UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA
(FINAL PROJECT PAPER)

PRESLEPITEV SISTEMOV ZA ZAZNAVANJE GOLJUFANJA Z ZLORABO APLIKACIJ TRETJIH OSEB Z OMREŽNO USMERJENIM RAZVOJEM

(Exploiting Third-party Software To Bypass Anti-cheat Detection Systems Through Network-driven Development)

KLEMEN JANEZ POLIČAR

2022
ZAKLJUČNA NALOGA (FINAL PROJECT PAPER)
POLIČAR

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga
(Final project paper)

Preslepitev sistemov za zaznavanje goljufanja z zlorabo aplikacij tretjih oseb z omrežno usmerjenim razvojem

(Exploiting third-party software to bypass anti-cheat detection systems through network-driven development)

Ime in priimek: Klemen Janez Poličar
Študijski program: Računalništvo in informatika
Mentor: izr. prof. dr. Jernej Vičič
Somentor: asist. Aleksandar Tošić
Delovni mentor: mgr. Filip Kliber

Koper, avgust 2022

Ključna dokumentacijska informacija

Ime in PRIIMEK: Klemen Janez POLIČAR

Naslov zaključne naloge: Preslepitev sistemov za zaznavanje goljufanja z zlorabo aplikacij tretjih oseb z omrežno usmerjenim razvojem

Kraj: Koper

Leto: 2022

Število listov: 43 Število slik: 8

Število referenc: 42 Število tabel: 3

Mentor: izr. prof. dr. Jernej Vičič

Somentor: asist. Aleksandar Tošić

Delovni mentor: mgr. Filip Kliber

Ključne besede: Anticheat, Preslepitev

Izvleček:
Razvoj sistemov proti goljufanju je že od samega začetka igra mačke in miši. Ob razvoju programa za goljufanje se algoritem za detekcijo v sistemu proti goljufanju prilagodi. Program je zaznan kot zlonameren, njegovim uporabnikom je prepovedano nadaljnje igranje igre, nakar se program prilagodi — znova ga ni mogoče zaznati in proces se ponovi. V zaključni nalogi predstavljamo nov pristop za preslepitev obstoječih sistemov proti goljufanju, s katerim lahko bolje zavarujemo uporabnike programa pred zaznavo in posledicami goljufanja. V zaključni nalogi je opisana tako teoretična osnova nove metode kot tudi implementacija v računalniški igri Albion Online. Naš cilj je usmeriti industrijo k prilagoditvi sistemov proti goljufanju za preprečitev uporabe omenjene metode za goljufanje.

Keyword documentation

Name and SURNAME: Klemen Janez POLIČAR

Title of final project paper: Exploiting third-party software to bypass anti-cheat detection systems through network-driven development

Place: Koper

Year: 2022

Number of pages: 43 Number of figures: 8

Number of references: 42 Number of tables: 3

Mentor: Assoc. Prof. Jernej Vičič, PhD

Co-Mentor: Assist. Aleksandar Tošić

Practical mentor: Mgr. Filip Kliber

Keywords: Anticheat, Bypass

Abstract: The development of anti-cheat systems has always been a game of cat and mouse. Once cheating software is discovered by anti-cheat companies, they respond by adapting their systems to detect it. The program is flagged as malicious and its users are prohibited from playing the game, after which the cheat program adapts — it becomes undetectable once again and the process repeats itself. In our work we present a novel approach for bypassing existing anti-cheat systems, which promises better protection from detection and sanctioning. Our thesis describes the theoretical basis of the new method as well as its implementation in the popular video game Albion Online. Our goal is to direct the industry towards adapting anti-cheat systems to prevent the use of our bypass.

Acknowledgements

I would like to express my gratitude to my mentor and co-mentor for their guidance, expertise and positivity. I am grateful to my friends and family, my older brother in particular, for his suggestions and enthusiasm.

List of Contents

List of Tables

List of Figures

List of Abbreviations

AAA triple-A. 5

API application programming interface. 6, 7, 9, 12, 18

CEO chief executive officer. 5

CSV comma-separated values. 23

DLL dynamic-link library. 5, 6, 8

DNS domain name system. 5, 7, 18

EAC Easy Anti-Cheat. 5

ESP extrasensory perception. 7, 8

FPS first-person shooter. 3, 4, 13

GUI graphical user interface. 7

LAN local area network. 18

MMORPG massively multiplayer online role-playing game. 1, 2, 15

NPC non-player character. 16, 21, 22, 24, 29

opcode operation code. 16

PPTP point-to-point tunneling protocol. 18

PVP player versus player. 15, 23

RAM random-access memory. 18

SSD solid state drive. 18

UDP user datagram protocol. 16

VAC Valve Anti-Cheat. 4

vCPU virtual centralized processing unit. 18

VNC virtual network computing. 17, 20

VPN virtual private network. 13, 18, 29

VPS virtual private server. 18

1 Introduction

The purpose of this thesis is to present a novel technique for bypassing anti-cheat detection systems. Alongside the development of the now-massive gaming industry, there was an ever-increasing need for sophisticated anti-cheat systems. However, as the anti-cheat systems advanced, so did the cheats themselves. Bypassing an anti-cheat system nowadays is no easy feat due to its exceedingly steep learning curve. These systems are designed not only to prevent but to highly discourage any novice from entering this field. There are numerous protection techniques employed, all of which must be addressed in order to create a functioning cheat solution. Failure to do so will result in detection and sanctioning. Not only is developing an undetected cheat difficult, but it is also tedious to maintain, as the software must constantly be updated to work with newer game versions and anti-cheat system updates. Our technique applies approaches that decouple the cheat software from the game's inner workings, which in turn greatly reduces the likelihood of needing to patch the cheat once the game's publisher pushes an update.

The importance of sophisticated anti-cheat solutions is noticeably increasing as the gaming industry grows larger and larger. Video games are becoming ever more popular, which is reflected in the revenue growth of the sector [32]. An important metric for evaluating anti-cheat solutions is the wide variety of cheating techniques it can detect. Any successful online game that merits a broad player base must be paired with a strong anti-cheat solution if it is to remain popular [34].

Throughout this thesis, we will explain the motivation fueling the discovery of our technique. We will describe the theory of our technique, providing implementation examples along with a real-world application for Albion Online [8], a popular massively multiplayer online role-playing game (MMORPG). Our implementation incorporates the trojan horse concept [13], which is no novelty in the space of cybersecurity, however, it is integrated in such a way that has so far been unheard of in in-game cheating. We compare our technique to existing methods within the field, listing the benefits and drawbacks. Our concluding remarks describe what parts of the industry our thesis affects and how companies can adapt their anti-cheats.

2 Anti-cheat systems

Anti-cheats are software systems designed to prevent hacking and cheating in video games, building upon three important tenets: prevention, detection, and deterrence [39]. Today, the most renowned anti-cheat systems use methods, similar to most antivirus software methods, to scan for malicious behavior [2]. They are designed to work with multiple games using general protection techniques, although some have dynamic capabilities, tailored to specific games. Their sole purpose is to reduce the number of cheaters in video games, thus ensuring a fair experience for players.

2.1 History and motivation for development

Anti-cheat systems have been around since the inception of multiplayer video games around the year 1980 [2,15]. At first, most cheat developers were just tech-savvy players looking to abuse their practical skills in games. As the gaming industry evolved, a new industry emerged in parallel. Developers began to package and sell their cheats, which made cheating more accessible to casual players [5]. With the rise in popularity, there was a growing concern about the legality of distributing and making a profit off of such software. Many cheats modified the game code, which raised copyright legality issues. Companies began to claim the cheats should fall under derivative work copyright law, an argument that has not been universally accepted among jurisdictions [23]. Some countries such as South Korea have even gone so far as to pass an amendment into law with the specific intent of preventing cheating in video games [18].

There was a growing issue that players were being disincentivized from playing games that had been overrun by cheaters [4, 34]. Additionally, as more and more people started playing competitive online games, the problem with cheaters began affecting other aspects of the gaming industry. Playing games used to merely be a hobby, but in the past decade, gaming has opened up various careers, such as e-sports and streaming [6]. With the widespread adoption of gaming, along with financial incentives, separating cheaters from fair players had never been as vital for the health of the gaming industry [4, 22].

