

# QUACK – Quick Universal Anti-Cheat Kit

Jonathan Berkeley

N00181859

Supervisor: Cyril Connolly

Second Reader: Catherine Noonan

Year 4 2021-22

DL836 BSc (Hons) in Creative Computing

## Abstract

This application is an anti-cheat software solution for video games. Players cheating in multiplayer video games has become a problem that threatens the continued existence of some video games.

Large multiplayer video game companies spend millions protecting their games from cheating, but for independent developers with a low spending budget there are few options. This software has been produced to fill that void, giving essential protection from the most common attack vectors that cheat software exploits.

This anti-cheat software aims to raise the skill level required to produce cheats for games, thus reducing the number of cheaters and publicly available cheats for a protected game. Additionally, when players are caught cheating, they can be punished. This threat of punishment further disincentivises players to cheat.

## Acknowledgements

// todo

// Supervisor / Second reader / <https://unknowncheats.me> community / research papers / etc

**DECLARATION:**

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student: Jonathan Berkeley (N00181859)

Signed: Jonathan Berkeley

---

## Table of Contents

Abstract.....	2
Acknowledgements.....	3
1 Introduction .....	8
2 Research.....	9
2.1 Introduction.....	9
2.2 Cheating in video-games .....	9
2.1 History of cheating .....	10
2.2 Cause and motivation.....	10
2.3 Types of cheats.....	11
2.3.1 Hard cheats .....	11
2.3.2 Soft cheats.....	13
2.4 Anti-Cheat methods on the client .....	13
2.4.1 Difficulties of implementation .....	13
2.4.2 Purpose of client-side anti-cheat.....	14
2.4.3 User-mode methods .....	14
2.4.4 Code encryption and packing .....	14
2.4.5 Signature based cheat detection .....	16
2.5 Kernel-mode anti-cheat .....	16
2.5.1 Privacy and security concerns.....	16
2.5.2 Identifying previous cheaters .....	17
2.5.3 Server-side anti-cheat .....	17
2.5.4 Difficulties of implementation .....	19
2.5.5 Identifying previous cheaters .....	19
2.6 Summary .....	20
2.6.1 Limitations: .....	20
3 Requirements.....	21
3.1 Introduction.....	21
3.2 Requirements gathering.....	21
3.2.1 Similar applications .....	21

3.2.2	Interviews.....	23
3.2.3	Survey.....	24
3.3	Requirements modelling.....	36
3.3.1	Personas.....	37
3.3.2	Functional requirements .....	39
3.3.3	Non-functional requirements .....	39
3.4	Feasibility.....	40
3.5	Conclusion .....	42
4	Design.....	43
4.1	Introduction.....	43
4.2	Application design.....	43
4.2.1	Technologies .....	45
4.2.2	Design pattern and paradigm .....	46
4.2.3	Database design.....	48
4.2.4	Visual design .....	49
4.3	Process design .....	53
4.4	Conclusion .....	54
5	Implementation .....	55
5.1	Introduction.....	55
5.2	Development environment.....	55
5.3	Agile development and sprints .....	57
5.3.1	Research.....	57
5.3.2	First sprint .....	57
5.3.3	Second sprint .....	57
5.3.4	Third sprint.....	58
5.3.5	Fourth sprint .....	58
5.3.6	Fifth sprint.....	60
5.3.7	Sixth sprint .....	60
5.3.8	Seventh sprint.....	61
5.4	Database.....	62

5.5	Backend .....	62
5.6	Frontend .....	62
5.7	Conclusion .....	62
6	Testing .....	63
	Bibliography .....	64

# 1 Introduction

The term “anti-cheat” refers to a suite of software that aims to inhibit cheating in video-games. Cheating in video-games is not typically an issue in singleplayer games, many singleplayer video-games even come with their own cheat codes for the player to use if they so choose.

The issue arises when cheats are used to gain an unfair advantage in multiplayer games, which can ruin the experience for other players. This often leads to a drop off in player base and revenue of affected games. Cheating poses an existential threat to competitive multiplayer games (Irdeto, 2020).

As an example of a game cheat, imagine there is an unprotected game where coordinates of a player’s position in 3D space is processed on the client, and sent to a game server; a cheating player could develop software to change their player’s position values in memory, giving them the ability to teleport to any location! Many different types of cheats exists, which vary from game to game, they provide an unfair advantage to players that are willing to cheat.

There is a large market in protecting games from cheaters, there are many multi-million-dollar companies that focus solely on the production of anti-cheat software which is sold to game companies (Owler, 2022a; Owler, 2022b; Dnb, 2022).

// todo technologies discussion

// GitHub / Git / C++ / Visual Studio / Trello / Forums

// todo explain future terminology ?



## 2 Research

### 2.1 Introduction

Cheating in video-games has become a rampant problem which threatens the profitability and continued existence of online video-games. Irdeto found that “a mere 12% of online gamers have never had their experience negatively impacted by cheating”, and that “77% of online gamers are likely to stop playing a multiplayer game online if they think other players are cheating” (Irdeto, 2018a; Irdeto, 2018b).

In this literature review, “cheating” will refer to any form of intentional software manipulation to gain an advantage in a video-game. “Anti-Cheat” will refer to the efforts to inhibit this behaviour through software.

This literature review will identify the causes of cheating and investigate the various software-based methods used for preventing cheating in video-games and will compare their effectiveness according to current research.

### 2.2 Cheating in video-games

This distinction of what constitutes as cheating or not is important for Anti-Cheats, an anti-cheat needs to have a clear definition of what exactly quantifies as a cheat, which will define what it should look for, and what it should punish.

There are disagreements on how to exactly define cheating, and what qualifies as cheating. Some simply define cheating in video-games as anything that gives players an unfair advantage (Mikkelsen, 2017), while others include even attempting unsuccessfully to parse a video-games memory as cheating (Consalvo, 2009). As a general definition of cheating, not pertaining to video-games specifically, J. Barton Bowyer defines cheating as “the advantageous distortion of perceived reality” (Bowyer, 1982).

The general consensus amongst most gamers is that cheating is any unfair advantage one player has over another in a multiplayer video-game (Been-Lirn Duh & Hsueh Hua Chen, 2009; Consalvo, 2009).

It’s important to make the distinction that some methods of cheating are outside the scope of most Anti-Cheat software solutions. Such methods of cheating include cheating by collusion, whereby a player works with another player to gain an advantage over other players in a way that is not intended (Such as working with an opponent to gain advantageous knowledge that would otherwise be unknown).

## 2.1 History of cheating

Cheating in video-games is not a new phenomenon, it has been around nearly as long as video-games have been (Corliss, 2019; FTI Consulting, 2021). Some of the earliest games contained built-in cheat codes, created on purpose by the game developer. Originally, cheat codes were not intended for use by players, instead it was a mechanic for the developers of the game to playtest and debug their game more easily. By using cheat codes, the game developers could skip through difficult parts of the game and get to the part that they needed to test (Corliss, 2019).

Players, through various methods, found these codes that had been left behind during the development of the game. This created a fascination in finding these hidden codes. Future games then intentionally created hidden cheat codes and left them for players to find, often with fun or humorous effects. Magazines and books were created on the topic, discussing, and revealing hidden codes for video-games (Corliss, 2019; Pochwatko & Giger, 2015).

## 2.2 Cause and motivation

What motivates players to cheat in video games? There is no one single reason, but there are some observable trends. The reason players cheat in multiplayer games can be broken up into several categories.

One of the main categories is people that cheat for fun, some of these people will have played legitimately to a point but became bored with the game and decided to cheat for a new experience. For others, the game was too difficult to begin with, if enemies are too good or the game is too challenging, some people resort to cheating. Some players are prompted to cheat out of frustration, such as players that decide to cheat after a losing streak, or after being defeated by other players that were cheating (Consalvo, 2009; gameranx, 2020).

There are cheaters that simply enjoy having power over others, or effortlessly beating others. Some will cheat to progress further in a game, if they are stuck on a level or have a low rank in a leader board that they want to improve. Another major category is those that cheat for the purpose of annoying others, these cheaters find entertainment in ruining the experience of other players and use cheats as a tool to do so (Consalvo, 2009; gameranx, 2020).

## 2.3 Types of cheats

There are a large variety of different ways to gain an unfair advantage in video games. Some types of cheats are specific to the game that they affect, whilst others are more general and can be applied across genres of games. There is a distinction between types of cheating that requires the cheater to make modifications to their copy of the game, and cheating by other means, such as exploiting bugs in the game.

### 2.3.1 Hard cheats

“Hard cheats” describe cheats that make modifications to game code to obtain an unfair advantage. This is what anti-cheat software mostly focuses on countering. The following include some of the most well-known and pervasive types of “hard cheats”:

- Aimbot – This type of cheat is typical in games that require a player to aim with precision, such as a shooter game. An aimbot will automate the players aim to give them a higher degree of precision.
- Wallhack / ESP – This type of cheat enables the cheater to gain information they should not have access to, such as seeing other players through walls or behind obstacles.
- Speedhack – This type of cheat increases the speed at which events in the game happen, enabling cheaters to do actions faster than others, such as move around or shoot.

(FTI Consulting, 2021; Lehtonen, 2020)

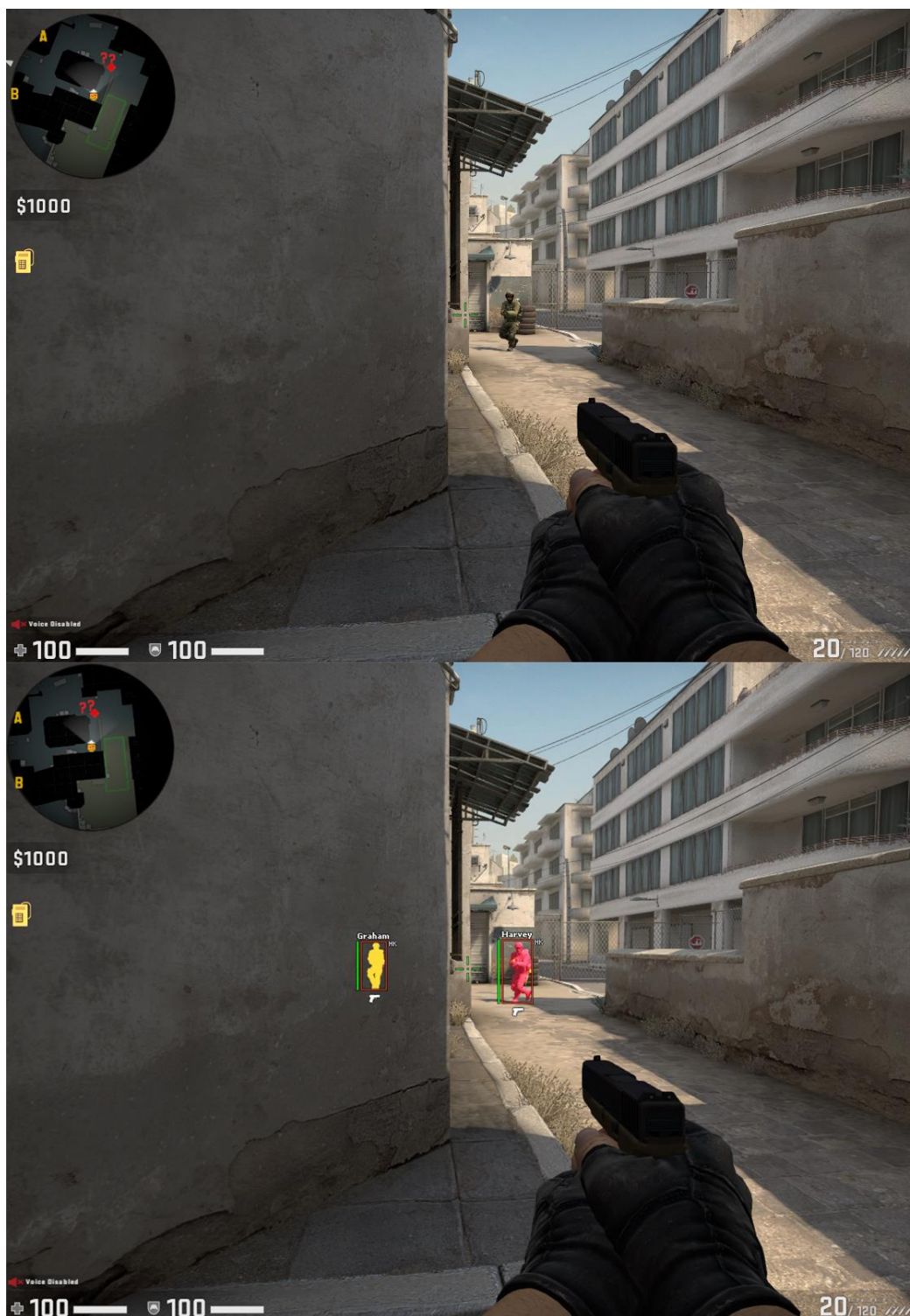


Figure 2.1 - Before and after showing the effect of a Wallhack/ESP cheat, enemies become clearly visible even when behind walls. Picture taken in the video game Counter-Strike: Global Offensive. (Image created by author)

### 2.3.2 Soft cheats

“Soft cheats” describe cheats that do not require the cheater to modify the game. The following are prominent types of soft cheats:

- Bug and exploit abuse – Here a cheater can abuse flaws in the video games design to gain an advantage, they can do this without the need for any software (Delfy, 2020).
- Collusion – This is where multiple people collaborate using external tools, such as a messenger chat, with the purpose of exchanging information that would not otherwise be available to them.

(Been-Lirn Duh & Hsueh Hua Chen, 2009; Lehtonen, 2020)

## 2.4 Anti-Cheat methods on the client

Preventing players from cheating typically begins on the client. The player downloads software with the game they want to play, this software employs various techniques to prevent the player from cheating, or to signal to a server that the player is attempting to, or is actively cheating. The term “On the client” refers to the fact that the anti-cheat software is on the computer of the player.

### 2.4.1 Difficulties of implementation

This way of preventing cheating has various vulnerabilities, due to the program that prevents the player from cheating being on the player’s computer. It’s possible for an experienced cheat developer to circumvent the protective software, circumventing anti-cheat software is referred to as an anti-cheat bypass. (Kaiser et al., 2009)

Whilst it is possible for anti-cheat developers to continue to patch ways that an anti-cheat is bypassed, this leads to an “arms-race with cheat developers”, as described by various prominent anti-cheat developers (Paoletti, 2021; Valve Anti-Cheat team, 2016). Whereby cheat developers continue to develop methods to bypass client-side anti-cheat, and anti-cheat developers continue patching the vulnerabilities used to create the bypass. Continuing to participate in the arms-race against cheat developers is expensive, it’s not a viable option for many small game companies.

