game cheating behaviors," *Information, Communication & Society*, vol. 21, no. 2, pp. 273–287, 2018.

[45] A. Boldi and A. Rapp, ""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*, vol. 40, no. 15, pp. 4021–4041, 2024.

[46] J. Blackburn, N. Kourtellis, J. Skvoretz, M. Ripeanu, and A. Iamnitchi, "Cheating in online games: A social network perspective," *ACM Transactions on Internet Technology (TOIT)*, vol. 13, no. 3, pp. 1–25, 2014.

[47] K. R. Hamlen, "Academic dishonesty and video game play: Is new media use changing conceptions of cheating?," *Computers & Education*, vol. 59, no. 4, pp. 1145–1152, 2012.

[48] S. Cho, J. Lusthaus, and I. Flechais, "The slippery slope: Exploring the parallels between game cheating and cybercrime through routine activity theory," in *2023 APWG Symposium on Electronic Crime Research (eCrime)*, pp. 1–13, IEEE, 2023.

[49] S. Pontiroli, "The cake is a lie! uncovering the secret world of malware-like cheats in video games," in *Virus Bulletin Conference*, 2019.

[50] B. Wodecki, "Ai is driving the next generation of video game cheats & exploits." https://aibusiness.com/computer-vision/ai-is-driving-the-next-generation-of-video-game-cheats-and-exploits, 2021. [Online; accessed 08-April-2025].

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

[52] Starz0r, "Are we anti-cheat yet?." https://areweanticheatyet.com/, 2025. [Online; accessed 08-April-2025].

[53] A. Jonnalagadda, I. Frosio, S. Schneider, M. McGuire, and J. Kim, "Robust vision-based cheat detection in competitive gaming," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 4, no. 1, pp. 1–18, 2021.

[54] R. Spijkerman and E. 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*, pp. 87–92, 2020.

[55] P. Karkallis, J. Blasco, G. Suarez-Tangil, and S. Pastrana, "Detecting video-game injectors exchanged in game cheating communities," in *Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part I 26*, pp. 305–324, Springer, 2021.

[56] J. P. Pinto, A. Pimenta, and P. Novais, "Deep learning and multivariate time series for cheat detection in video games," *Machine Learning*, vol. 110, no. 11, pp. 3037–3057, 2021.

[57] M. S. Anwar, C. Zuo, C. Yagemann, and Z. Lin, "Extracting threat intelligence from cheat binaries for anti-cheating," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 17–31, 2023.

[58] C. Sun, K. Ye, L. Su, J. Zhang, and C. Qian, "Invisibility cloak: proactive defense against visual game cheating," in *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 3045–3061, 2024.

[59] B. Bryant and H. Saiedian, "An evaluation of videogame network architecture performance and security," *Computer Networks*, vol. 192, p. 108128, 2021.

[60] M. M. Azad, A. H. S. Ahmed, and A. Alam, "Digital rights management," *IJCSNS*, vol. 10, no. 11, pp. 24–26, 2010.

[61] O. Mihus, "Strengthening intellectual property protection in the eu: It law and its impact on the computer games industry," *Public Administration and Law Review*, no. 4 (20), pp. 20–34, 2024.

[62] A. V. Moshirnia, "Giant pink scorpions: Fighting piracy with novel digital rights management technology," *DePaul J. Art Tech. & Intell. Prop. L*, vol. 23, p. 1, 2012.

[63] D. Quarta, M. Ianni, A. Machiry, Y. Fratantonio, E. Gustafson, D. Balzarotti, M. Lindorfer, G. Vigna, and C. Kruegel, "Tarnhelm: Isolated, transparent & confidential execution of arbitrary code in arm's trustzone," in *Proceedings of the 2021 Research on offensive and defensive techniques in the Context of Man At The End (MATE) Attacks*, pp. 43–57, 2021.

[64] P. Rubin, "A brief history of drm in video games." https://promptingculture.substack.com/p/a-brief-history-of-drm-in-video-games, 2024. [Online; accessed 06-April-2025].

[65] J. Kunat, "What exactly is drm in video games and why should you care?." https://www.gog.com/blog/what-exactly-is-drm-in-video-games-and-why-should-you-care/, 2024. [Online; accessed 06-April-2025].