One of the oldest and most popular MMORPGs that has had two decades of problems with botting is *RuneScape*, developed by Jagex. At the time, there were no clear and proven methods of developing anti-cheat systems. Jagex's first large response to the influx of cheaters happened on December 10, 2007, when they announced the removal of free trade and heavily restricted wilderness gameplay, along with other game features that bots would abuse [29]. The changes were highly controversial, as

they greatly affected fair players, leading to strong backlash by the community. In spite of this, the changes proved to be very effective in greatly decreasing the prevalence of bots. The bots had adapted to these changes throughout the years, which caused Jagex to launch an update code-named *ClusterFlutterer*, also known as *Bot Nuking Day* [30]. This update detected reflection and injection bots, which led to over 1.5 million bans [30]. Finally, in September of 2012, Jagex launched *Botwatch*, a heuristic-based anti-cheat system that monitors user gameplay and detects non-human behavior [40]. The problem with relying so heavily on heuristics became disastrously apparent on September 27, 2012. Nearly 40,000 legitimate accounts had received an unappealable permanent ban within less than an hour by the anti-cheat system, due to a bug that had been pushed to production. This event had later been labeled *Botwatch rogue banning glitch* [40].

There are many more cases of games with a long history of cheat development. Quake is a notable first-person shooter (FPS) game that was one of the first to present the difficulty of fighting cheat developers in an open-source environment [27].

The traditional exploit-patch lifecycle for cheats in games is similar to that of malware and anti-virus software [2,14]. The development of the anti-cheat industry has largely been guided by this never-ending cycle. As new methods for cheat development emerged, anti-cheat companies responded and adapted their solutions to prevent such techniques [14]. The state of combating cheating today is more diverse and complex and varies from game to game, however, in principle, the cycle remains the same.

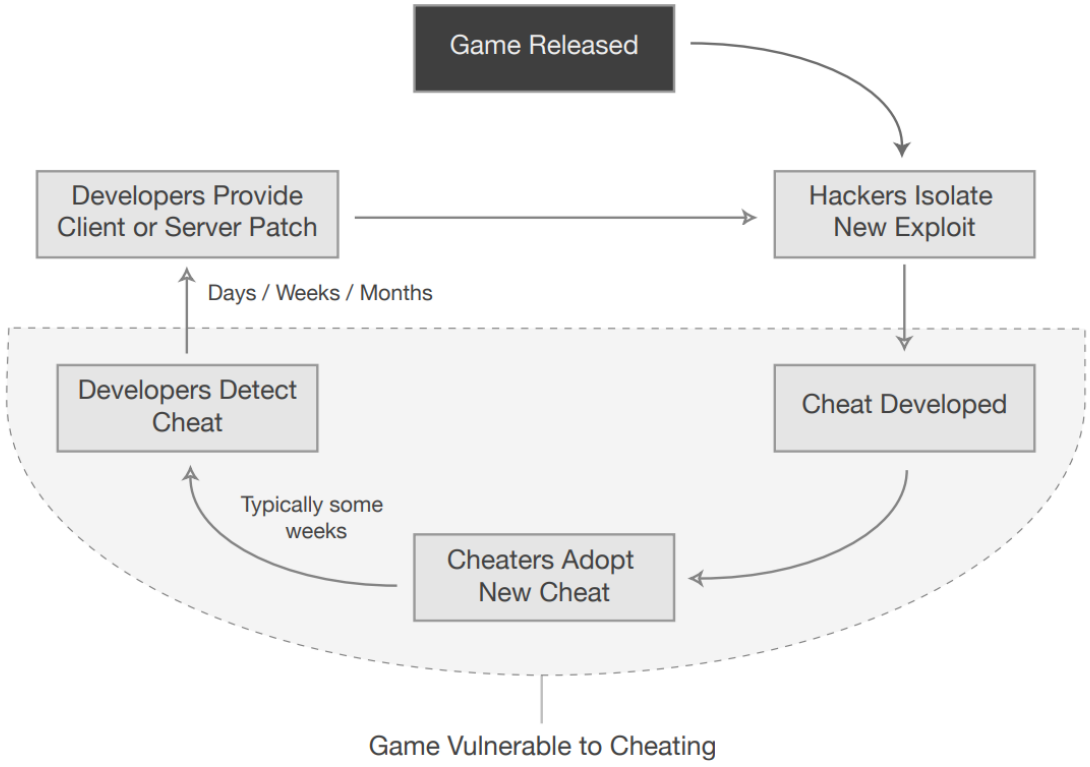

Figure 1: Traditional exploit-patch lifecycle of a game
Source: A Novel Approach to the Detection of Cheating in Multiplayer Online Games [14]

2.2 Existing solutions

Punkbuster is one of the first widely-adopted commercial anti-cheat systems to have had a lasting impact. It was developed by Tony Ray and first released in September 2000, motivated by the developer's negative experience in Team Fortress Classic [17]. It had been integrated into some of the most popular FPS franchises of the decade, such as the Call of Duty series, Battlefield, Quake, Far Cry, etc. Punkbuster works as an external process that must be running for players to join cheat-proof servers. Players could still play on servers that had disabled Punkbuster, though these are often overrun with cheaters and are thus avoided by fair players. Tony Ray has stated that cheaters regularly find ways around Punkbuster, but a quick software patch would mend the problem, that is until the next exploit, which is in line with the traditional lifecycle depicted in Figure 1 [17, 42].

Valve Anti-Cheat or *VAC* for short was developed by Valve in 2001, soon after the introduction of Punkbuster. They released the anti-cheat solution together with their massively popular game Counter Strike in 2002. VAC has matured extensively

throughout the years and it was one of the first anti-cheats to adopt domain name system (DNS) cache scanning, for which they were initially widely criticized, out of fear of what they could be doing with such information [20, 38]. In fact, Valve's trustworthiness has been put into question many times due to their aggressive approaches to cheat detection. Valve's chief executive officer (CEO) Gae Newell issued a statement in 2014 addressing the social engineering aspect, claiming that such criticism was being devised by cheat developers to discredit Valve's trust-based system [19].

nProtect Gameguard is considered to be one of the three anti-cheat solutions that dominate the online game security market [25]. It was developed by INCA Internet, a company that holds more than 70% of the market share of information security for Korean financial institutions and more than 90% of game portal security [41]. The software consists of both a user-mode and kernel-mode rootkit, proactively preventing cheat software from running [2]. Due to its intrusive nature, GameGuard has also been the target of widespread criticism.

Easy Anti-Cheat or *EAC* for short is a well-renowned kernel-mode anti-cheat, having reached 275,000,000 monthly active users by 2019 [21]. It was developed by Kamu in Helsinki and was acquired by Epic Games in 2018, which thereinafter led to its integration alongside many high-budget high-profile games, often referred to as triple-A (AAA) titles, such as Apex Legends, Fortnite, Rust, and many more [12]. EAC represents the next generation of anti-cheats, as it runs largely in the cloud, utilizing big data to detect cheaters.

2.3 Architecture of a modern anti-cheat system

The architecture of anti-cheat systems varies from company to company, however, at the current state most top-end anti-cheats use a standard architecture, consisting of four main components: the master server, a kernel driver, an external process, and an internal dynamic-link library (DLL), all of which are interconnected [21].

The anti-cheat server runs on the internet. When a user opens the game executable, the anti-cheat service establishes a connection to the server. If this connection is terminated for whatever reason, the service of the anti-cheat is flagged as no longer fully operational, which suspends the game session immediately [21]. This prevents users from playing the game without the anti-cheat running. The server transmits challenges to the client to discover whether the game client is engaged in unauthorized behavior [1]. Additionally, it receives alerts that are produced whenever something irregular happens on the host computer. The anti-cheat service will dump the suspicious process or DLL and send it back to the server for analysis [21].

Another component running on the host computer is the anti-cheat driver. Run-

ning in ring 0, the driver is guaranteed full memory and hardware access [15], providing powerful capabilities which may otherwise not be possible to implement for an exclusively user-level anti-cheat.

The internal component of an anti-cheat is a DLL that is loaded directly into the game's process. This DLL is responsible for analyzing the memory of the game, constantly monitoring memory regions to discover if any new executable region has been allocated or if a thread has executed code from an unexpected region of memory and more [21].

Finally, the external process running in ring 3 provides an overview of the system. It scans all the running processes, prohibiting the use of blacklisted programs [15]. It manages the logic of the driver and the connection with the master servers [21].