“Developers have compared developing proactive anti-cheats to an endless treadmill ... whoever evolves faster wins, and due to the higher manpower on the side of the cheaters, they tend to win it out most of the time.” (Li, 2021).

#### 2.4.2 Purpose of client-side anti-cheat

The primary advantage of client-side anti-cheats is that they raise the skill ceiling of developing cheats. Companies continue to produce and maintain client-side anti-cheat despite the inability for it to prevent cheating completely for this reason. An inexperienced cheat developer will be unable to bypass a mature client-side anti-cheat. This restricts the development of these programs to a few experts, and therefore dramatically reduces the amount of cheating in the video game (Paoletti, 2021).

Additionally, the cheats that are created for well protected games are typically limited to small sized private communities, it becomes unviable for cheat developers to distribute cheats publicly for sale as this massively increases the chances it will be detected (Johnson, 2016).

#### 2.4.3 User-mode methods

“User-mode” refers to software that runs in the typical address space of the computer, these processes have regular permissions. In the Windows operating system, each process that runs in user-mode has its own virtual address space, which is private to other applications. The applications in user-mode have limited capabilities, they cannot directly manipulate the operating system or read arbitrary memory, they must obey the rules and restrictions the operating system puts on them (Linfo, 2005b; Microsoft, 2021c).

#### 2.4.4 Code encryption and packing

There are many different methods of inhibiting cheating available to user-mode anti-cheat software. Code encryption and executable packing is one such method. In a game employing this protection, the binary executable for the game is encrypted or packed, and a process of unpacking begins each time the game is run. This inhibits “static analysis”, the analysis and reverse engineering of files that are not running. A cheat developer has to spend time reversing the unpacking or unencrypting routine in order to properly create cheats for the protected game (Atallah et al., 2004; DeepSource, 2020).

This protection can inhibit cheat development even further by changing frequently, such as every update of the game. This will require the cheat developer to continually update their cheat to get around the encryption routine (Lehtonen, 2020).

Making the process of developing cheats as laborious and skill dependent as possible is the goal of modern-day anti-cheat software. Difficult to reverse code encryption routines that change regularly are an effective way to increase both the time it takes to develop a cheat, and the amount of skill required to do so.

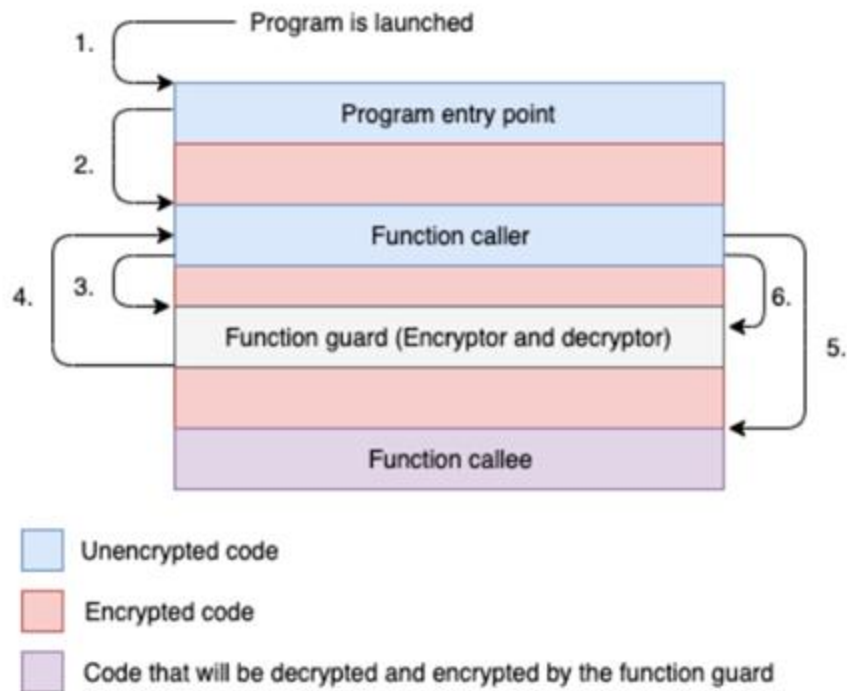
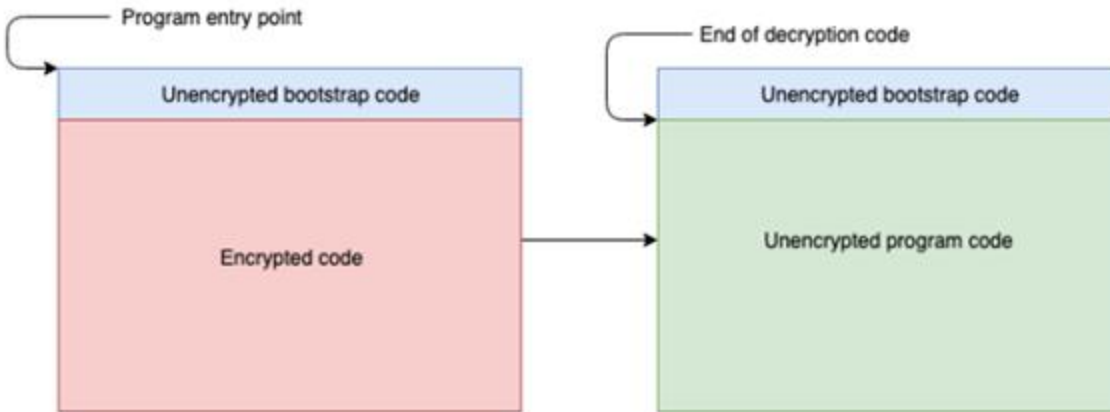


Figure 2.2 – The top diagram shows bulk decryption (where decryption is performed at the games launch). The lower diagram shows on-demand decryption, where the code is decrypted as the program needs it and encrypted again when the program finishes using it. (Lehtonen, 2020)



#### 2.4.5 Signature based cheat detection

Signature based detection of cheat software is one of the most fundamental ways a user-mode anti-cheat operates. It scans the computer's memory for programs that are known to be malicious. The anti-cheat has a database of cheat program signatures to search for. These signatures are unique hashes made from the memory of known cheat programs. The anti-cheat continually scans memory on an interval, generating and sending signatures of programs back to the server belonging to the game company. When a programs signature matches that of one of the known cheat programs, it's definite that the player is running known cheat software or a variant of it (Kaiser et al., 2009; Lehtonen, 2020).

When a new cheat signature is added to the database, the anti-cheat typically does not ban players it detects instantly. If the anti-cheat banned instantly, it would warn other users of that same cheat software not to cheat on that particular day.

To counteract this, anti-cheats often ban in "ban-waves", whereby the anti-cheat will refrain from banning any cheating player for some period of time, it will instead build up a database of cheating players without them knowing that they have been caught. Then after a large list of cheating players has been created, all the players that had cheated over that period of time with the detected software are banned together (Franceschi-Bicchierai, 2021; Higgins, 2016; Vape, 2019).

#### 2.5 Kernel-mode anti-cheat

"Kernel-mode" refers to the highest privileged section of address space in a computer. In Windows, all code in this section shares a single virtual address space, there is no isolation between code running in kernel-mode. This means that code running in kernel-mode can directly affect other code inside the kernel, including the operating system itself. Software in this mode on a computer can affectively do as it pleases, it has the ability to do whatever the hardware will allow it to. (Linfo, 2005a; Microsoft, 2021c)

##### 2.5.1 Privacy and security concerns

Some anti-cheat for video games can employ a kernel-mode module which will have unrestricted access to the data and events on the machine, the anti-cheat escapes the ability to be reversed by purely user-mode processes. This gives the anti-cheat a far larger degree of power than purely user-mode counterparts, though it's not without a cost (Ciholas et al., 2020). There are privacy concerns for many people with allowing an invasive software like this to run on their computer (Warren, 2021).

Another concern people have with allowing a kernel-mode anti-cheat onto their computer is that the anti-cheat itself can be vulnerable, a bug in the anti-cheat software could enable malware to obtain total power of a computer. Malware running in kernel mode is extremely difficult to remove, as it can evade anti-virus software (Alder, 2020).



Conversely, there are security developers that argue that the same risks apply for any software running on a computer, and that user-mode malware can destroy a computer as easily, it doesn't need to find a vulnerable kernel driver (vmcall, 2020).

### 2.5.2 Identifying previous cheaters

After a cheater has been caught and banned, there is still a battle to keep them banned. A cheater can just buy or create another account and continue playing if there are no countermeasures. Client-side software has more power to do this than server-side software. The various pieces of hardware used by the computer have IDs. These IDs are unique to that piece of hardware (Microsoft, 2021b; time2win, 2019; Yin-Poole, 2021).

Client-side anti-cheat software can gather a list of hardware IDs (HWIDs), for example the HWID of the monitor, the motherboard in the computer, and the graphics card. Then when a new player installs and plays the game for the first time, they can check the IDs of the hardware against those of previously banned players and see if any match (Unknowncheats, 2018; time2win, 2019; Yin-Poole, 2021).

This means that a cheater has to either spoof the HWIDs of their hardware, or buy new hardware every time they are banned if they wish to play again. Comprehensive implementations of this protection are one of the most effective ways to prevent reoffenders (Unknowncheats, 2018; time2win, 2019; Yin-Poole, 2021).

### 2.5.3 Server-side anti-cheat

Server-side software is often used in conjunction with client-side anti-cheat. This type of anti-cheat is for multiplayer games, the software runs on the server that enables players to play with each other. The server is in the control of the video game company, this is in direct contrast to client-side anti-cheat solutions. The client sends data to the server, and to varying degrees of complexity, the server-side anti-cheat can read the data and discover cheating behaviour.

There are also best practices that can be taken to mitigate cheating when it gets past the anti-cheat software. Moving more of the game logic from the client onto the server prevents a cheater from modifying this data.

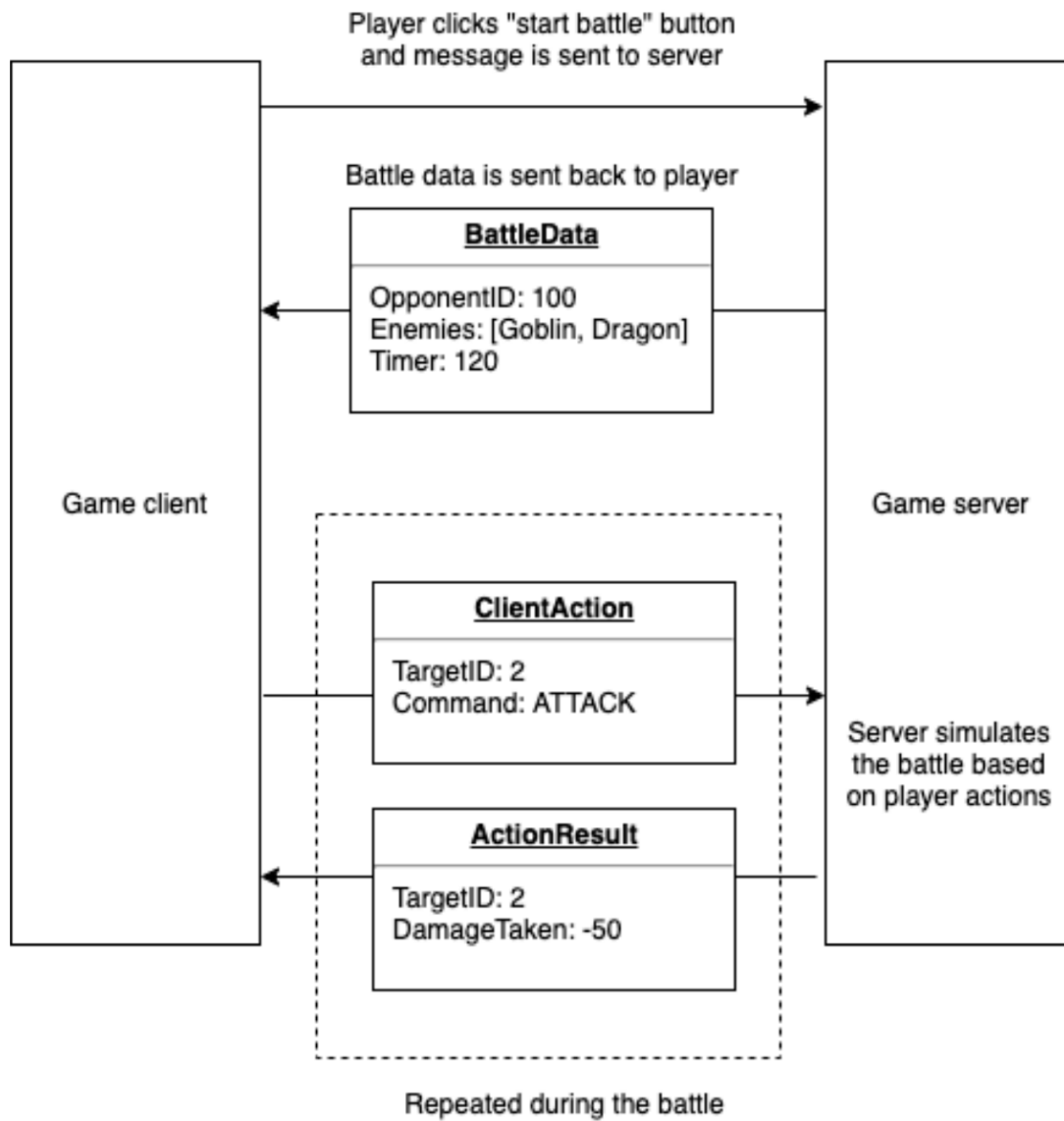


Figure 2.3 - Example of a theoretical game where computation of game events is performed on the server, the client purely sends desired actions, and the result is sent to the client. This model puts no trust in the client. (Lehtonen, 2020)

#### 2.5.4 Difficulties of implementation

There are several issues with having anti-cheat on the server. Generally, the more accurate and effective the anti-cheat software, the more of the server's computational power you will need to spend running the software. With client-side anti-cheat the computational power for the anti-cheat is spent on the client's machine. Moving game logic from the client to the game increases server-side computational costs more. This is especially costly for games which require accurate anti-cheat and have many players using the server (Kaiser et al., 2009).