[66] C. W. Hill, "Digital piracy: Causes, consequences, and strategic responses," *Asia Pacific Journal of Management*, vol. 24, pp. 9–25, 2007.

[67] A. Sundararajan, "Managing digital piracy: Pricing and protection," *Information Systems Research*, vol. 15, no. 3, pp. 287–308, 2004.

[68] S. Kumar, L. Madhavan, M. Nagappan, and B. Sikdar, "Malware in pirated software: Case study of malware encounters in personal computers," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pp. 423–427, IEEE, 2016.

[69] E. Badawi, G.-V. Jourdan, G. Bochmann, and I.-V. Onut, "Automatic detection and analysis of the "game hack" scam," *Journal of Web Engineering*, vol. 18, no. 8, pp. 729–760, 2019.

[70] M. Kammerstetter, C. Platzer, and G. Wondracek, "Vanity, cracks and malware: Insights into the anti-copy protection ecosystem," in *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 809–820, 2012.

[71] S. Kaushik, "A game of digital piracy: The network effect," *EUROPEAN INTELLECTUAL PROPERTY REVIEW*, vol. 46, no. 2, pp. 114–123, 2024.

[72] S. Goode and S. Cruise, "What motivates software crackers?," *Journal of Business Ethics*, vol. 65, pp. 173–201, 2006.

[73] A. Dewalska-Opitek and M. Hofman-Kohlmeyer, "Players as prosumers-how customer engagement in game modding may benefit computer game market.," *Central European business review*, vol. 10, no. 2, 2021.

[74] F. Walsdorff, "Video game modding and money. from precarious playbor to reimbursed labor of love," Medienwissenschaftliches Seminar Uni Siegen, 2022.

[75] E. Weisdorfer, "Transformative play: The legalities of modding in the video game industry," *Cybaris®*, vol. 16, no. 1, p. 3, 2024.

[76] J. Curtis, G. Oxburgh, and P. Briggs, "Heroes and hooligans: the heterogeneity of video game modders," *Games and Culture*, vol. 17, no. 2, pp. 219–243, 2022.

[77] G. Liu, D. Liu, S. Hao, X. Gao, K. Sun, and H. Wang, "Ready raider one: Exploring the misuse of cloud gaming services," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1993–2007, 2022.

[78] D. Lodge, "Sandbox vuln in tales of maj'eyal." https://www.pentestpartners.com/security-blog/sandbox-vuln-in-tales-of-majeyal/, 2016. [Online; accessed 03-March-2025].

[79] J. Vojtěšek, "Dota 2 under attack: How a v8 bug was exploited in the game." https://decoded.avast.io/janvojtesek/dota-2-under-attack-how-a-v8-bug-was-exploited-in-the-game/, 2023. [Online; accessed 03-March-2025].

[80] L. Auriemma and D. Ferrante, "Exploiting game engines for fun & profit." https://revuln.com/files/Ferrante_Auriemma_Exploiting_Game_Engines.pdf, 2013. [Online; accessed 23-February-2025].

[81] L. Auriemma and D. Ferrante, "Ea origin insecurity." http://revuln.com/files/ReVuln_EA_Origin_Insecurity.pdf, 2013. [Online; accessed 22-February-2025].

[82] T. Wilde, "Valve adds new security check after attackers compromise steam accounts of multiple game devs and update their games with malware." https://www.pcgamer.com/steam-malware-attack-new-security/, 2023. [Online; accessed 01-April-2025].

[83] B. Toulas, "Steam pulls game demo infecting windows with info-stealing malware." https://www.bleepingcomputer.com/news/security/steam-pulls-game-demo-infecting-windows-with-info-stealing-malware/, 2025. [Online; accessed 01-April-2025].

[84] W. Tremblay, "Documentation and proof of concept code for cve-2022-24125 and cve-2022-24126." https://github.com/tremwil/ds3-nrssr-rce, 2022. [Online; accessed 08-April-2025].

[85] J. L. Kröger, P. Raschke, J. P. Campbell, and S. Ullrich, "Surveilling the gamers: Privacy impacts of the video game industry," *Entertainment Computing*, vol. 44, p. 100537, 2023.

[86] R. Phaenthong and S. Ngamsuriyaroj, "Analyzing security and privacy risks in android video game applications," in *International Conference on Advanced Information Networking and Applications*, pp. 307–319, Springer, 2024.