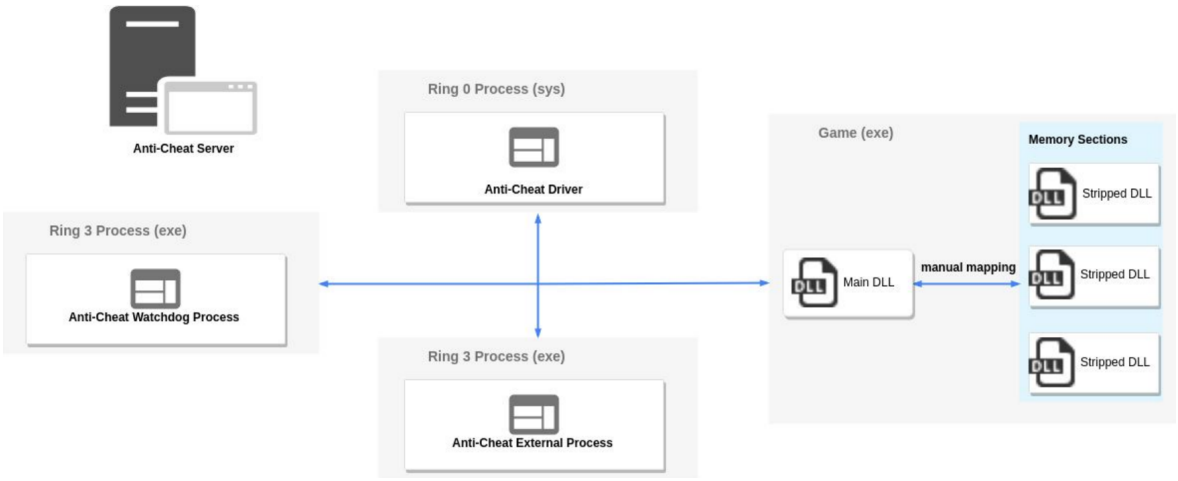

Figure 2: Architecture of a modern-age anti-cheat system
Source: Unveiling the Underground World of Anti-Cheats, Blackhat Europe 2019 [21]

2.4 Traditional approach for bypassing anti-cheat systems

To correctly bypass an anti-cheat system, it must first be considered what sort of cheat is appropriate. Cheat software is separated into two main categories: *external* and *internal* [26]. External cheats are independent programs running in their own process, often relying on indirect means to gather information and interact with the game. These cheats may rely on the use of Microsoft Windows application programming interface (API) calls to manipulate the game's memory and simulate inputs. In order to discover information about the game's runtime, the cheat will require a valid handle

to the game process. Some of the protection techniques which target external cheats are the following [21, 33]:

- **Detect suspicious handles** — The anti-cheat system monitors any valid handles to the game process and verifies which program has created the handle. If the program is unsigned or has been signed by an unreputable source, it may be deemed suspicious and in need of further analysis [21].
- **Monitor Windows API calls** — The anti-cheat system hooks various Windows API functions, such as memory manipulation calls, redirecting their execution to the anti-cheat, so as to analyze the behavior of programs that are calling such functions [21].
- **Detect unsigned executables** — The anti-cheat system monitors all running applications and, if it detects that an unsigned executable is running, it may monitor the behavior of the program more closely [21].
- **Detect emulated inputs** — The anti-cheat system detects input commands that have been injected through non-physical media [3, 33].
- **Screenshot detection** — The anti-cheat system takes a screenshot of the player's desktop, which reveals extrasensory perception (ESP) cheating from an external graphical user interface (GUI) [2, 14].

In general, an external cheat is more or less safe to use and will not result in an immediate ban until it has been marked as malicious and put on a blacklist by the anti-cheat company. Once this has happened, the cheat will face aggressive protection techniques designed to detect whether or not the software is running. The items listed below are just some of the known detection techniques cheat developers must face once their software has been flagged as malicious [2]:

- **Signature-based detection** — The malicious software's unique byte patterns are stored and actively scanned for by the anti-cheat software [2, 15].
- **DNS cache scans** — The player's DNS cache is scanned for domain names associated with cheating. Uncovering that the player has visited such a website increases the probability that the player may be cheating [2, 15, 20].

On the other end of the spectrum, there are internal cheats. In contrast to their counterpart, these operate within the memory space of the game, making them much more difficult to detect when developed properly. The items listed below are some of the most common features anti-cheats provide to tackle internal cheats. They are used

to determine whether or not a process is behaving in a suspicious manner and, if they deem so, they may ban the player immediately or send over information about the process's runtime to their server for further analysis [2, 21].

- **Game executable hash validation** — The anti-cheat system creates a cryptographic hash of the player's game executable binary and compares it to the hash stored in the central server. If the hashes do not match, the player's executable has been modified [2, 15].
- **Prevent DLL injection** — The anti-cheat system closely monitors all DLL files that are loaded by the game and prevents any malicious code from being loaded [21].
- **Detect thread execution with suspicious context** — The anti-cheat system analyzes each region of the game's memory and monitors executable regions, detecting thread execution within an unexpected region of memory [21].
- **Binary validation challenges** — The anti-cheat system continuously performs binary validation challenges, which compare hashes of in-memory binary code with the same code in the binaries on the filesystem. A mismatch indicates there has been memory tampering [2].
- **Control of access flags** — The anti-cheat system constrains tampering with various regions of memory with restrictive access flags [21].
- **Prevent thread injection and hijacking** — The anti-cheat system detects and prevents malicious attempts at creating new threads in the game's process or hijacking existing threads [2].
- **Screenshot detection** — The anti-cheat system takes a screenshot of the player's game, which reveals ESP or modified game visuals from the game's internal graphics engine.

Anti-cheat systems that also consist of a kernel driver include an even broader overview of the system and control over applications running alongside the game. Some of the protection techniques are the following [21]:

- **Register kernel callbacks** — The anti-cheat system hooks into kernel API calls to monitor and handle low-level events on the system [21].
- **Rejection of kernel/user mode debugging** — The anti-cheat system prevents the player from running the game if the operating system is running in debug mode [21].

- **Block blacklisted/unsigned drivers** — The anti-cheat system prevents certain drivers from running, to prevent cheaters from entering kernel space. Such drivers are either vulnerable to exploitation or are unsigned [21].

- **Detection of virtual environments** — The anti-cheat system detects virtual environments and prevents the game from running. The reasoning for this feature is that anti-cheat systems cannot detect cheating software outside the virtual environment [21].

In addition to defeating all these protective measures, there are considerations that must be made during the development of the cheat software. Numerous protection techniques must also be developed for the cheat itself, to prevent anti-cheat developers from targeting the cheat specifically, as well as to discourage others from stealing the proprietary technology [2].

- **Anti-debugging techniques** — The cheat program's control flow is modified upon detection of debugging software to prevent analysis of the program's behavior [2].

- **Minimizing the footprint** — The cheat program uses generic or commonly-used programming patterns, such as hooking into the Windows API instead of directly hooking into the game's code, to make itself less distinguishable [2].

- **Masking the footprint** — The cheat program is obfuscated to prevent analysis of the binary file. A common method of obfuscation is packing, which hides the executable within another executable that decrypts it on runtime [2].

3 Methodology for bypassing anti-cheat systems

Understanding the vast amount of protective features employed by anti-cheat services, as described in Chapter 2.4, there was a clear incentive to approach cheat development in a different manner. Circumventing each of the techniques separately becomes a very difficult and tedious task.

Our approach proposes a separation of the cheat program from the host computer running the anti-cheat software. Taking into account the fact that anti-cheats work by continuously scanning the host computer, as described in Chapter 2, moving the cheat software out of bounds of the anti-cheat voids currently known protection techniques.

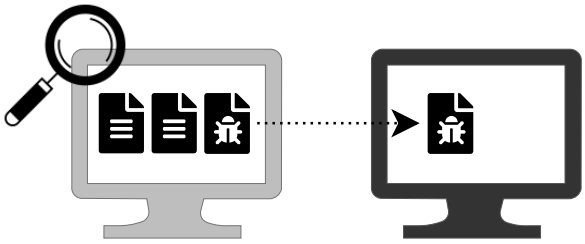

Figure 3: Moving the cheat software out of bounds of the anti-cheat

3.1 Collecting information about the game

The first challenge that arises from the idea of separation is: how will the cheat obtain information about the game? We propose four different solutions:

1. Reroute network traffic to the cheat server and parse the game packets (assuming the traffic is not encrypted)
2. Send a video stream of the game to the cheat server and gather information using computer vision
3. Send an audio stream of the game to the cheat server and gather information through audio analysis
4. A hybrid approach

By intercepting the network traffic, we are able to uncover valuable information that the game client sends to the server. Any interaction our player makes that is visible to

other players within the game and vice versa is information that is undoubtedly being sent over the network and can thus be parsed.

On the other hand, sending a video stream would allow us to parse information from the game using computer vision techniques. We are given all the information that any regular player is given, which in theory means we should be able to replicate any fair player behavior. Depending on our use case, we may be limited in terms of response time. Due to this limitation, this technique may not be applicable to fast-paced games.

Parsing information from an audio stream could also be a viable solution, be it very limited in terms of the features the cheat could offer. The complexity of the analysis that is performed depends on the use case; a trivial program could simply scan whether or not the volume level exceeds a certain threshold.

The hybrid approach is comprised of a simple program, which is used to read the game's memory, and this is then sent to the cheat server. The pros of using this approach are, that we are guaranteed to have a smaller footprint than if the program contained the entire business logic, that we have access to all the game's information we might otherwise be stripped of and that we protect the cheat from being reverse-engineered by other bot developers. The largest con of this approach is that the program is once again detectable, resulting in potential bans for its users.