The server-side anti-cheat is limited in what it can do, it cannot see if the player has cheat programs running to enhance performance. It's unable to prevent repeat offenders as effectively as client-side anti-cheat. A cheater willing to cheat subtly can fool server-side anti-cheats, which wouldn't be able to differentiate the cheating player from a very good player. This type of cheating is commonly referred to as "legit cheating" – cheating in a subtle way that appears legitimate. (Unknowncheats, 2015)

#### 2.5.5 Identifying previous cheaters

Server-side options to identify previous cheaters are much more limited and considerably less effective than client-side options. A common way to identify a previous cheater historically has been to ban them by IP address. However, an IP address is easy to change, and modern networks are usually dynamic, meaning they will change IP address over time (Vaughan-Nichols, 2019). Additionally, since IP addresses are reused, it's also possible to accidentally ban a legitimate player unlucky enough to receive the previous IP address of a cheater (Unknowncheats, 2020).

Modern and more effective server-side methods of identifying repeat offending cheaters use a system whereby all players are required to give information such as a mobile number and/or an email. Once verified as real, the player can play. If the player cheats and is banned, they will need to obtain a new mobile number and/or email to play again, otherwise the server would recognize the cheating player if they came back on a new account with the same details (Thorn, 2018; Seppala, 2016).

## 2.6 Summary

Cheating is a pervasive problem which is endemic to the video game industry. It is impossible to ever solve the problem of cheating, however as seen in this research there are many methods that can be employed to mitigate the problem and inhibit the development of cheats.

There is one no best method of preventing cheating for any given game, it depends on the resources available, the type of game, and more.

//todo write more here

### 2.6.1 Limitations:

This literature review has described a brief overview of some of the primary methods that are available to counteract cheating, however there are many methods which are not covered in this review.

Note that some sources are taken from forums, as information on the exact implementation of anti-cheat methods are not provided by anti-cheat developers. They do not officially document this information as it would aid in the development of cheats. Therefore, some sources occasionally rely on information from cheat developers.

//todo write more here

## 3 Requirements

### 3.1 Introduction

This chapter will reflect the requirements phase which focused on researching the requirements for the software. This entails discovering what the software will need to do in terms functionality, and what it's userbase will expect from it.

This phase is important because otherwise there's a larger possibility to produce solutions to problems few users have, whilst leaving other problems neglected.

### 3.2 Requirements gathering

Outline for the process of gathering information about the requirements for this software project.

#### 3.2.1 Similar applications

The production and maintenance of anti-cheat software is a multi-million-dollar industry. There are many anti-cheat software solutions available already.

##### 1. Easy Anti-Cheat (EAC) <https://www.easy.ac/en-us/>

EAC is an industry leading anti-cheat that protects over 140 games to date (Easy Anti-Cheat, 2022). This anti-cheat software is marketed towards game studios to reduce the impact of cheating in their games. EAC has a broad set of features, the exact feature list in place varies from game to game depending on need.

The company keeps their anti-cheat updated to the newest discovered vulnerabilities and cheats. Cheat software that is developed for EAC protected games are typically niche, limited to either private use or require paying high monetary subscription costs to access.

Different video games protected by this software have differing implementations, but most use a Kernel level anti-cheat.

2. BattlEye <https://www.battleye.com/>

BattlEye is another popular anti-cheat, this service is also marketed to game studios. Like EAC, it also has a varying implementation that differs from game to game. It has proven to be effective during its 17+ years of existence.

However, in recent years it has been the subject of some controversies and major vulnerabilities (Clayton Sterling Cyre, 2021; vmcall, 2020b).

BattlEye is a kernel-level anti-cheat.

3. PunkBuster <https://www.evenbalance.com/>

PunkBuster is an older anti-cheat that was formerly popular, originally released in the year 2000. It uses anti-cheat techniques such as signature scanning, which alone is considered primitive by today's standards. The anti-cheat doesn't use heuristics to catch cheaters (bashandslash, 2020).

PunkBuster has kernel-level functionality.

4. Valve Anti-Cheat (VAC) <https://help.steampowered.com/en/faqs/view/571A-97DA-70E9-FF74>

VAC is an anti-cheat developed by the large video game studio, publisher, and digital distribution service provider Valve. Despite being originally released in 2002, it is still maintained and in widespread use today. It is the anti-cheat of choice for Valve published multiplayer games.

VAC bans in waves, when a cheat is detected, it waits a varying duration of time to ban the users. This is done to catch as many cheaters using the cheat as possible.

Unusually, VAC is one of the very few anti-cheats that chooses not to run at kernel-level, instead it remains on application / userland level. This severely limits its capability to catch cheaters as cheaters can produce cheats that run at the kernel level, which userland applications have little ability to examine.

VAC is not an effective anti-cheat; cheating is rampant in VAC protected games. Additionally, it's easy to source free cheats for VAC protected games.

Valve have moved away from traditional anti-cheat, now focusing on the creation of a machine learning based approach. This machine learning project is called VACNet.

VACNet catches cheaters by observing behaviour of players. This is effective at catching blatant cheaters that do not try to hide their cheating behaviour.

5. Vanguard <https://playvalorant.com/en-gb/news/game-updates/valorant-anti-cheat-fall-2021-update/>

Vanguard is an anti-cheat developed by Riot Games for the video game Valorant. This anti-cheat differs from the rest named in this section; in that it was produced to protect a single game. This software is not sold or marketed to game studios, it's private proprietary software.

Riot Games has taken a 0-tolerance policy towards cheating, this software is a testament to that. Vanguard is an effective anti-cheat; it limits cheating software to highly priced private communities. The cheats that are produced for Valorant have limited features and capabilities.

Riot Games also introduced bounties to incentivise hackers to reveal techniques used to bypass the anti-cheat for a large monetary reward, aimed at further reducing the market for cheating (Fowler, 2020).

Vanguard is more invasive than other anti-cheats mentioned in this list, it starts when Windows starts, meaning it is always present on the system, even when the player is not playing the video game. As a result, it has been the subject of controversy (Minna Adel Rubio, 2020).

These aforementioned anti-cheats were picked for comparison as they are popular and have factors that separate them from the rest. There are many more anti-cheat applications that are not mentioned in this section.

### 3.2.2 Interviews

// todo

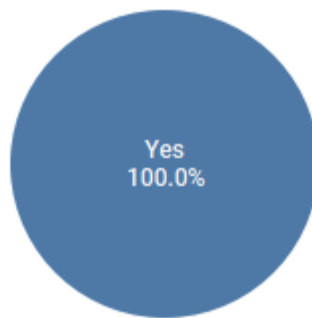
### 3.2.3 Survey

A survey was created during the requirements gathering phase to quantify the effect of cheating and measure the need for anti-cheat software in video games.

The survey was created using Google forms. Participants were first explained what the purpose of the survey was, and then asked for their consent to continue.

The survey was distributed to students between the ages 20 to 30 and received 35 responses. The first question proposed by the survey was the following:

Do you play video games?



*Figure 3.1 - First question on the survey and a visualisation of the responses*

Figure 3.1 shows that 100% of respondents to the survey play video games. This was asked as this survey is only relevant to people that play video games. Answering no to this question completes the survey without further questions.

All further questions from this point are therefore answered by people that play video games.



## How long on average do you spend playing video games per week?

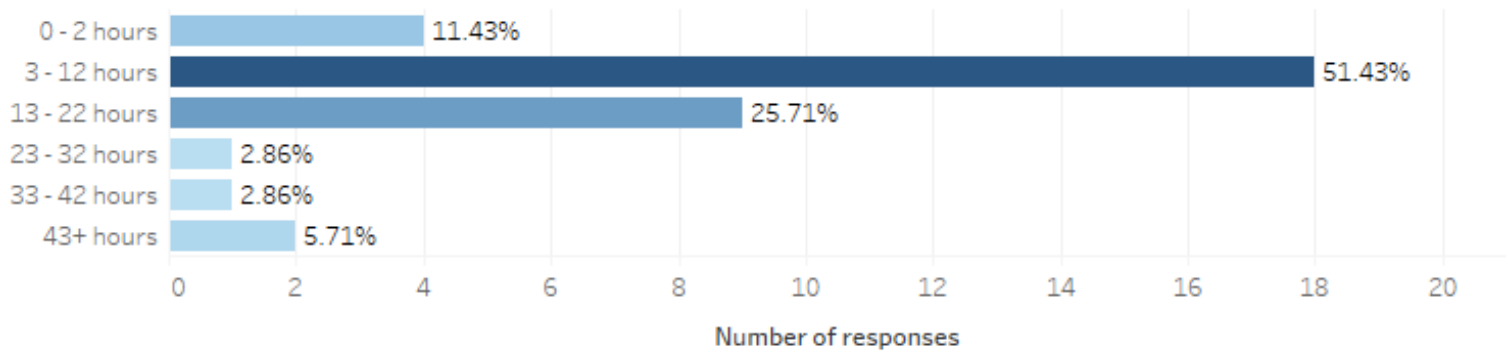


Figure 3.2 - Second question on the survey and a visualisation of the responses

Figure 3.2 shows that the majority of respondents play video games regularly, with only 11.43% of respondents playing less than 3 hours per week. This is important information to consider for the rest of the survey, as information from regular gamers is more likely to provide a meaningful insight into the cheating problem. Gamers that play games more regularly are more likely to encounter cheaters.

If you have a favourite video game, what is it?

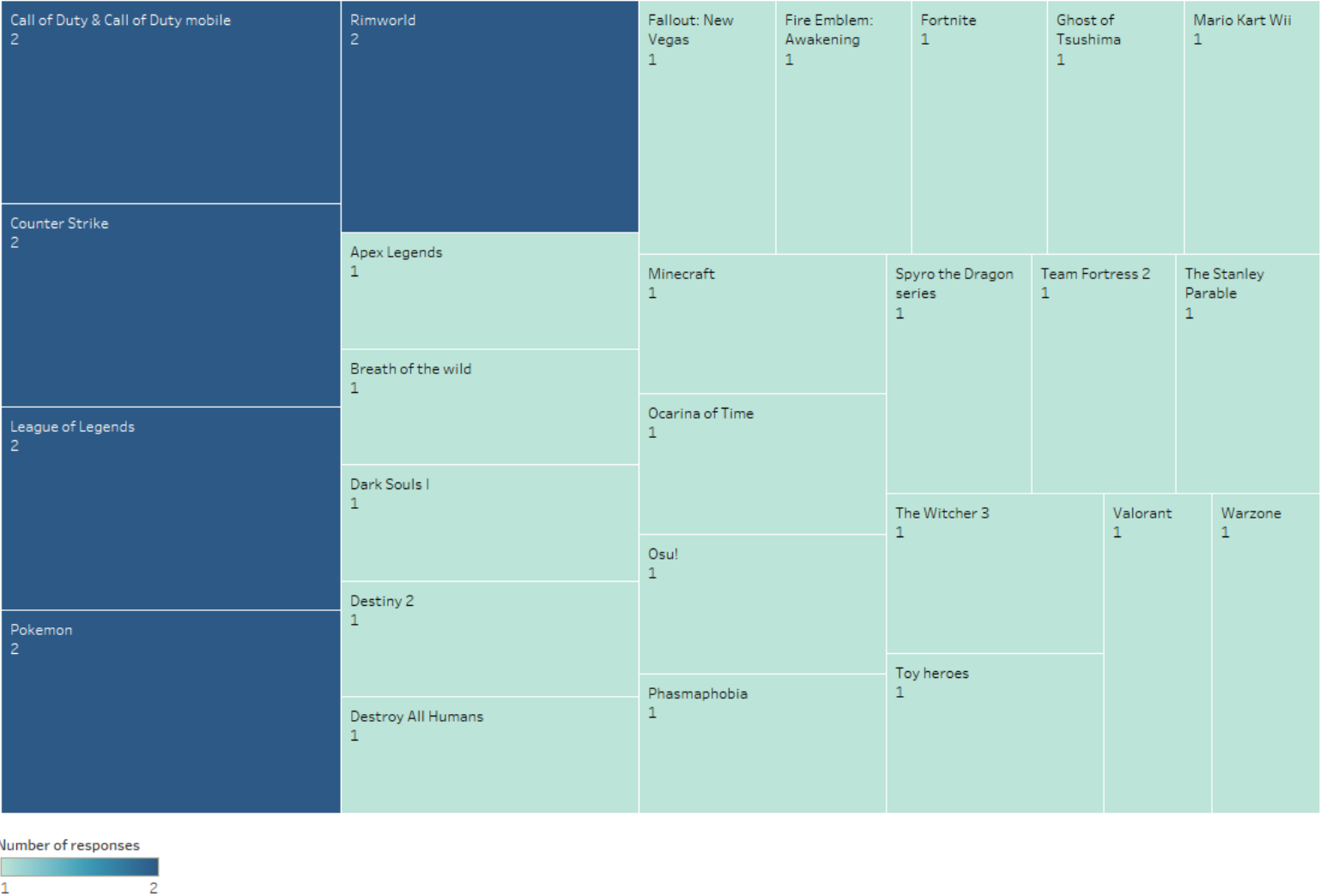


Figure 3.3 - Favourite games of respondents visualised

Figure 3.3 shows the responses to the optional third question on the survey. 31 out of 35 respondents chose to answer this question. This question differs from the others on the survey as it was not multiple choice, but instead it was an open question. Slight variations of the same answer were grouped for this visualisation.

The question was asked to understand if respondents were primarily playing the same genres or games, therefore possibly skewing the results.

The games that were mentioned were from a variety of different genres, showing that respondents are not all primarily playing the same game or genre. Additionally, many of the games mentioned have no multiplayer functionality.

## Prevalence of cheating by favourite game

Is cheating a common occurrence when you are playing video games?	If you have a favourite video game, what is it?	
Yes	Breath of the wild	1
	Call of Duty	1
	CS:GO	1
	Destiny 2	1
	Destroy All Humans	1
	Phasmaphobia	1
	Team Fortress 2	1
	Valorant	1
	Warzone	1
Maybe	Fire Emblem: Awakening	1
	Ghost of Tsushima	1
	LoL & League of Legends	3
	Mario Kart Wii	1
	Minecraft	1
	Ocarina of Time	1
	Rimworld	1
	Spyro the Dragon series	1
	The Witcher 3	1
No	Apex Legends	1
	Call of Duty mobile	1
	Dark Souls I	1
	Fallout: New Vegas	1
	Fortnite	1
	Osu!	1
	Pokemon	2
	Rimworld	1
	The Stanley Parable	1
	Toy heroes	1

Figure 3.4 - Visualisation of commonly occurring cheating compared to favourite game

Figure 3.4 shows a visualisation of respondent's answers to the question "Is cheating a common occurrence when you are playing video games?", displayed against what they had answered that their favourite game is.