[87] A. C. Tally, Y. R. Kim, K. Boustani, and C. Nippert-Eng, "Protect and project: Names, privacy, and the boundary negotiations of online video game players," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW1, pp. 1–19, 2021.

[88] T. Tu, "Sony data breach possibly causing ps3 consoles to be banned." https://gamerant.com/sony-data-breach-ps3-ban/, 2021. [Online; accessed 04-March-2025].

[89] M. Chin, "Bethesda leaks personal data of fallout 76 players." https://www.tomsguide.com/us/fallout-76-breach,news-28781.html, 2018. [Online; accessed 04-March-2025].

[90] M. McShaffry and D. Graham, *Game coding complete*. Delmar Learning, 2012.

[91] S. Liu, A. Kuwahara, J. J. Scovell, and M. Claypool, "The effects of frame rate variation on game player quality of experience," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pp. 1–10, 2023.

[92] J. Cho and K. Ali, "Exploring quality assurance practices and tools for indie games," in *2023 IEEE/ACM 7th International Workshop on Games and Software Engineering (GAS)*, pp. 16–24, IEEE, 2023.

[93] M. Maybaum and K. Ziolkowski, "Technical methods, techniques, tools and effects of cyber operations," *Peacetime Regime for State Activities in Cyberspace*, pp. 103–134, 2013.

[94] L. Poretski and O. Arazy, "Placing value on community co-creations: A study of a video game'modding'community," in *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*, pp. 480–491, 2017.

[95] D. Charrieras and N. Ivanova, "Emergence in video game production: Video game engines as technical individuals," *Social Science Information*, vol. 55, no. 3, pp. 337–356, 2016.

[96] A. Vanègue, V. Bourcier, F. Petrillo, and S. Costiou, "Debugging video games: A systematic mapping," in *Proceedings of the 1st ACM International Workshop on Future Debugging Techniques*, pp. 23–30, 2023.

[97] Ars Technica, "Anti-cheat software causing big problems for windows 10 previews." https://arstechnica.com/gadgets/2019/02/anti-cheat-software-causing-big-problems-for-windows-10-previews/, 2019. [Online; accessed 13-February-2025].

[98] P. Paganini, "'bring your own vulnerable driver' attack technique is becoming popular among threat actors." https://cybernews.com/security/bring-your-own-vulnerable-driver-attack/, 2023. [Online; accessed 05-May-2025].

[99] SteamDB, "Steam games using anti-cheat expert technology." https://steamdb.info/tech/AntiCheat/AntiCheatExpert/, 2025. [Online; accessed 03-March-2025].

[100] Steam, "Garry's mod workshop." https://steamcommunity.com/app/4000/workshop/, 2025. [Online; accessed 03-March-2025].

[101] W. Scacchi, "Computer game mods, modders, modding, and the mod scene," *First Monday*, 2010.

[102] A. Kennedy, "Ghirbi, the gatekeeper." https://weatherfactory.biz/ghirbi-the-gatekeeper/, 2022. [Online; accessed 03-March-2025].

[103] H. Morse, "What is a pc game launcher?." https://www.ncesc.com/gaming-pedia/what-is-a-pc-game-launcher/, 2024. [Online; accessed 06-April-2025].

[104] S. Prescott, "The most popular desktop gaming clients, ranked." https://www.pcgamer.com/the-most-popular-desktop-gaming-clients-ranked/, 2019. [Online; accessed 06-April-2025].

[105] Valve, "Steam." https://store.steampowered.com/about/, 2025. [Online; accessed 06-April-2025].

[106] GOG, "Gog galaxy 2.0." https://www.gog.com/galaxy, 2025. [Online; accessed 06-April-2025].

[107] Epic Games, "Epic games store." https://store.epicgames.com/en-US/, 2025. [Online; accessed 06-April-2025].

[108] itch.io team, "About itch.io." https://itch.io/docs/general/about, 2025. [Online; accessed 06-April-2025].

[109] Ubisoft Entertainment, "Ubisoft connect." https://www.ubisoft.com/en-us/ubisoft-connect, 2025. [Online; accessed 06-April-2025].

[110] Valve Developer Community, "Steam browser protocol." https://developer.valvesoftware.com/wiki/Steam_browser_protocol, 2024. [Online; accessed 22-February-2025].