Table 1: Comparison of information gathering techniques

	Packet sniffer	Computer vision	Audio analysis	Memory scanner
Computationally intensive	low	high	neutral	low
Information retrieval capability	neutral	high	low	high
Vulnerable to deception	yes	yes	yes	no
Stability across updates	neutral	high	neutral	low
Complexity	low	high	neutral	low

3.2 Sending commands to the game

The second obstacle to overcome is: how will the cheat send input to the game? We propose three different solutions:

1. Sending spoofed network packets to the game server

2. Sending input commands to the game through a third-party application

3. A hybrid approach

Sending spoofed network packets to the game server will allow us to execute very precise commands that may otherwise be difficult to emulate had we restricted ourselves to input commands. By choosing this technique we open ourselves to the risk of detection since games often have protective features in place to uncover spoofed packets. This approach is also more complex due to the fact that we need to reverse-engineer the game's protocol in order to inject our own packets.

Sending input commands is a very straightforward approach to automating in-game actions, as we just simulate ordinary player mouse movements and keystrokes. In order to remain stealthy, we exploit a third-party application that runs on the host computer. By communicating with this application, we instruct it to send input commands to the game client. The third-party application thus behaves as a trojan horse for the cheat software. The criteria for selecting an application is that the program must be controllable over a network and is able to send commands to the game.

The hybrid approach proposes a similar concept to what is described in Chapter 3.1. We are able to send commands using a simple program running on the host machine. The program can execute commands using any of the existing techniques, such as simulating input commands, be it using the Microsoft Windows API or other input data stream injection techniques, or by modifying the game's memory, etc.

3.3 Limitations

Excluding the hybrid approach, our technique provides much more limited cheating potential than traditional approaches. If we use any of the options mentioned in chapters 3.1 and 3.2, we are stripped of the ability of reading and modifying the game's memory. As a result, it is of our opinion that the use of our technique is more appropriate for botting software, which emulates real player behavior, in contrast to giving players an upper hand with additional information about the game or modifying the game's behavior.

Since we are limited in the methods with which we can gather information about the game's state, there may be information that is impossible to discover using our technique, which may not be the case with traditional approaches.

3.4 Comparison with traditional approaches

The use of traditional methods for bypassing anti-cheats is generally quite risky for the end-user. Any bypass that is developed will eventually be patched by anti-cheat companies once discovered. When this happens, any user caught using the bypass will be sanctioned as per the game company's directive, most often with an immediate permanent hardware ban [2,10]. This can happen because the cheat's usage has become detectable, whereas the use of our technique is merely preventable, as is further outlined in Chapter 3.5.

Our technique inherently evades most protective features current anti-cheats have in their arsenal. Furthermore, cheat developers no longer need to worry about protecting their software from reverse engineering. Taking the traditional approach, each protective measure for the specific anti-cheat system would need to be counteracted individually. This is in contrast with our method, which greatly reduces the barrier to entry for developers.

A large drawback that must be mentioned is that due to network latency and largely increased computational intensiveness compared to traditional approaches, our technique is unsuitable in applications that demand quick response time, such as FPS and other fast-paced games.

3.5 Potential solution for preventing our technique

The use of our technique largely invalidates cheat detection and player sanctioning as a viable anti-cheat solution. When implemented properly, it is not possible to differentiate fair players from cheaters. This is because fair players may also be using third-party software for their own non-exploitative needs.

We propose anti-cheat companies focus their attention on prevention in regard to our technique, rather than attempting to detect cheaters outright. The difference between prevention and detection is subtle. Detection is defined as successfully discovering an illegal action, whereas prevention inhibits future cheating using a specific technique [11, 28]. There are a couple of potential solutions that could prevent the usage of our technique:

- Prohibit usage of well-known third-party applications that are able to inject input commands while the game is running (eg. remote desktop applications)
- Prohibit packet redirection (eg. disallow virtual private network (VPN) connections) while the game is running

The proposed solutions could be implemented similar to virtual machine detection — the anti-cheat detects that the game is running in a virtual environment and prevents gameplay. Although the two solutions could prevent cheaters from using our technique, they come with a potentially large drawback: along with cheaters, also fair players would be affected as well, hence they would need to be carefully assessed.

Both of the proposed solutions can also be patched by the cheat developer. The cheat developer may switch to a different third-party application and could avoid packet redirection prevention, for example by duplicating network traffic, possibly on the router (therefore out of bounds of the anti-cheat), by sending a copy of the packets to the cheat server.

Apart from direct approaches, heuristics analysis could be a viable detection method. By processing the player's behavior the anti-cheat should be able to determine the probability of whether or not the player is behaving in a non-human fashion [2]. The required complexity of heuristic analysis solutions depends on the level of sophistication of the cheat software, meaning its effectiveness will vary among use-cases.

4 Implementation in Albion Online

Albion Online is a free medieval fantasy MMORPG developed by Sandbox Interactive, a studio based in Berlin, Germany. Albion features a player-driven economy, classless combat system, and intense player versus player (PVP) battles. The game runs on all major platforms: Microsoft Windows, Linux, macOS, Android, and iOS [8]. In the year 2021, the game ranked 23rd among the most played games reaching over 140,000 daily active users [24]. Albion Online is protected by Easy Anti-Cheat, mentioned in Section 2.2, which is to say it is the anti-cheat system we bypassed in our thesis.

4.1 Automate currency generation

The purpose of the cheat we have designed is to behave as a *gold-farming bot*, automatically producing in-game currency. There are many ways of making an income in Albion Online. The activity we will be focusing on are trade missions.

Trade missions are an activity available through faction warfare that involves purchasing city heart fragment resources in a faction city and transporting them to a smuggling post near another city. After successfully completing a trade mission our character is awarded city heart fragments, which can be sold on the global exchange for a profit [9]. Our bot will be running trade mission quests in succession. Players may choose between three different options when starting a trade mission, depending on their attitude towards risk:

1. **Minor contract** requires 3 heart fragments and returns 4 as a reward
2. **Medium contract** requires 7 heart fragments and returns 9 as a reward
3. **Major contract** requires 15 heart fragments and returns 18 as a reward

In the case that the player's character dies during the trade mission, the quest is canceled, which incurs losses. The losses, as well as the rewards, are gradually larger, depending on the chosen option.

4.2 Using packet sniffing to collect information about the game

For gathering information about the game's state we chose option 1 from Chapter 3.1. Albion Online uses Photon Realtime, a renown and well-documented software service for their networking solution [7]. Due to its popularity, there are publicly available

open-source libraries that handle decoding network packets. This greatly lowers the barrier to entry for developing a network-based cheat for Albion Online. Because of the vast amount of real-time data that is persistently sent between clients in Albion, most of the network traffic is not encrypted, including information about player movement and interaction with non-player characters (NPC).

For our implementation, we utilized libpcap [35], a system-independent interface for user-level packet capture, which is well-known for its integration in Wireshark, a widely-used network protocol analyzer [36]. We filter packets by their source and destination port, according to Albion's networking protocol.

Once the appropriate packet arrives, the first thing we do is extract the payload. Albion's real-time data is transported using the user datagram protocol (UDP). Next, we parse the packet type. Packets are separated into two categories:

Request packet This packet is sent from our client to Albion's server, most often containing but not limited to information about our character's activity

Event packet This packet is sent to our client from Albion's server, most often containing but not limited to information about other players' activity

After working out which packet type we are dealing with, we parse the packet operation code (opcode). This opcode will determine what sort of content we can expect to parse in the next step and which handler should be initiated. Assuming no unexpected errors have occurred during parsing, we finalize the procedure by passing the packet contents to the appropriate handler.

Handlers are registered for specific opcodes. For instance, the move request operation handler is designated for updating the local character position within our data store.

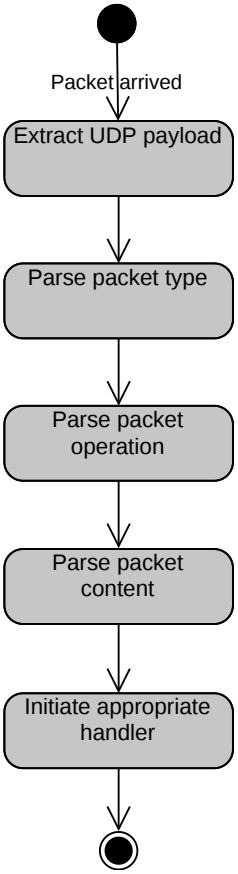

Figure 4: Packet parsing steps

4.3 Sending input commands to the game through a VNC connection

In our implementation we chose option 2 from Chapter 3.2, utilizing a virtual network computing (VNC) connection to send input commands to the client running Albion Online. The cheat software communicates with TightVNC, our third-party application of choice, instructing it where to move the mouse and which keystrokes to perform at any given time [16].