It is important to note that 7/11 (63.64%) of the games in the "No" category shown in Figure 3.4 listed their favourite game as one that is either not multiplayer or has minor multiplayer functionality. Whereas 7/9 (77.78%) of the games in the "Yes" category are either exclusively multiplayer or have a primary focus on multiplayer. It can be concluded from this that cheating is a common occurrence to most of the respondents that enjoy primarily multiplayer games.

### Have you ever played against a cheater in a multiplayer video game?

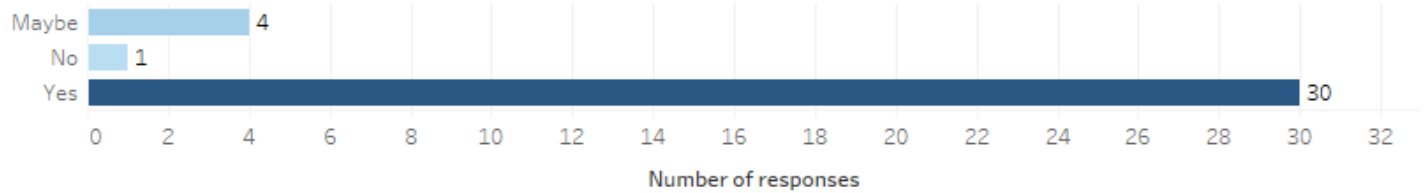


Figure 3.5 - Fourth question on the survey and a visualisation of the responses

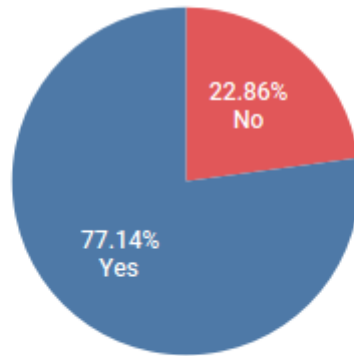
Figure 3.5 shows that a single respondent (2.86%) was confident that they had played against only legitimate players in multiplayer games. 30 respondents (85.71%) answered that they had played against a cheating opponent, with the remaining 4 respondents (11.43%) responding with “Maybe”.

It was expected prior to creating the survey that a high number of respondents would have encountered cheaters while playing a multiplayer video game at some point. One of the most popular First-Person Shooter games of all time “Counter Strike: Global Offensive” released the probability of encountering a cheater in a competitive multiplayer match; being a 1 in 40 chance for a competitive match to contain at least one cheater (Johnson, 2020).

Since a competitive match of Counter Strike: Global Offensive takes up to 90 minutes (CS:GO, 2021), a player that has played 60 hours’ worth of competitive has almost certainly encountered a cheater, whether knowingly or not. Considering the answers from question 2, shown in Figure 3.2, most players would encounter a cheater after several weeks.

Many cheaters do not make their cheating behaviour known; they cheat in secret by giving themselves a believable advantage. This makes it difficult for others to know for definite if enemies are cheating or not.

Has other players cheating had a negative impact on your experience of a video game?



*Figure 3.6 - Fifth question on the survey and a visualisation of the responses*

Figure 3.6 shows that 27 respondents (77.14%) had their experience of a video game negatively affected by the presence of a cheater. 8 respondents (22.86%) answered “No” to this question.

Some of the responses being “No” for this question can be explained due to a portion of these respondents either not having played against a cheater before, or not being sure that they had, as demonstrated in Figure 3.7.

# Exposure to cheating and it’s effect on players

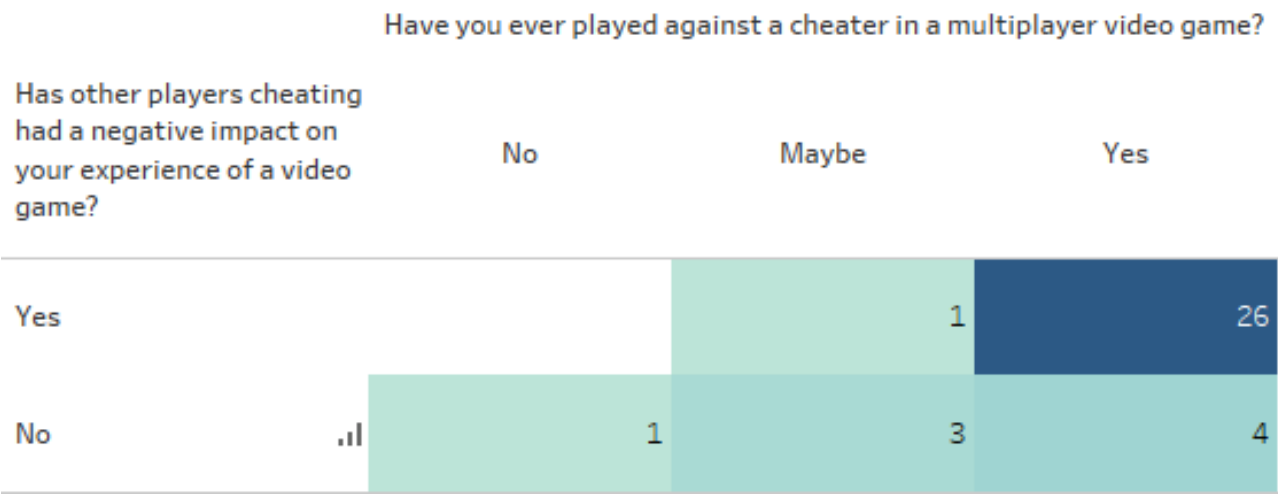


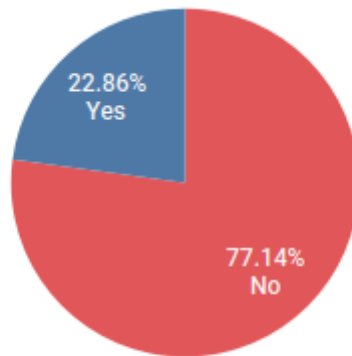
Figure 3.7 - Responses for questions 4 and 5 compared

It can be seen in Figure 3.7 that 4/8 of the respondents that answered “No” to the question shown in Figure 3.6 had not been confident that they had played against a cheater before. 30 respondents were confident that they had played against a cheater, and of this group, 26 of them (86.67%) had their experience negatively impacted.

The results from this question verify the intuitive, that playing with cheaters is a negative experience for the majority of people in video games.

This data gathered correlates with much larger sample sized surveys, 12% of online gamers surveyed by Irdeto had never had their gaming experience negatively impacted by cheaters (Irdeto, 2018).

Have you ever cheated in a multiplayer video game?



*Figure 3.8 - Sixth question on the survey and a visualisation of the responses*

Figure 3.8 shows that 8 respondents (22.86%) have cheated in a multiplayer video game at some stage. These answers indicate that cheating is not a niche activity that few players have experience with, but rather over a fifth of players have cheated.

One of the primary factors that encourages an uptake of cheating is other players cheating, if a player has been cheated against, they are more likely to start cheating themselves (Consalvo, 2009). Out of the 8 respondents that had admitted to previously cheating, all of them had answered to question 4 (shown in Figure 3.5) that they had played against a cheater in a multiplayer video game.

This suggests that reducing the number of cheaters in video games will also reduce the desire for players to begin cheating. This is the goal of anti-cheat software, to reduce the effect and number of cheaters in video games.

## Exposure to cheaters increasing likelihood of cheating

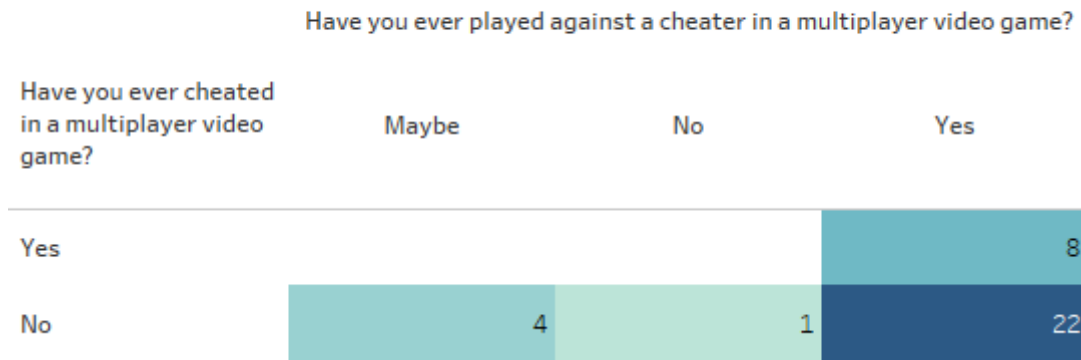


Figure 3.9 - Responses for questions 4 and 6 compared

Figure 3.9 verifies what was previously stated, that all 8 of respondents that cheated had played against cheaters. This aligns with the existing research in this field, that cheating in video games is a domino effect. The more prevalent cheating is, the more people take up cheating themselves.

## Do you think anti-cheat software is necessary for competitive multiplayer games?

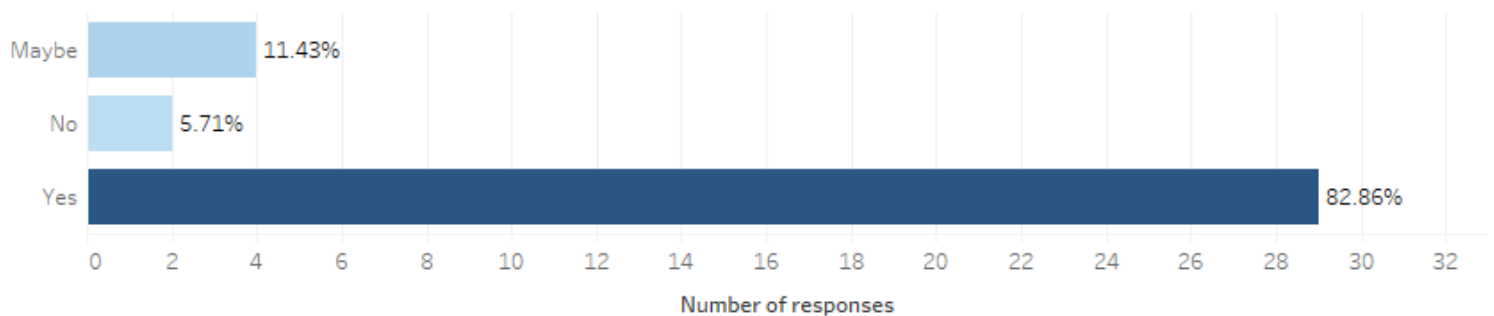


Figure 3.10 - Seventh question on the survey and a visualisation of the responses

Figure 3.10 shows that the vast majority of respondents (82.86%) to the survey believe anti-cheat is necessary for competitive multiplayer games.

The answers from this question are not surprising, successful competitive multiplayer video games have anti-cheat without exception. Without anti-cheat, there is little risk of getting caught for those that cheat secretly and therefore competitive integrity is jeopardised.



# Need for anti-cheat responses by group

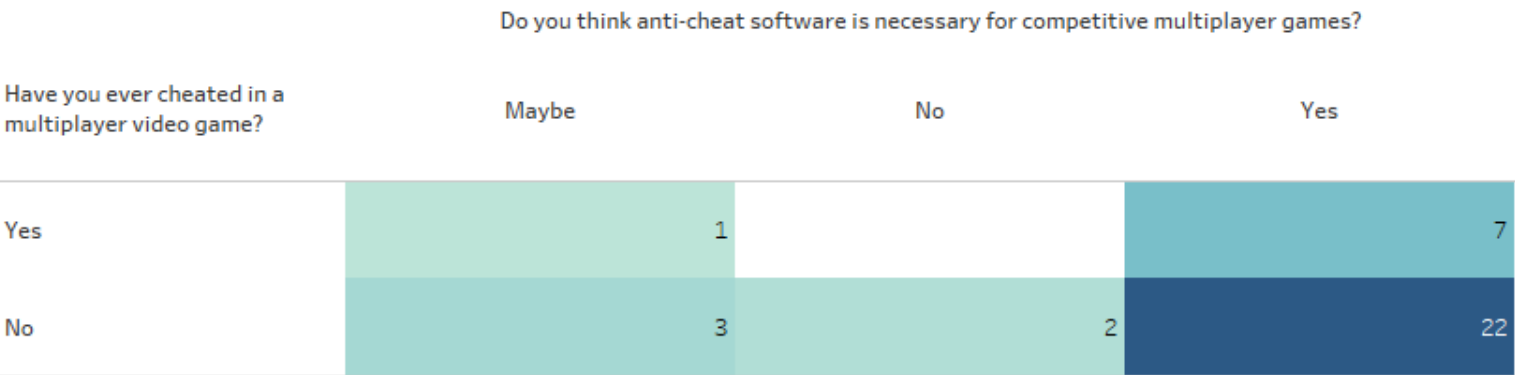
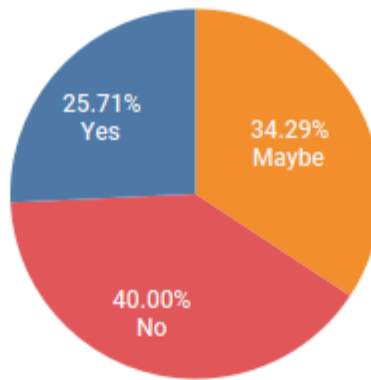


Figure 3.11 - Players that have cheated showing desire for anti-cheat

Another interesting and somewhat unexpected piece of information was uncovered by this survey, that the overwhelming majority of players that admitted to cheating still believe that anti-cheat is necessary in competitive multiplayer video games, as shown in Figure 3.11.

Considering the information from Figure 3.9, it is possible that these cheating players would prefer not to have been cheated against even if it meant that they could not cheat themselves.

Is cheating a common occurrence when you are playing video games?

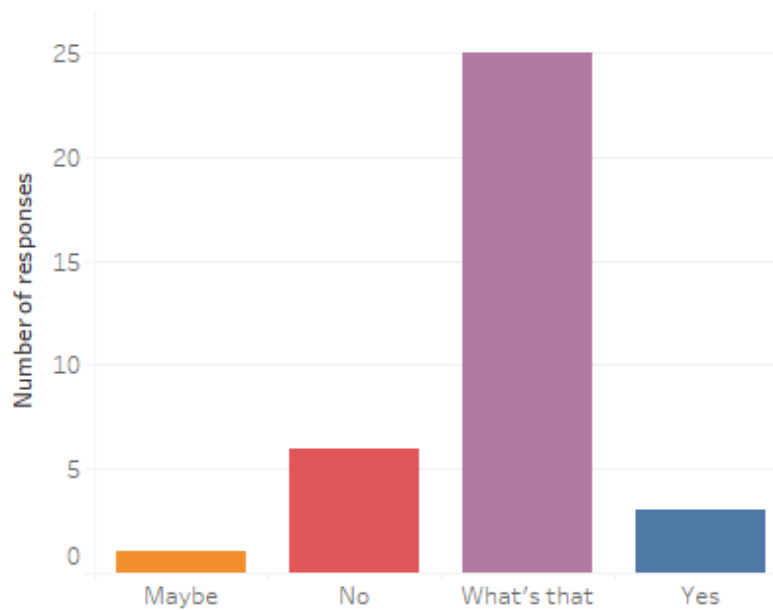


*Figure 3.12 - Eighth question on the survey and a visualisation of the responses*

Figure 3.12 shows the varying answers of respondents on whether cheating is a common occurrence when they play video games. 14 respondents (40%) answered “No”, and the remaining 21 respondents (60%) answered “Maybe” or “Yes”.