[111] SteamDB, "Steam game releases." https://steamdb.info/stats/releases/, 2024. [Online; accessed 22-February-2025].

[112] Steam, "Ea sports fc 25." https://store.steampowered.com/app/2669320/EA_SPORTS_FC_25/, 2024. [Online; accessed 05-May-2025].

[113] OpenCVE, "Vulnerabilities (steam client)." https://app.opencve.io/cve/?product=steam_client&vendor=valvesoftware, 2025. [Online; accessed 22-February-2025].

[114] Microsoft, "Opportunistic locks." https://learn.microsoft.com/en-us/windows/win32/fileio/opportunistic-locks, 2025. [Online; accessed 23-February-2025].

[115] Unity Technologies, "Unity engine." https://unity.com/products/unity-engine, 2025. [Online; accessed 08-April-2025].

[116] D. Ankit, "10 most popular games made in unity engine." https://300mind.studio/blog/top-games-made-in-unity-engine/, 2024. [Online; accessed 08-April-2025].

[117] K. T. Jensen, "25 years later: The history of unreal and an epic dynasty." https://www.pcmag.com/news/25-years-later-the-history-of-unreal-and-an-epic-dynasty, 2023. [Online; accessed 08-April-2025].

[118] Epic Games, "Unreal engine." https://www.unrealengine.com/en-US, 2025. [Online; accessed 08-April-2025].

[119] C. Enriquez, "Top 20 games made with unreal engine." https://vagon.io/blog/top-games-made-with-unreal-engine, 2025. [Online; accessed 08-April-2025].

[120] Valve Developer Community, "Source." https://developer.valvesoftware.com/wiki/Source, 2025. [Online; accessed 08-April-2025].

[121] A. Ermolaev, "A brief history of crytek studios: From the first far cry to the crysis 4 freeze." https://vgtimes.com/articles/120530-a-brief-history-of-crytek-studios-from-the-first-far-cry-to-the-crys html, 2025. [Online; accessed 08-April-2025].

[122] Crytek, "Cryengine features." https://www.cryengine.com/features, 2025. [Online; accessed 08-April-2025].

[123] SteamDB, "Steam games using unreal engine." https://steamdb.info/tech/Engine/Unreal/?max_release=2012-12-31, 2025. [Online; accessed 08-April-2025].

[124] Valve Developer Community, "Console command list." https://developer.valvesoftware.com/wiki/Console_Command_List, 2024. [Online; accessed 22-February-2025].

[125] Valve Developer Community, "Source rcon protocol." https://developer.valvesoftware.com/wiki/Source_RCON_Protocol, 2024. [Online; accessed 23-February-2025].

[126] Center for Internet Security, "Ransomware: The data exfiltration and double extortion trends." https://www.cisecurity.org/insights/blog/ransomware-the-data-exfiltration-and-double-extortion-trends, 2021. [Online; accessed 06-April-2025].

[127] K. Orland, "Bethesda warns of user data breach." https://www.gamedeveloper.com/business/bethesda-warns-of-user-data-breach, 2011. [Online; accessed 04-March-2025].

[128] S. Chng, H. Y. Lu, A. Kumar, and D. Yau, "Hacker types, motivations and strategies: A comprehensive framework," *Computers in Human Behavior Reports*, vol. 5, p. 100167, 2022.

[129] Microsoft, "About dynamic-link libraries." https://learn.microsoft.com/en-us/windows/win32/dlls/about-dynamic-link-libraries, 2025. [Online; accessed 22-January-2025].

[130] Microsoft, "Advantages of dynamic linking." https://learn.microsoft.com/en-us/windows/win32/dlls/advantages-of-dynamic-linking, 2025. [Online; accessed 22-January-2025].

[131] Microsoft, "Dynamic-link library search order." https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order, 2025. [Online; accessed 22-January-2025].

[132] E. Pena, R. Boonen, and B. Hawkins, "Dll side-loading and hijacking — using threat intelligence to weaponize r and d." https://cloud.google.com/blog/topics/threat-intelligence/abusing-dll-misconfigurations/, 2020. [Online; accessed 22-January-2025].

[133] MITRE, "Hijack execution flow: Dll search order hijacking." https://attack.mitre.org/techniques/T1574/001/, 2024. [Online; accessed 22-January-2025].