Before we could start the TightVNC client on our cheat server, we needed to install a graphical desktop environment. For our implementation, we opted for xfce4, a free and open-source desktop environment for Linux [37]. In order to send input commands to the TightVNC client, we used xdotool, an open-source program for simulating keyboard input and mouse activity on unix-based systems [31].

4.4 Deployment architecture

To achieve the separation described in Chapter 3, we needed to design our deployment architecture accordingly. There were multiple options that were taken into consideration for what physical media to run the cheat software on:

1. On the host computer in a virtual machine
2. On a separate computer in the same local area network (LAN)
3. On a machine on the internet

Option 1 comes with the drawback that it could be possible for the cheat to be detected since it is not physically out of bounds of the anti-cheat software. We chose option 3, as it could scale effortlessly if implemented in a commercial product. Utilizing the massive cloud computing industry, developers are able to deploy arbitrarily many bot instances on-demand. Since all the traffic is routed through a server on the cloud, each instance is inherently assigned a unique IP address, preventing DNS cache scan detection.

Although deploying a new server for each instance would incur financial costs, low-end servers have become very affordable. Furthermore, the ownership and server billing can be passed on to the bot's user. The user could supply the cheat provider with an API access token to his server provider, effectively granting him full control.

In our implementation, we rented the cheapest server available from Digital Ocean Inc., a popular cloud service provider. Our virtual private server (VPS) has the following features: 1GB of random-access memory (RAM), 25GB Regular Intel solid state drive (SSD) storage and 1 virtual centralized processing unit (vCPU) with a clock speed of 1.8GHz, running Ubuntu 20.04 x64.

Because our software is no longer running on a machine connected to the same network as the host computer running Albion Online, we are no longer able to observe game packets. To fix this, we established a VPN connection from the host computer to the cheat server, rerouting all network traffic, including traffic between the Albion client and server.

The host computer is running Albion Online, along with Easy Anti-Cheat and a TightVNC server. The machine will connect to the cheat server through the Microsoft Windows Built-in VPN provider.

The cheat server is running the cheat program, a point-to-point tunneling protocol (PPTP) VPN server, and a TightVNC client, which will connect to the host computer.

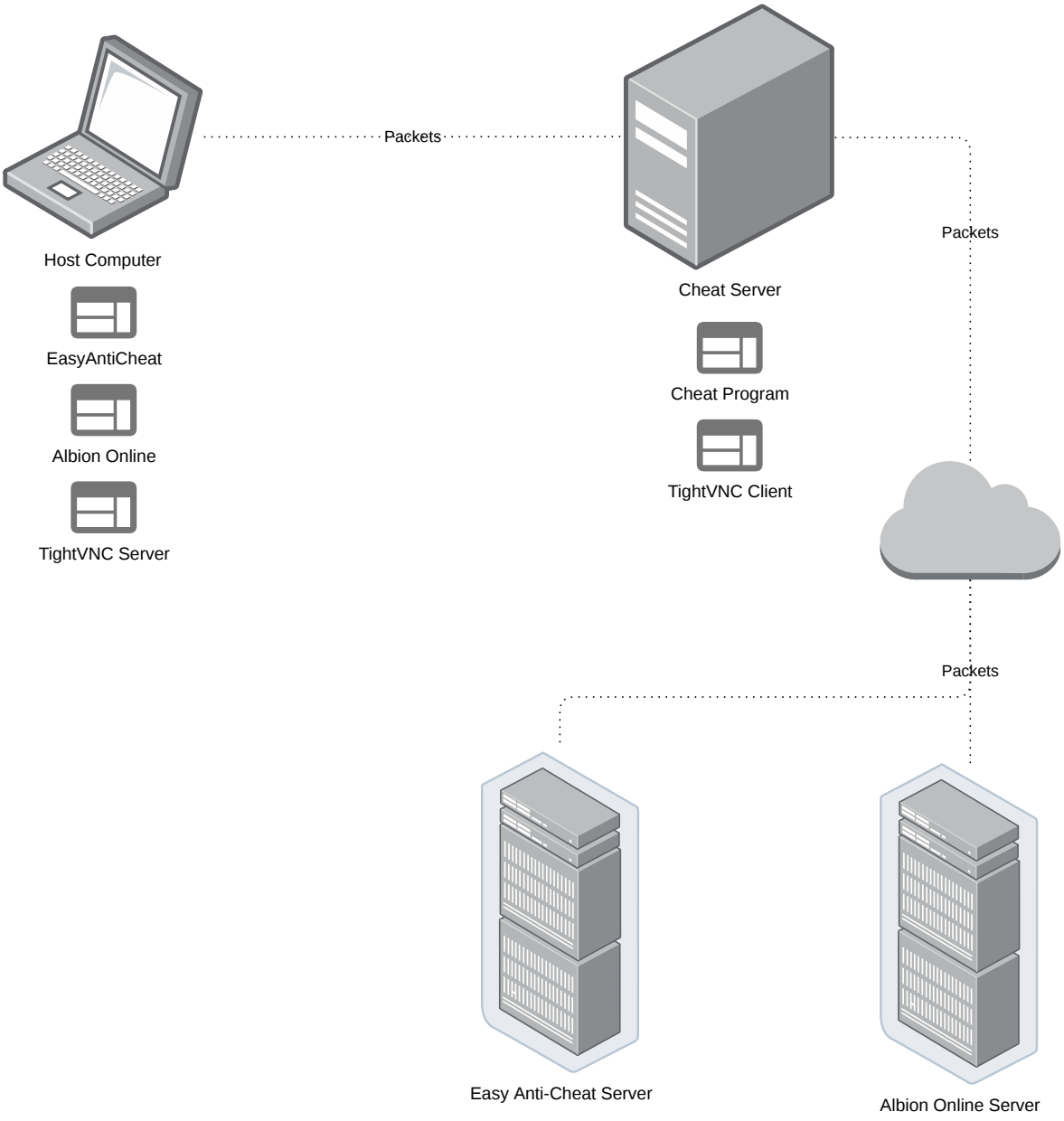

Figure 5: Cheat implementation deployment architecture

4.5 Automating in-game actions

4.5.1 Movement

By parsing packets representing our character's movement within the game, we are able to determine the precise location of our character. Given an array of waypoints containing in-game coordinates, we are able to construct a route that can be taken by the cheat program.

Because we have chosen VNC to send commands to the game, we are taking control of the cursor. This means we have to find out where to move the mouse in order to move our character in the desired direction. In Albion Online, moving the character requires the player to position his cursor in the direction towards which he wishes the character to move and then hold the right mouse button. There are also other ways of moving the character, however, due to its convenience, we have chosen to stick with this method. Using the simple procedure written in Algorithm 1, which takes the player's current position and a waypoint as its arguments, we calculate the direction our player should move towards to reach the waypoint. After calculating the direction, we simply offset the cursor position relative to the center of the screen, because the local character in Albion Online is always positioned in the center, regardless of its in-game position. The coordinate system in Albion Online is slightly rotated compared to the screen coordinate system, which is visualized as a tilted grid in Figure 6. To adjust for this, the direction vector must be rotated by $-\frac{\pi}{4}$. The pseudocode for positioning the mouse cursor towards a waypoint can be seen in Algorithm 1.

Algorithm 1 Positioning the mouse cursor to move the character towards an in-game coordinate

procedure MoveCursorTowardsGameCoordinate($character, waypoint$)
$offsetFromCenter \leftarrow$ 10% of screen height ▷ assuming height $<$ width

$direction.X \leftarrow waypoint.X - character.X$
$direction.Y \leftarrow waypoint.Y - character.Y$
$direction \leftarrow$ rotate direction by $-\frac{\pi}{4}$
$direction \leftarrow$ normalize direction into interval [0,1]
$offset \leftarrow direction * offsetFromCenter$