This is valuable information as it shows that the majority of respondents are frequently either not sure that they are playing against legitimate players or are confident that they are playing against cheating players. It also has to be considered that some of those in the “No” category do not primarily play games with a multiplayer focus, and as such will not encounter any potential cheating.

### Do you take issue with kernel level / kernel mode anti cheat?



*Figure 3.13 - Ninth and final question on the survey and a visualisation of the responses*

Figure 3.13 shows that 25 respondents (71.43%) did not know what a kernel level anti-cheat is. The other 10 respondents (28.57%) are divided. Only 3 respondents (8.57%) answered that they take issue with kernel level anti-cheat, whilst 6 respondents (17.14%) do not have a problem with it.

The last 1 respondent answered “Maybe”, this is possibly indicating that they take issue with it depending on the company that is running it, or how it is implemented.

Some kernel level anti-cheats such as the aforementioned Vanguard stays on even when you are not playing the game which raises privacy, security, and performance concerns. In the past, kernel level anti-cheats have been abused by malicious developers for their own gain, an anti-cheat developed by ESEA was found to have been secretly mining bitcoins (Savage, 2013). These reasons give credence to distrust of this type of software even for legitimate players.

The counter to this argument is that without kernel level anti-cheat, the fight against cheaters is already a lost battle, circumventing userland anti-cheat is trivial in comparison. Additionally, kernel level access is not at all required to run malicious code, malware has been infecting computers with userland permissions for decades (vmcall, 2020a).

The conclusion from this question is that a significant number of gamers do not know what a kernel level anti-cheat is, and the majority of those that do know, do not take issue with it.

### 3.3 Requirements modelling

After the requirements gathering phase, requirements modelling is the next step. Here the data is compiled into concrete requirements.

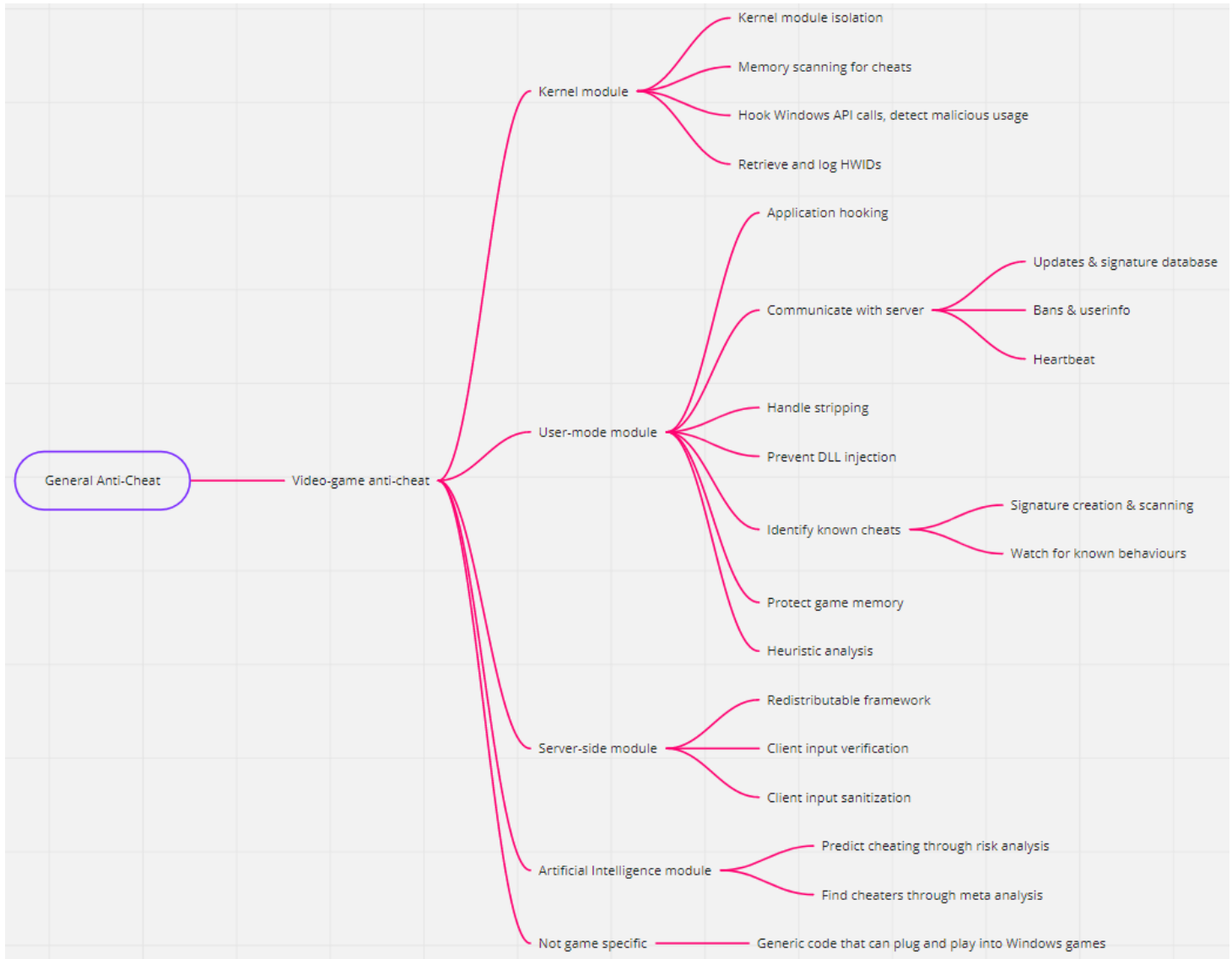


Figure 3.14 - Early mind map of functional possibilities

In order to produce a list of requirements, a mind map was created with a collection of possible functionalities for the application to have, shown in Figure 3.14.

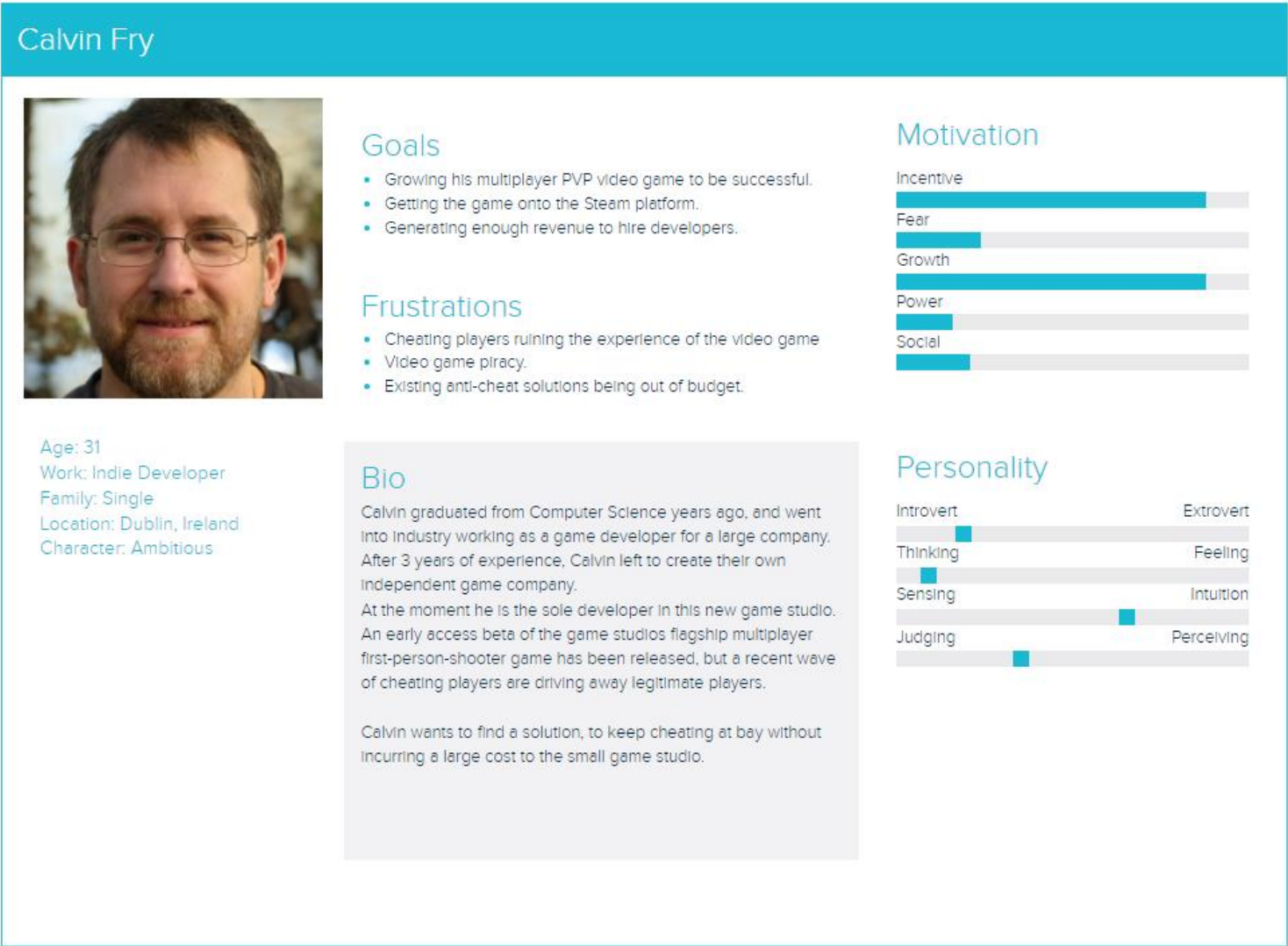


Figure 3.15 - Calvin Fry persona

The persona shown in Figure 3.15 shows an example of a solo developer that is having trouble with cheaters in their game. Calvin would have good reason to use this software as it is targeted toward multiplayer video games and could reduce the impact of the cheating problem with low performance and monetary cost.

## Turner Cortez



Age: 28  
Work: Game developer  
Family: Engaged  
Location: Dublin, Ireland  
Character: Caring, Enthusiastic,  
Problem solver

### Goals

- Grow with their game studio to have a profitable company.
- Become a well known game developer.
- Be well respected amongst his peers.

### Frustrations

- His work being ruined by the actions of others.
- Low pay of current job while trying to start a family.
- Existing anti-cheat solutions being out of budget.

### Bio

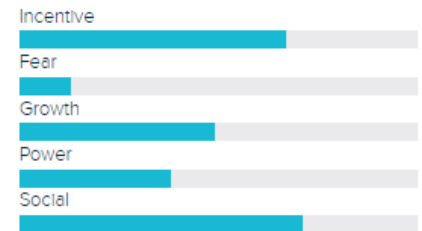
Turner received certifications for game development through nightly lectures. He began working for a game studio startup with 4 other developers.

Turner and his co-workers are creating an online murder mystery game that is rapidly gaining traction. The company was not ready for the large spike in player base, and now security has become a real problem.

Cheating software for the game was released for free on an online forum, and now there are a wave of players cheating, driving away legitimate players and threatening to kill the game's success.

Turner is investigating solutions to the problem, looking for a quick way to reduce cheating in the game before it does too much damage.

### Motivation



### Personality

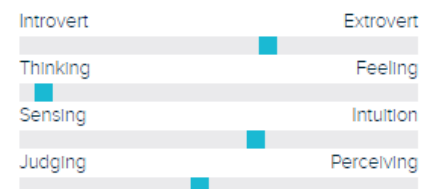


Figure 3.16 - Turner Cortez

The persona shown in Figure 3.16 demonstrates another case, where this game developer instead works for a small start-up game studio. This start-up also has a tight budget, and commercial anti-cheats are too heavy-duty for the requirements for this game. They want the game to be able to run on old hardware with low processing power, and many commercially available anti-cheats incur large performance hits.

### 3.3.2 Functional requirements

1. Offers protection against primitive and well-known methods of injecting code into a program.
2. Signature scanning to identify cheat software even when it has been modified.
3. Communication with a control server, with a heartbeat system and reporting cheating behaviour.
4. Sending samples back to control server for manual inspection.
5. Identification system to identify repeat offenders.
6. Heuristic analysis module to detect new cheats.
7. Enforcing game file integrity, disallowing players from playing with a statically modified game.
8. Modularise the anti-cheat system so different games with different requirements can use different features.

### 3.3.3 Non-functional requirements

1. Profiling and optimization to increase the performance of the anti-cheat.
2. A kernel level module to increase the effectiveness of the anti-cheat.
3. Partial packing / obfuscation to make it more difficult to bypass the anti-cheat.
4. A network module to prevent packet manipulation and packet-based attacks (e.g., replay attacks).
5. Server-side framework to detect blatant cheating (e.g., teleportation cheats).
6. Other minor functionalities to prevent cheating such as handle stripping, hooking functions commonly used for cheating etc.

### 3.4 Feasibility

This application will be developed primarily in modern C++. Modern C++ is a loose term that typically refers to C++11 (released in 2011) and standards after that. Before C++11 the last ISO standard version was C++03 which was only a minor change from C++98 (Meyers & O'Reilly Media, 2018, pp. 1–2).

The style of code and best practices between C++98 and post-C++11 are radically different, so the term modern C++ and legacy C++ separate them. The specific version of C++ this project will be produced in is C++20, which is the most recent stable standard available. C++20 recently reached widespread compiler support (cppreference, 2022; Microsoft, 2021d).

Visual Studio 2022 community edition, an IDE by Microsoft, will be used to develop and test the code for the application. A plugin called ReSharper C++ by JetBrains will be used to provide help aligning with best practices of modern C++ and preventing bugs.

It will be feasible to produce a userland anti-cheat that prevents primitive attempts at injecting code into a video game, as well as protecting file integrity and identifying known cheats. It will also be feasible to communicate information to a control server.