$cursorLocation \leftarrow center + offset$
end procedure

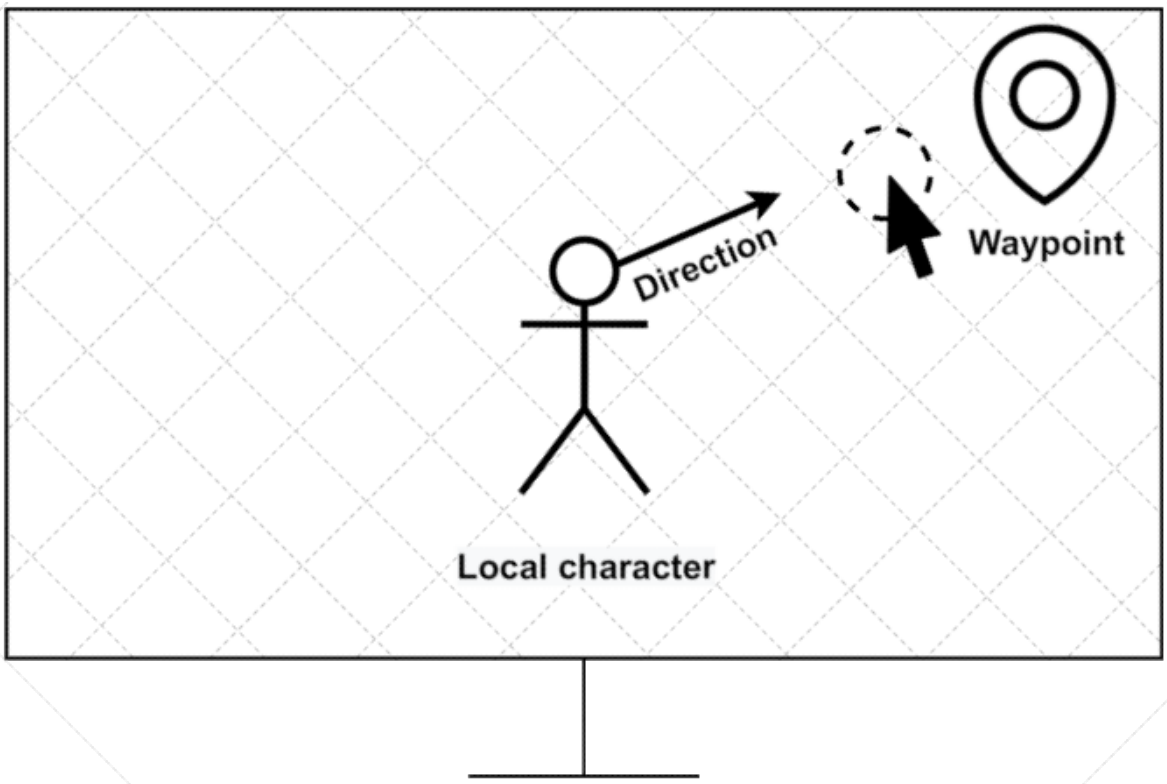

Figure 6: Moving the local character towards a waypoint

4.5.2 Interacting with NPCs

Interaction with NPCs in Albion Online can be initiated by clicking on them with the mouse cursor. Pinpointing their exact location was slightly more difficult than extracting our local character's position since they do not move within the world — they are statically positioned, which means there is no data about their position within the network stream.

To determine their location, we helped ourselves with the position of our character. During development, we moved our character on top of the NPC, with which we wanted to interact with during the bot's operation, and marked its coordinates. When the bot is running and the character reaches a minimum distance threshold to the NPC, the bot transitions from a movement state to interaction.

Once the interaction has been initiated, the exact behavior depends on which NPC we are interacting with and what we would like to achieve. For instance, to begin a trade mission, the bot initiates contact with the faction NPC, which opens up a dialogue interface. Within this user interface, the bot must open the tab which lists available trade missions, select the appropriate option and confirm the quest. Because every NPC interaction is unique, each must be programmed individually.

4.5.3 Running trade missions

Each trade mission quest run can be represented as a finite state machine consisting of a combination of predefined and dynamically recorded steps. The condition to reach a certain step is to have successfully completed the preceding step. The steps that are taken can be described with eight basic sequential states:

Run to the bank This is the starting state. The character will run from the quest NPC to the bank.

Bank items The character will interact with the bank NPC, depositing any reward items and withdrawing the items required to begin the trade mission quest.

Run to quest The character will run from the bank to the quest NPC.

Start quest The character interacts with the faction leader, starting the trade mission quest.

Run to smuggler The character will run from the faction leader to the distant smuggler.

Progress quest The character will interact with the smuggler, progressing the trade mission quest.

Run to the city The character will run from the smuggler back to the faction city from whence he came.

Finish quest The character interacts with the faction leader, finishing the trade mission quest.

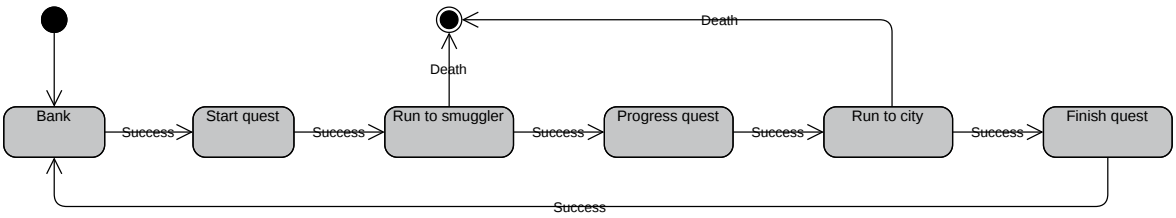

Figure 7: Trade mission bot steps presented as a state machine

The predefined steps include banking, beginning the quest, and progressing the quest, each of which consists of movement steps and/or interactions. The reason these steps are predefined and cannot be overridden is that they are constant and necessary for any such trade mission. It makes sense that the routes are represented as dynamic

steps, recorded by the bot user because these steps determine the majority of the quest's execution. Additionally, they are points of failure, because the character is exposed to PVP combat, meaning they can be scored for efficiency and safety. The transition between steps is executed when satisfying the condition for the step.

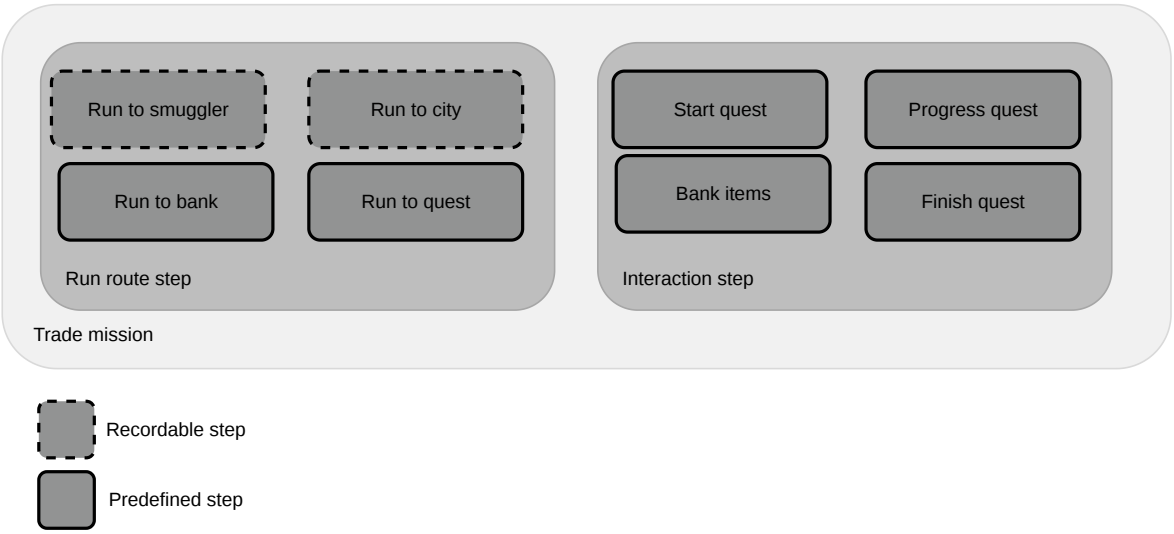

Figure 8: Hierarchy of trade mission steps

By implementing these various steps we are able to fully automate trade missions. After each successfully completed trade mission, our character is issued a reward. We are able to run these missions indefinitely, which in theory proposes an infinite source of income. In practice, because we have not handled hostile player encounters, the bot may eventually be killed and would need to be manually restarted. Even if we would implement combat or fleeing behavior, it would be difficult to guarantee a successful bot in such encounters. Another feature would be to automatically reset the bot in the case of the character's death.

4.5.4 Recording routes

There were two options that have been decided upon regarding running trade missions: whether or not the route the bot takes should be fixed or whether the user can record his own custom routes. We have developed our bot to support the latter, providing it with a route configuration file as an argument. This file is in comma-separated values (CSV) format, with each line containing the subsequent step that should be taken by the bot, along with its arguments, as displayed in Table 2.

Table 2: List of valid steps in routes

Name	Arguments		
Metadata	origin	destination	name
Move	x coordinate	y coordinate	
Cluster	identifier	alias	
Quest			

Each step is handled according to its function within the bot software. The step transitions are not strictly adhered to, in that the step may be skipped if during the bot's operation it is determined that the bot has already satisfied an upcoming step's transition condition. More concretely, if the character's position has updated to have reached an upcoming waypoint, the bot skips the preceding waypoints. Each registered step serves a specific purpose:

Metadata This entry is used to describe the route file. Therefore, a transition to the next step is done immediately thereafter.

Move This step behaves as a waypoint towards which the character should move. The waypoint is reached once the character is within a fixed distance threshold from the waypoint's coordinates.