More complex features such as intelligent heuristic scanning, a kernel level module, and a network anti-cheat module may be unfeasible due to time constraints. It's important to note that existing anti-cheat software solutions were developed by companies with millions of dollars of spending over years with professional development and research teams. Since the scope of this project is much smaller, the end product will not be of comparable quality to those solutions. Instead, this project will aim to provide fundamental protection for otherwise unprotected games.

Kernel-level development is difficult and tedious; much of the standard library of C++ cannot be used in kernel mode, meaning you can only use core language concepts. There are a number of other reasons that make development of kernel-level code in C++ a tricky task, as some C++ features and constructs produce incompatible or suboptimal code. Due to this, Microsoft recommends developing in ANSI C (Microsoft, 2021f).

To develop a kernel mode driver for Windows that can be easily distributed to users, the driver has to be signed. Signing a driver is a difficult and expensive task, the process consists of sending details and software to a certificate signing company that is partnered with Microsoft (Microsoft, 2021e). These companies will review your software and certify the authenticity of your program. It costs over €400 per year to have a driver signed (DigiCert, 2021).



The workaround for this is to develop the software with the “TESTSIGNING” mode on Windows enabled. This mode allows the system to load unsigned drivers, this is purely for use in development (Microsoft, 2022). Unsigned drivers that would be produced for this project would therefore not be distributable to users. However, this does not completely rule out the feasibility of creating a kernel mode driver, it could be produced in code but the fee for the code signing would be the responsibility of the game developers that wish to use that module.

Another major issue for the feasibility of this application is that resources are obscured on this topic. Information is difficult to find on this topic, especially surrounding the development or implementation of anti-cheats. The companies that develop anti-cheats purposely make it as difficult as possible to discover the techniques used to catch cheaters, this is done to prevent cheaters finding a way to bypass it. These anti-cheat companies typically heavily obfuscate their code and employ other features such as packing and streaming the anti-cheat’s modules over the internet.

This leaves the majority of the documentation to be sourced from the other side, the hackers that reverse engineer and document their findings publicly. Online communities exist that discuss and demonstrate the methods that anti-cheats use to prevent cheating, with the purpose of developing methods to circumvent it. Their documentation is very useful in understanding how an anti-cheat system operates. To produce this application, specific information on how to create modules will therefore have to be sourced from the third-party documentation of many different independent hackers.

To conclude this section, the project is feasible. However, some features that exist in other anti-cheat software may be beyond the scope of this application. In order to make this project work, it will be produced modularly. The core logic of the anti-cheat will be developed first, and the additional modules will be an add-on to that. In the case that modules do not make it into the final application, the final application will still contain the functionality of all the modules that did.

### 3.5 Conclusion

This chapter has reviewed the existing market, providing an analysis for various existing solutions to the cheating problem. From this research, the advantages, and disadvantages of different approaches to preventing cheating has been outlined.

// todo discuss interviews

The need for this application to exist has been demonstrated by the results from the survey. The opinions of gamers on anti-cheat software has been discussed in detail, and conclusions drawn from the results.

The requirements modelling portion of this chapter has covered the functional requirements that the application will need to have, as well as the non-functional requirements that the application would ideally have.

The personas showed the people and problems that fit the ideal use case for this software. The software can be developed with these personas in mind, targeting the development to cater to their needs.

Lastly, the feasibility of the project has been discussed. This outlined the difficulties and workarounds required to create this application.

In conclusion this chapter has solidified the direction for this application by outlining the purpose and scope of it. The goals of this project are now more clearly defined, and work can begin on specific requirements now that this preliminary research has verified its feasibility and its purpose.

## 4 Design

### 4.1 Introduction

This chapter will outline and describe the anticipated design and architecture of Quack. This project has no major plans for a user interface, the primary focus of this chapter will therefore be on the software design.

The software design plan bridges the gap between the requirements phase and the subsequent implementation phase, this design chapter will describe the design required for features outlined in the requirements chapter.

### 4.2 Application design

From a high-level, some basic requirements will need to factor into the design. The anti-cheat will need to be easily redistributable, if the file size of the final implementation was large, it may dissuade game developers from incorporating it into their game, or clients from playing it. Executable code takes up a small amount of space compared to media such as images and video, however large library dependencies can inflate the file size.

This design aspect is particularly important when taken into context of other previously discussed security measures that may be implemented. If the final application is packed, the large file size will hamper the effectiveness and speed of unpacking. If the modules of the anti-cheat are streamed dynamically over the internet rather than statically located on the computer, a large file size would make this unviable as it would have the undesirable effect of absorbing a portion of the end-user's bandwidth during download on each start of the protected video game.

It is therefore very important to take these fundamental software design decisions into account early, as overlooking a key detail at this stage could potentially limit the capability and usefulness of the final product.

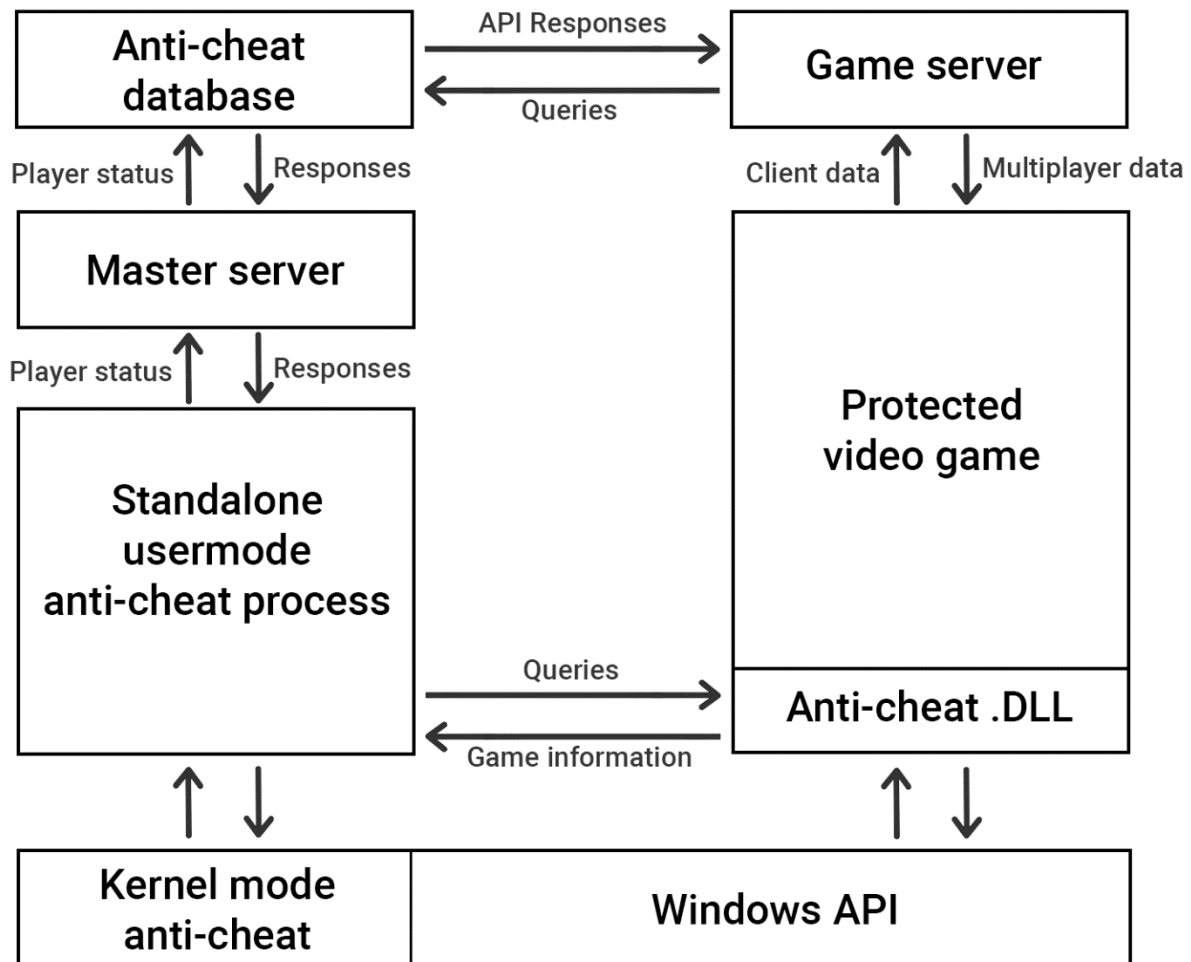


Figure 4.1 - Application architecture for Quack

Figure 4.1 displays a high-level architectural overview of the applications planned design. The arrows in this diagram represent communication between the separate components. The K in QUACK stands for Kit, the reason for this becomes clear from the diagram shown in Figure 4.1, the software is actually a collection of independent components working together to achieve a common goal.

The "Protected video game" and "Game server" components in this diagram are placeholders for the multiplayer video game that Quack would protect. The "Windows API" component is also not part of Quack, in this diagram it represents communication with the Windows operating system on a client.

#### 4.2.1 Technologies

Visual Studio 2022 community edition by Microsoft with the JetBrains ReSharper C++ plugin. This is the Interactive Development Environment (IDE) being used, it was chosen as it is especially suited for Windows development. The compiler and build tools that come with Visual Studio are especially suited for Windows development, which is the target platform for this project. The ReSharper plugin is an assistant that aids with modern C++ best practices and general code design issues, it is essentially a feature rich in-depth C++ linter.

The main alternative to this is CLion, which was considered as well. However, CLion relies on an antiquated and complex build system software called CMake, it enables cross-platform build support. This build system isn't needed as this project is not cross-platform. Although it is possible to configure CLion to work without CMake, it makes more sense to use Visual Studio which is designed for Windows development from the start, and uses the feature-rich Microsoft Visual C++ compiler (MSVC).

The IDE Code::Blocks and code editor Visual Studio Code were also considered; however, both of these options are missing a lot of features when compared to the aforementioned IDEs. The compilers available outside of MSVC are either ported from Linux or are lacking substantial feature support for the latest C++ features (cppreference, 2022).

A core technology to this project is C++20. This major update to C++ is still not in a fully stable state, however a large stable subset of it exists in Visual Studio, which is what will be used for this project (Microsoft, 2021d). Effort will be made to keep the project to modern C++ best practice guidelines, which offers better memory safety guarantees, faster runtimes, and more maintainable code among other benefits (ISOCPP, 2022).

Node.js is being used to develop proof-of-concept server-side handling for the anti-cheat. Although it's not part of the anti-cheat software itself, it will exist to provide an example implementation that a real game studio could set up to interface with the anti-cheat. It will act as a master server, processing information sent by all the client's local anti-cheat.

// todo mention virtual machine for kernel testing, a game?, (cheatengine, reclass, IDA pro)

#### 4.2.2 Design pattern and paradigm

This application will be created under the module design pattern. This system of modular design is important for this program, as different modules of the anti-cheat may be required for different games. For example, some games may need kernel level anti-cheat, whilst others do not. Therefore, to meet the needs of all games, functionality of the anti-cheat needs to be detachable without issue.

Functionality of the application will be created as loosely coupled modules. The ideal resulting application will allow modules to be removed or inserted without needing major refactoring. This modular design also has other benefits, the separation of concerns design principle reduces complexity of the final application. It prevents code that is heavily intertwined with jumping control flow (Alexandru Telea et al., 2009).

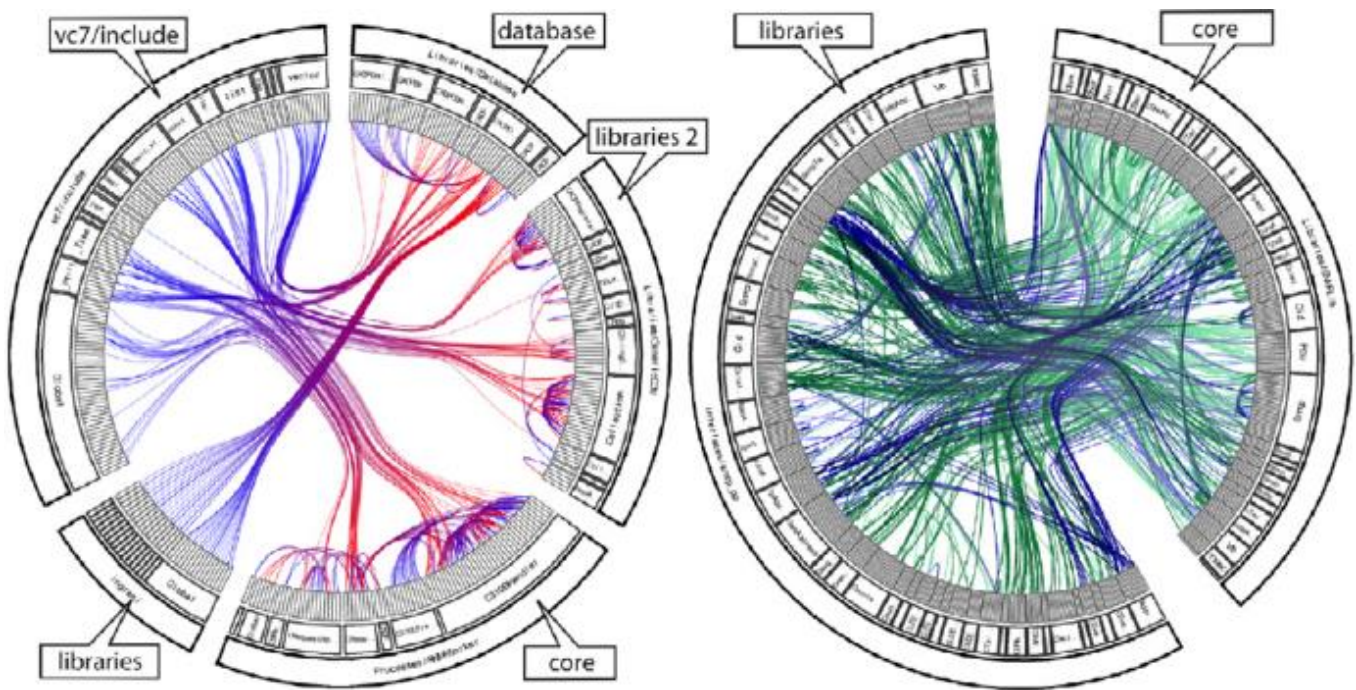


Figure 4.2 - Modular (left) vs unstructured (right) (Alexandru Telea et al., 2009)

Figure 4.2 shows the difference in code structure between modular design and unstructured code bases. These are two dependency graphs of separate projects. The middle lines inside the circles that are coloured show the dependencies inside the code.

It can be seen from the visualisation the complexity of the unstructured code base. Since code freely references other sections of code, it would be very difficult to remove a section from this code-base, as any number of other sections may rely on it.