Cluster This step marks the transition from one in-game cluster to another. Once a cluster change packet has been parsed off the network stream, this step's transition condition is satisfied.

Quest This step marks the arrival of the character to the smuggler NPC, initiating the progress quest step and skipping to the next route step, which indicates the beginning of the route back to the faction city.

5 Results

For our thesis, we have successfully completed 303 sample runs. Each run was done on a single route, from Fort Sterling to Aspenwood, Lymhurst. The duration of the trade missions has a standard deviation of 20 seconds and the median of a successful run is 17.8 minutes. With our current estimate, we can expect to run 3.3 trade missions per hour with an 8.6% chance of the character dying while running the mission.

Our results are the following. Let $rate = 3.3$ denote the number of trade mission runs per hour, let $error = 0.09$ be the rate of error of our bot, which in our case means death of our character and consequently failure of the trade mission, and let $cost = 52,000$ be the cost of a single heart fragment, which is set to the approximate 4-week average price of a heart fragment at the time of writing. To calculate the expected profit for each contract we denote fee as the number of hearts that must be paid for starting a trade mission and let $reward$ denote the number of hearts that are rewarded for successfully completing a trade mission. The values of these two parameters for each contract are mentioned in Chapter 4.1. We calculate $profit = rate * reward * cost$ as the gross profit and $loss = rate * error * fee * cost$ as the loss due to death. The expected net profit is calculated by subtracting the loss from the profit. To calculate the expected net profit of minor contracts, we calculate $profit = 3.3 * 1 * 52,000 = 171,600$ and $loss = 3.3 * 0.09 * 3 * 52,000 = 46,332$. The net profit thus amounts to $125,268$. Similar calculations are done for the two other contracts. A comparison of profits for each contract can be seen in Table 3.

Table 3: Expected profit calculations per hour for each contract

Contract	Gross profit	Losses	Net profit
Minor	171,600	46,332	125,268
Medium	343,200	108,108	235,092
Major	514,800	231,660	283,140

We observe that in our example, it is most profitable to run major trade missions. The profitability of minor and medium trade missions varies depending on the losses we incur due to death, which means it may make sense to alternate between these contracts depending on how likely we are to fail the trade mission. We have not analyzed the variables that may affect this probability, but our assumption is that, among other variables, the time of day has a high correlation with the rate of death. We presume that during peak hours, when the player count is high in Albion Online, we can expect our bot to fail more often. This assumption may be further researched in our future work. Our analysis is very limited due to the fact that we had sampled data for just

a single route. Our prediction is that cheaters record their own custom routes, each of which could have varying success rates. A lower rate of error could result in much larger net profits and vice versa.

6 Conclusion

This thesis has put forward a novel approach to developing undetectable cheats for games protected by modern anti-cheat systems. We have compared our approach to others in the industry and have successfully upheld the consideration of safety for cheating. Pulling apart the concepts of detectability and preventability, we have reached a formidable standard by promising complete undetectability and thus safety from sanctions. More importantly, we have proposed a viable solution for anti-cheat companies to adapt their systems to prevent the use of our technique. Our thesis proves there are many imaginative ways to get around anti-cheat systems.

Based on the numerous protective techniques employed by anti-cheats described in this thesis, it is clear our technique is a great choice for novice developers that lack the knowledge required to bypass anti-cheat systems in more traditional ways. Little to no understanding of the inner workings of these systems is required to implement our bypass. Due to the apparent limitations, applications could rarely be adapted to fit our technique and it would most often make more sense to rebuild the cheat software from scratch.

Our technique could be further researched to determine the viability of individual cheating methods. In our thesis, we presented the limitations that arise due to the increased latency and in-access to the game's memory — issues that may very well be surmountable. We have merely presented a practical application of our technique as a game bot, but there are various different cheating options players could exploit to be given an unfair advantage. Our thesis opens the door for developers to attempt to incorporate our technique in their cheats.

There is noticeable room for improvement within our implementation. Our bot works well so long as it does not encounter hostile players. To provide full autonomy we would need to implement several additional features to better react to certain incidences. To protect against heuristics detection, it would also be beneficial to adjust the bot's movement patterns by adding a degree of randomness to its behavior. The results from our data suggest that our bot has the potential to be very profitable, especially if we reduce the losses incurred due to quest failure.

As anti-cheat companies adapt their systems to prevent our technique, it must be further researched how to adapt our approach to once again bypass their inhibitory solutions. As has always been the case in this industry, our technique still falls within the cat-and-mouse depiction, with the need for consistent patching.

7 Povzetek v slovenskem jeziku

V današnjih časih je za industrijo računalniških iger vedno bolj pomemben razvoj sistemov proti goljufanju. V zadnjem desetletju je prišlo do bogatega razcveta industrije kar je odrplo več novih kariernih poti. V tovrstnih službah je znaten problem goljufanja, saj lahko prevaranti lažno pridejo do zaslužka. Poleg tega je goljufanje precejšnja težava za podjetja, ki prodajajo računalniške igre. Študije so pokazale, da se igralci ob preveč pogostem srečavanju z goljufi igre hitreje naveličajo in prenehajo z igranjem, kar negativno vpliva na dobičkonosnost. Zaradi tega je podjetjem v interesu, da se prepreči goljufanje. Da bi to dosegli, se v delovanje igre integrira sistem proti goljufanju. Dandanes je teh sistemov kar nekaj na razpolago, kot so Punkbuster, Valve Anti-Cheat, nProtect Gameguard in Easy Anti-Cheat. Preden so bile te rešitve objavljene, so morali razvijalci iger sami poskrbeti za razvoj tovrstnega sistema.

Večino današnjih obstoječih sistemov proti goljufanju so sestavljeni iz standardne arhitekture. Vsebujejo štiri komponente: strežnik v oblaku, gonilnik, zunanji program in notranji program v obliki dinamične povezovalne knjižnice. Komponente so med seboj povezane in delujejo kot enoten sistem.

Za preslepitev sistema proti goljufanju je treba premagati vse njihove zaščitne funkcije. S tradicionalnim pristopom je pogosto treba premagati vsako funkcijo posebej, kar je zamudno in zahtevno opravilo. Naš pristop predlaga ločitev programa za goljufanje od samega računalnika, kjer deluje sistem proti goljufanju. Sistem nenehno skenira računalnik za zlonamerno obnašanje, kar pa pomeni, da postane ob ločitvi program za goljufanje izven dosega sistema za preprečitev goljufanja. S tem premagamo vse zaščitne funkcije.

S tem, ko ločimo program za goljufanje od računalnika, kjer deluje igra, se pojavita dva nova problema: kako pridobiti informacije in kako pošiljati ukaze igrici. Tu obstaja več rešitev, vsaka s svojimi prednostmi in slabostmi. Uporaba našega pristopa nam onemogoča direktni dostop do spomina in manipulacije programske kode igrice. Zaradi tega je po našem mnenju integracija našega pristopa bolj primerna za goljufanje na način oponašanja vedenja navadnih igralcev. Zaradi omejenosti morda ne bo mogoče pridobiti vseh informacij o igri, kot bi jih lahko sicer z uporabo tradicionalnih pristopov. Kljub temu je velika prednost uporabe naše metode v varnosti, saj zagotavlja nezaznanljivost in s tem zaščito pred sankcijam. To je zaradi tega, ker sistem proti goljufanju ne more ločiti med poštenimi igralci in goljufi. Podjetja, ki izdelujejo sisteme proti goljufanju, lahko sicer prilagodijo svojo rešitev na način, da preprečijo uporabo naše metode, smo pa mnenja, da je bolj malo verjetno, da bi lahko zaznali in sankcionirali goljufive uporabnike.

Za zaključno nalogo smo implementirali program za goljufanje v igri Albion Online.

Program izvaja trgovske misije in s tem oponaša poštenega uporabnika. Kot nagrado za uspešno opravljeno trgovsko misijo je igralec nagrajen z izdelkom, ki ga lahko proda na tržnici za dobiček. V implementaciji smo uporabili metodo vohanja paketov za pridobitev informacij iz igre. Za pošiljanje ukazov smo se odločili za zlorabo aplikacije TightVNC, s katero pošiljamo vhodne ukaze, ki oponašajo tipkovnico in miško. Rešitev smo objavili na virtualni računalnik v oblaku, kjer se izvaja program za goljufanje in aplikacija TightVNC. Na lokalnem računalniku se izvaja igrica, sistem proti goljufanju in strežnik TightVNC, na katero se poveže virtualni računalnik v oblaku. Ves promet iz lokalnega računalnika je preusmerjen na računalnik v oblaku z uporabo povezave VPN, kar omogoča prestrezanje paketov in odkrivanje informacij o delovanju igrice.