The left graph demonstrates a code base that enforces loose coupling, whereas the right graph has tight coupling between components.

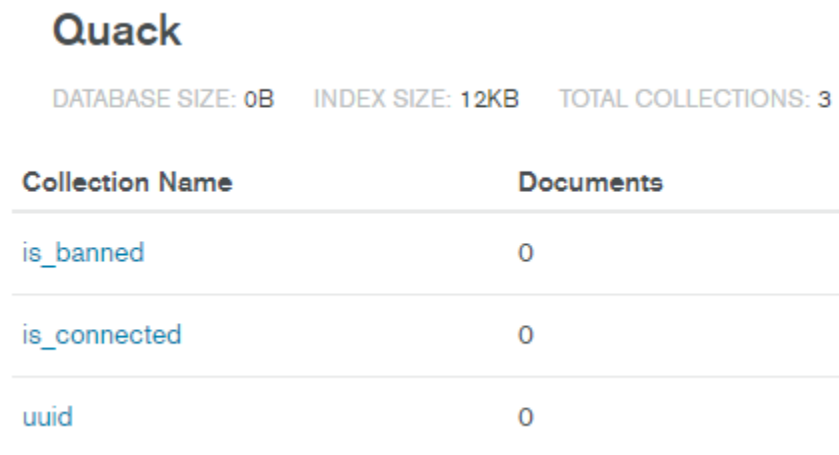
The primary programming paradigm for this project will be Object-Oriented. This has great support for modular design and separation of concerns. C++ also has fantastic support for the Object-Oriented paradigm, which has been a core concept of the language since its inception (Stroustrup, 1988).

C++ is a multi-paradigm language, and it is optimal when used in a multi-paradigm way. It will therefore use Object-Oriented programming where it is sensible to do so, and not when there is a better alternative for certain functionality.

### 4.2.3 Database design

Databases for this application will be implemented by the consumers of this software. As different games will have different database needs, they may incorporate the anti-cheat information in different ways. Particularly if the game already has a database system, they may wish to merge the anti-cheat information with the existing table.

A database will be produced as a proof-of-concept for this application. It will be a simple database that represents the functionality that could be easily implemented for a real game.



The screenshot shows the MongoDB interface for a database named 'Quack'. At the top, it displays 'DATABASE SIZE: 0B', 'INDEX SIZE: 12KB', and 'TOTAL COLLECTIONS: 3'. Below this is a table with two columns: 'Collection Name' and 'Documents'. The table lists three collections: 'is\_banned' with 0 documents, 'is\_connected' with 0 documents, and 'uuid' with 0 documents.

Collection Name	Documents
is_banned	0
is_connected	0
uuid	0

Figure 4.3 - Proof-of-concept database on MongoDB

The proof-of-concept database shown in Figure 4.3 was created using MongoDB. It contains a way to uniquely identify a player, and two Boolean values that dictate whether the player is banned and if they are connected. The `is_banned` Boolean here will be set to true if a player has been caught cheating, and false otherwise.

The `is_connected` Boolean will be true if the player has the anti-cheat running on their computer, and false otherwise. A video game server could query this database to see if it should allow a player to connect.



#### 4.2.4 Visual design

This application, like other anti-cheats, will have a small visual presence. Anti-cheat software works in the background, it maintains a low visual profile. As such it will have no user interface. The only major visual aspect that anti-cheats have is the icon of the application.



*Figure 4.4 - Easy Anti-Cheat's logo (Easy Anti-Cheat, 2022b)*



*Figure 4.5 - BattlEye's logo (BattlEye, 2022)*

As shown in Figure 4.4 and Figure 4.5, existing major anti-cheats employ an icon that has a striking visual factor. Both are clip-art style with an angry appearance.

Quack will have an icon in a similar style to these, depicting an angry duck. The angry duck icon will represent the anti-cheat and be its major visual presence. A duck was chosen because of the applications name, Quack.

A professional artist, Alex Berkeley (Berkeley, 2022), was commissioned to create the logo for this application, given the previously outlined desired outcome.

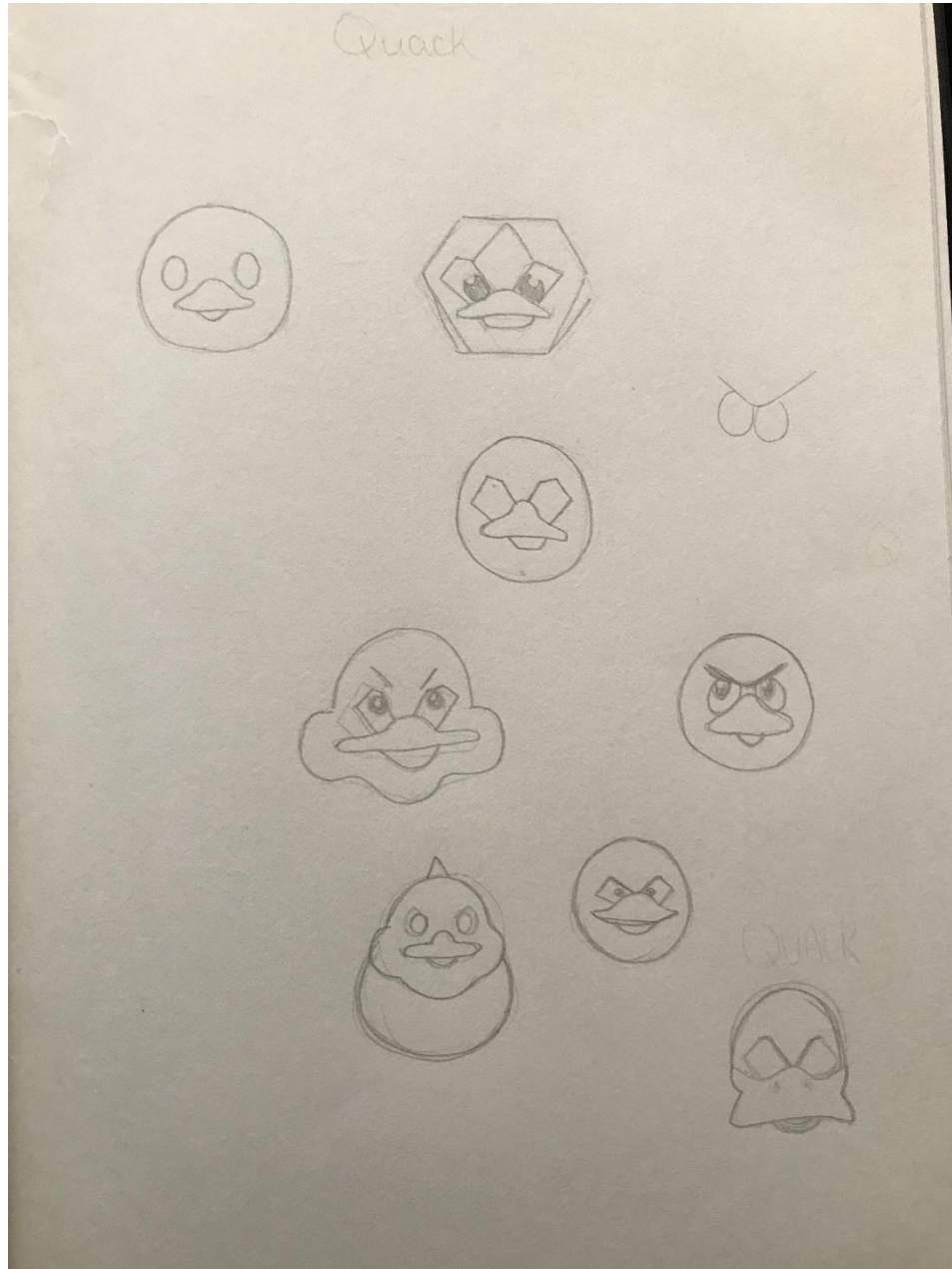
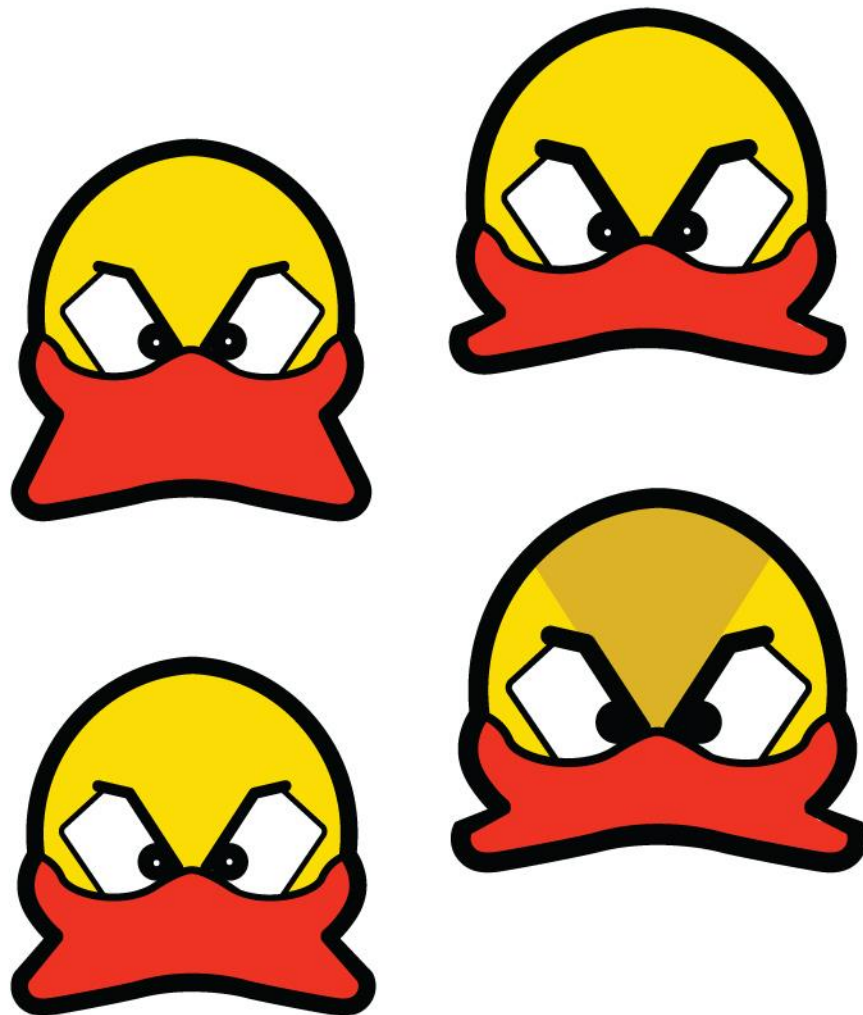


Figure 4.6 - Initial rough sketches of logo design

Figure 4.6 shows the initial rough design sketches created by the professional artist for the possible logo designs. The bottom right design shown in the figure was chosen for further development and improvement.



*Figure 4.7 - Digital prototypes of Quack logo design*

The designs in Figure 4.7 were developed after the initial paper prototypes. These digital prototypes explored additional possible styles for the final logo.



*Figure 4.8 - Final Quack logos*

These final logos shown in Figure 4.8 were created by the artist. They are slight variations of the same design. The top right icon in this set was chosen as the final design as the nares on the beak made it more apparent what the animal is at a glance.

The design captures the intended emotion of being angry, whilst still remaining a cartoon and playful. It's not taking itself too seriously but maintains the striking visual appearance of the other anti-cheat companies.

The colours were specifically chosen for this icon to be bright and striking, this is why a yellow duck is chosen over any other colour. The heavy black outline makes it stand out even on white surfaces, which yellow risks fading into.

The design had to be simplistic as it will be used as an application icon, which can occupy as little as 256 pixels of space, heavily detailed icons would risk becoming illegible at this resolution.

## 4.3 Process design

// todo I'll come back to this section when I have coded more, as I can generate more concrete diagrams and charts – This would be very difficult to plan in advance due to the fact this project will be very heavy on code content.

## 4.4 Conclusion

This chapter has reviewed the planned design of application. From the research of this chapter, the application design of the process is well described. Work on the application can now begin, with these design concepts in mind.

The technologies required to develop this project have been outlined, and alternatives to these tools have been examined in detail to ensure that they are the best option available.

This chapter would be highly useful to a developer in the future, as it describes the thinking behind the design choices for this application. If this project would require additional developers, the required tools and technology experience would be outlined by this chapter.

This program does not have a major visual presence or any user interface design requirements. The one important visual concern of this program is the logo, which has now been created and the reasoning behind its design has been described.

// todo Talk about process design section here

In conclusion this chapter has determined the design requirements for this project. It has described what is in the scope of this application and what is not.

## 5 Implementation

### 5.1 Introduction

This chapter will describe the implementation phase of Quack. This is the phase where the software is created, and preliminary testing is done. When code is added to components of Quack, it is tested before being pushed to a release build, this gives a stable foundation of code to continue working on.

### 5.2 Development environment

“Microsoft Visual Studio 2022 community edition” was the primary IDE used to develop Quack. This IDE has great support for C++ and Windows development, and the tools available in it are essential for painless debugging. It is possible to debug .DLL modules inside other processes with the tools available in this IDE. This is a niche feature that is crucial for the development of Quack, as a large portion of the code is inside a .DLL with executable code. Visual Studio Code, a similarly named but distinctly different development environment was used for the Quack-server software as that software is written in JavaScript.

Quack is made up of several modules that are built separate but rely on each other. This introduces several considerations when setting up a development environment for the project. An individual Visual Studio 2022 solution was created for each of the components that make up Quack, then these components were added to a main solution so that they can be managed and built from the same instance of Visual Studio.

The individual components were standardised in code style and layout to make the project less complex to traverse. Components that directly relate to functionality of the anti-cheat are prefixed with “Quack-“, such as “Quack-internal” and “Quack-client”.