Za uspešno opravljanje trgovske misije smo sprogramirali premikanje igralca po virtualnem svetu in interakcijo z osebami NPC. Postopki misije so sledeči: teci do banke, naredi polog nagrajenih izdelkov in vzemi potrebne izdelke za začetek misije, teci do lokacije za pričetek misije, prični misijo, teci do tihotapca, napreduj z misijo, teci nazaj do mesta in končaj misijo. Če se vsi omenjeni postopki uspešno opravijo je igralec nagrajen. Obstaja pa tveganje, da je naš igralec med opravljanjem misije napaden s strani drugih igralcev, kar lahko privede do smrti našega igralca in neuspešno opravljeno misijo. V trenutnem stanju se takšne situacije ne obravnava in program se zaključi z izvajanjem. Implementirali smo funkcijo snemanja poti, kar pomeni, da lahko uporabniki sami določijo, po kakšni poti program izvaja postopek teci do tihotapca in teci nazaj do mesta.

Program smo zagnali na 303 poskusih, med katerim je prišlo do napake v 8.6 % primerih. Povprečni čas izvajanja misije je 17,8 minute, kar pomeni, da lahko pričakujemo 3,3 izvedene trgovske misije na uro. Z uporabo naštetih podatkov smo preračunali, da lahko pridobivamo do 283.140 nagrajene valute na uro. V prihodnosti lahko izboljšamo program na način, da znižamo možnost neuspešno opravljene misije, kot tudi da zmanjšamo povprečni čas za izvajanje misije. S tem lahko povečamo dobičkonosnost uporabe našega programa.

8 References

[1] Matthew Bamberger and Nicholas Shaffner. Anti-cheat facility for use in a networked game environment, October 30 2012. US Patent 8,302,199.

[2] Nick Cano, editor. *Game Hacking: Developing Autonomous Bots for Online Games*. William Pollock, No Starch Press, 2016.

[3] changeofpace. Mouclassinputinjection. `https://github.com/changeofpace/MouClassInputInjection`, July 2019. Accessed: 2022-07-18.

[4] Mia Consalvo. *Cheating: Gaining advantage in videogames*. MIT press, 2009.

[5] Lorenzo Franceschi-Bicchierai. For 20 years, this man has survived entirely by hacking online games. `https://www.vice.com/en/article/59p7qd/this-man-has-survived-by-hacking-mmo-online-games`, July 2017. Accessed: 2022-07-18.

[6] Atish Ghoshal. Ethics in esports. *Gaming Law Review*, 2019.

[7] Exit Games GmbH. Photon Realtime. `https://www.photonengine.com/Realtime`. Accessed: 2022-07-18.

[8] Sandbox Interactive GmbH. Albion Online. `https://albiononline.com`. Accessed: 2022-07-18.

[9] Sandbox Interactive GmbH. Trade missions - Albion Online wiki. `https://wiki.albiononline.com/wiki/Trade_Missions`. Accessed: 2022-07-18.

[10] Samuel Horti. Elder scrolls blades hacks explained: what do cheats and mods do. `https://www.gamesradar.com/elder-scrolls-blades-hacks-cheats/`, May 2019. Accessed: 2022-07-18.

[11] Epic Games Inc. Anti-cheat interfaces documentation. `https://dev.epicgames.com/docs/services/en-US/GameServices/AntiCheat/index.html`. Accessed: 2022-07-18.

[12] Epic Games Inc. Easy Anti-Cheat. `https://www.easy.ac/`. Accessed: 2022-07-18.

[13] Carl E Landwehr, Alan R Bull, John P McDermott, and William S Choi. A taxonomy of computer program security flaws, with examples. Technical report, Naval Research Lab Washington D.C., 1993.

[14] Peter Laurens, Richard F Paige, Phillip J Brooke, and Howard Chivers. A novel approach to the detection of cheating in multiplayer online games. In *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*. IEEE, 2007.

[15] Samuli Lehtonen. Comparative study of anti-cheat methods in video games. Master's thesis, University of Helsinki, Helsinki, Finland, March 2020.

[16] GlavSoft LLC. TightVNC - VNC-compatible remote control / remote desktop software. `https://www.tightvnc.com/`. Accessed: 2022-07-18.

[17] CBS News Matt Slagle. Cheats could ruin online gaming. `https://www.cbsnews.com/news/cheats-could-ruin-online-gaming/`, December 2002. Accessed: 2022-07-18.

[18] Alissa McAloon. South Korea cracks down on cheaters with law targeting illicit game mods. `https://www.gamedeveloper.com/console/south-korea-cracks-down-on-cheaters-with-law-targeting-illicit-game-mods`, December 2016. Accessed: 2022-07-18.

[19] Gabe Newell. Valve, VAC, and trust. `https://www.reddit.com/r/gaming/comments/1y70ej/valve_vac_and_trust/`, February 2014. Accessed: 2022-07-18.

[20] BBC News. Valve challenged over anti-cheating tools. `https://www.bbc.com/news/technology-26240140`, February 2014. Accessed: 2022-07-18.

[21] Joel Noguera. Unveiling the underground world of anti-cheats. In *Black Hat Europe 2019*, London, United Kingdom, December 2019. UBM Tech.

[22] Morgan Park. Call of duty player tries to prove he isn't cheating, accidentally proves he's cheating. `https://www.pcgamer.com/call-of-duty-player-tries-to-prove-he-isnt-cheating-accidentally-proves-hes-cheating/`, March 2022. Accessed: 2022-07-18.

[23] Pedro Pina. Computer games and intellectual property law: Derivative works, copyright and copyleft. In *Business, Technological, and Social Dimensions of Computer Games: Multidisciplinary Developments*. IGI Global, 2011.

[24] MMO Populations. Top MMOs in 2021 - MMO populations & player counts. `https://mmo-population.com/top/2021`, January 2022. Accessed: 2022-07-18.

[25] Rake. Anticheat nProtect Gameguard bypass. https://guidedhacking.com/threads/anticheat-nprotect-gameguard-bypass.8008/, January 2011. Accessed: 2022-07-18.

[26] Rake. Internal vs. external hacks - what's the difference? https://guidedhacking.com/threads/internal-vs-external-hacks-whats-the-difference.8808/, July 2013. Accessed: 2022-07-18.

[27] Eric Steven Raymond. The case of the quake cheats. http://www.catb.org/~esr/writings/quake-cheats.html, December 1999. Accessed: 2022-07-18.

[28] Waranyoo Ronkainen. Prevention vs detection in online game cheating. Bachelor's thesis, University of Oulu, Oulu, Finland, November 2021.

[29] Jagex Ltd. Runescape. Unbalanced trade reminder. https://secure.runescape.com/m=news/unbalanced-trade-reminder, December 2007. Accessed: 2022-07-18.

[30] Jagex Ltd. Runescape. Bot-busting and bonuses for all. https://secure.runescape.com/m=news/bot-busting-and-bonuses-for-all, October 2011. Accessed: 2022-07-18.

[31] Jordan Sissel. xdotool. https://github.com/jordansissel/xdotool. Accessed: 2022-07-18.

[32] Stephen E Siwek. Video games in the 21st century. *Entertainment Software Association*, 2007.

[33] SnGmng. What internal/external hacks detection methods are known? https://guidedhacking.com/threads/what-internal-external-hacks-detection-methods-are-known.13021/, July 2019. Accessed: 2022-07-18.

[34] Irdeto Media Team. Widespread cheating in multiplayer online games drives gamers in Asia Pacific. https://irdeto.com/news/widespread-cheating-in-multiplayer-online-games-drives-gamers-in-asia-pacific-away/, June 2018. Accessed: 2022-07-18.

[35] The Tcpdump team. TCPDump. https://www.tcpdump.org/. Accessed: 2022-07-18.

[36] The Wireshark team. Wireshark. https://www.wireshark.org. Accessed: 2022-07-18.

[37] Xfce Development Team. Xfce desktop environment. https://www.xfce.org/. Accessed: 2022-07-18.

[38] Reddit theonlybond. VAC now reads all the domains you have visited and sends it back to their servers hashed. https://www.reddit.com/r/GlobalOffensive/comments/1y0kc1/vac_now_reads_all_the_domains_you_have_visited/, February 2014. Accessed: 2022-07-18.

[39] Michael VanKuipers. Riot's approach to anti-cheat. https://technology.riotgames.com/news/riots-approach-anti-cheat, July 2018. Accessed: 2022-07-18.

[40] Jagex Ltd. Runescape Wiki. Botwatch. https://runescape.fandom.com/wiki/Botwatch, May 2022. Accessed: 2022-07-18.

[41] Wikipedia. INCA Internet. https://en.wikipedia.org/wiki/INCA_Internet. Accessed: 2022-07-18.

[42] Wikipedia. Punkbuster. https://en.wikipedia.org/wiki/PunkBuster. Accessed: 2022-07-18.