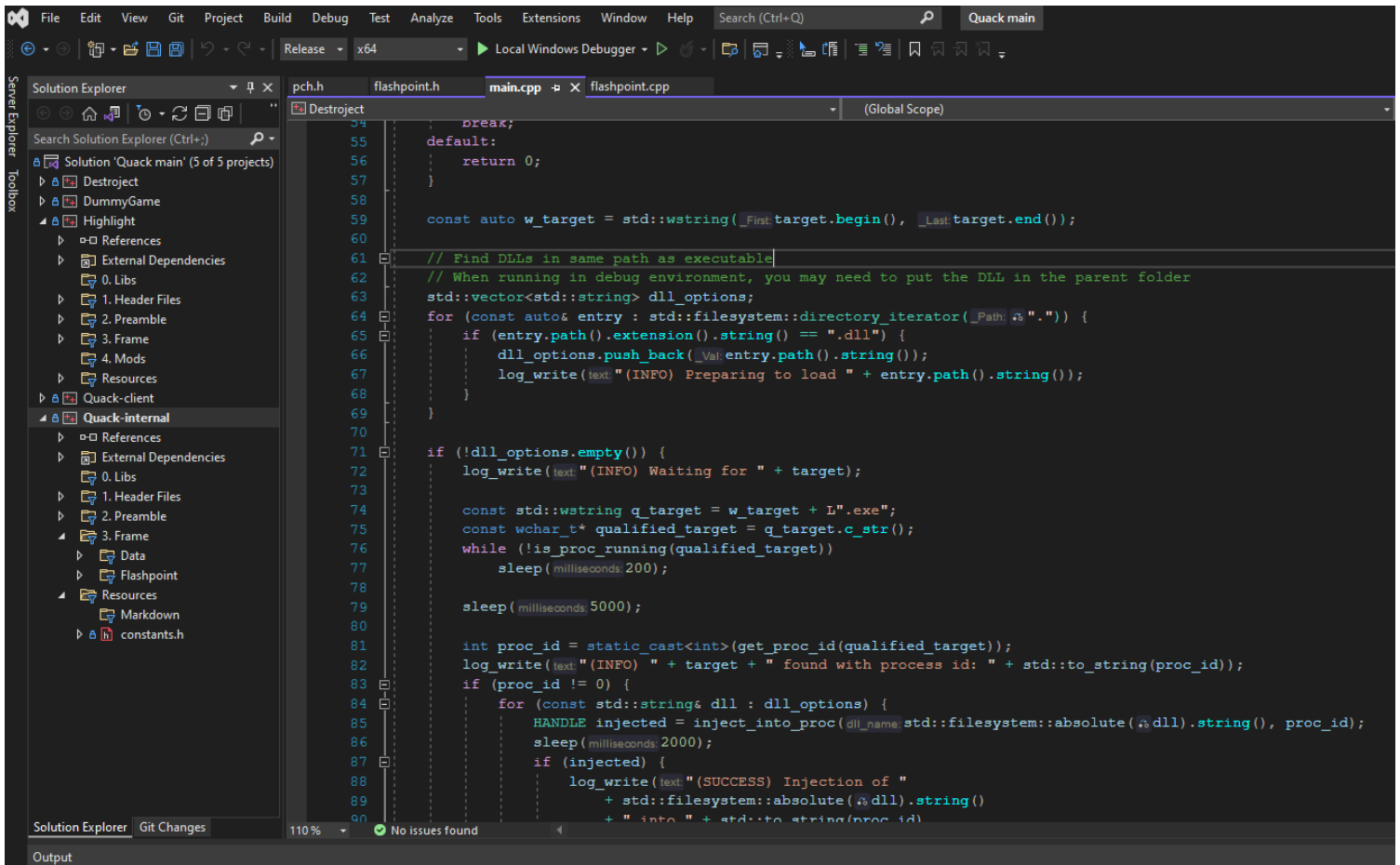


Figure 5.1 – Quack's main solution folder open in Visual Studio (Screenshot of alpha version)

The left-most panel in Figure 5.1 shows the individual components that make up Quack, including the testing software DummyGame and the red teaming software Destroject and Highlight.

Git bash linked to remote GitHub repositories was used for version control, this allows easy rollbacks when there are mistakes, and provides an easy platform for code sharing between devices. There are 4 key repositories related to this project:

- Main repository: <https://github.com/JonathanBerkeley/Quack>
- Server software: <https://github.com/JonathanBerkeley/Quack-server>
- Cheat injector: <https://github.com/JonathanBerkeley/Destroject>
- Cheat code: <https://github.com/JonathanBerkeley/Highlight-Generic/>



### 5.3 Agile development and sprints

This project was developed with agile software development principles. Every two weeks there was a meeting to determine progress and future development plans with a supervisor. There was also additional interim presentations showing the full progress of the project and plans with a second supervisor. This section will briefly outline the progress made in each sprint from a high level, though it should be acknowledged that due to the large volume of work this will be merely an overview. For a more comprehensive view, check the commit history of the GitHub repositories.

#### 5.3.1 Research

Before the first sprint, there was a large amount of research. This projects topic area is very niche, and the amount of learning required is high. Prior to beginning the sprints, there was a long period of research into the feasibility of the project, as well as learning techniques used within these secretive industries.

#### 5.3.2 First sprint

The first sprint was focused on red teaming, this previously discussed concept is where you play the part of the opposition. It focused on creating the hack tools that could be used to test the effectiveness of the anti-cheat. The initial development and functionality of “Destroject” and “Highlight-Generic” was created during this sprint.

Their repositories can be found here:

<https://github.com/JonathanBerkeley/Quack/tree/main/Red%20team>

The initial software design of the project was schemed during this sprint. This sprint also involved many menial setup tasks such as setting up the Kanban board / GitHub repositories / Visual Studio solutions and more.

#### 5.3.3 Second sprint

This sprint involved the creation of architecture diagrams that outline the planned structure of the project. The project was updated to have formatted console output during debug mode, so that the software can provide feedback on current activities while running. Heartbeat functionality was added, so that the software can communicate with a remote server to verify that it is still alive.

The “Quack-server” repository was created during this sprint, this NodeJS server acts as a potential implementation of a remote authoritative server for the anti-cheat to send information to. An Insomnia collection was created alongside this, as well as the functionality to receive and process heartbeat information from the anti-cheat clients.

#### 5.3.4 Third sprint

In order to test detection functionality of the anti-cheat, a few pieces of software had to be developed first, this was the focus of the third sprint. The injector “Destroject” was heavily updated, including new functionality to manually map cheat code into the target process. This injector now contained some of the same functionality from real-world injectors. The software was heavily documented and then partially released to the public during this sprint, where it received over 150 downloads. Although the primary focus of this project is on the anti-cheat, the supporting software such as the injector is also part of it.

The main Git repository was restructured to contain all of the various modules that the anti-cheat is comprised of in the one root repository.

The domain <https://quack.ac> was purchased and configured during this sprint, with plans to use it for the anti-cheat.

The “DummyGame” videogame placeholder was set up during this sprint. This mock DirectX9 game is what would be used for rapidly testing anti-cheat functionality, it is composed mostly of code from the ImGui library, to create a basic rendering window and some interactable boxes and performance information.

Finally, the “Quack-internal” module was created, this .DLL based anti-cheat component is to be loaded into the protected videogame, with the advantage being that memory scans can be done efficiently and easily without needing repeated calls to the WinAPI’s OpenProcess, ReadProcessMemory and WriteProcessMemory functions.

#### 5.3.5 Fourth sprint

During this sprint the logo for Quack was completed and incorporated into the anti-cheat. Internal multithreaded communication between different modules of the anti-cheat was developed, allowing different modules of the anti-cheat to send messages to each other without hindering the main thread of execution.

The most important part of this sprint was the development of the signature scanning functionality. This functionality scans the game’s memory for known cheats. It does this by searching for byte patterns in memory that are known to belong to cheats. When a byte pattern is found that is known to be present in cheat software functionality, it’s proof that the player is cheating.

// todo DNS scanning writeup

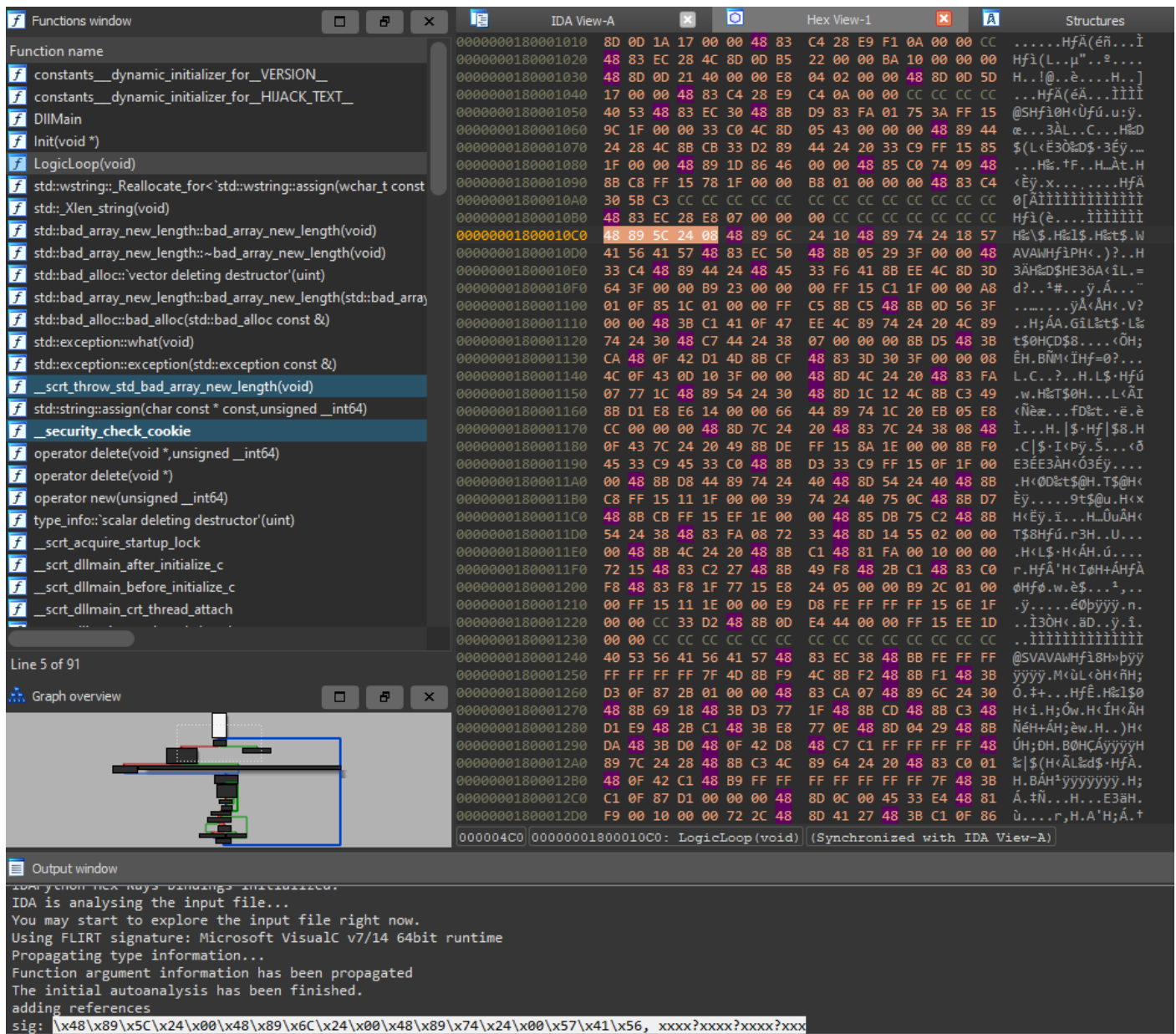


Figure 5.2 - View of cheat data, making signatures from its hexadecimal patterns

Figure 5.2 shows an example of the process of retrieving signatures from cheat samples. By reverse engineering the cheat, identifiable sections can be extracted and made into a pattern that the anti-cheat can scan for.

### 5.3.6 Fifth sprint

The “Quack-server” section of the software was updated to connect to a MongoDB instance and update it with information about players that are caught cheating. This piece of functionality completes the full chain from the initial detection of cheating to the storing of that data in a remote database.

The “Quack-internal” module was updated to verify if modules that were loaded into the videogame were signed by a valid signature. This optional functionality checks that the .DLL modules that are loaded in the game are signed. This could be used alongside a blacklist / whitelist system to make naïve code injection attempts more difficult.

The internal heartbeat system was also improved with large improvements to performance by making more of the code multithreaded.

A task dispatch system was developed that queries CPU load before deciding to launch a potentially expensive anti-cheat action, such as a memory scan. This system decides what tasks to run and when to run them based on what load the CPU is under, preventing slowdowns and keeping the core idea of creating a fast anti-cheat.

A testbed project was set up to speed up the new feature development process, rather than recompile the project and run all the modules each time the functionality needs testing, a testbed project was made that isolates new features until they are stable and functional enough to be added to the main project.

A large portion of this sprint was also spent creating Doxygen documentation in the C++ code. As well as refactoring and improving existing code.

### 5.3.7 Sixth sprint

The sixth sprint focused on identification of previous cheaters. If a player has previously been caught cheating, they should be banned even if they decide to create a new account or buy a new copy of the game. To achieve this, a system to identify previously seen users was created.

Identification across computers can be made by querying information from the Address Resolution Protocol table in Windows. This table contains information about known external computers in the same network as the computer, including the router and any connected devices with networking enabled. This information contains MAC addresses for the networked devices, these can then be used as unique identifiers that represent the network (Microsoft, 2021a).

For privacy reasons, the MAC addresses are converted to SHA256 hashes using a C++ hashing library (<https://github.com/stbrumme/hash-library>). These hashes then can be used to identify networks that cheating players use.

Now when a player launches the game, the devices on their network can be checked against known cheater network hashes. If there is a match, it's a high-risk indicator of a repeat offender. It's worth noting, however, that if there's a legitimate player and a cheating player on the same network it will flag the legitimate player as well, therefore detections should be manually reviewed if detected by this system.

Like all other major features in the anti-cheat, it can be easily disabled if the specific game implementing it doesn't require or desire such functionality.

#### 5.3.8 Seventh sprint

The seventh sprint tackled a problem whereby a cheater could circumnavigate detection by manually mapping the cheat code into the game, rather than loading it as a module through the WinAPI.

This method of loading a cheat was included in the Destroject injector as part of this project. The functionality for this is much more complex than regular loading of a module. Instead of calling the appropriate WinAPI functions to load the module, the loading logic is manually recreated, and the memory is mapped in. Doing this has the advantage to a cheater that manually mapped cheats do not appear in the module list, and their code thus cannot as easily be found. Windows has no way to tell that a cheat module is loaded when it is done this way.

The method to detect this was implemented in this sprint. The method of detection is to enumerate the executable pages of process memory of the game. The same method of signature scanning is then used to determine if known cheat code is present in these executable regions. The cheat code will have to have been mapped to an executable region of memory, as it contains executable code. This method will be slower than enumerating modules but will catch a wider range of cheats.

A real-world cheat was created for the game "Inertia", this was done to better demonstrate the use of an anti-cheat. The cheat created for this multiplayer game sets health and ammunition to maximum and prevents them from decreasing. This makes the cheater invincible to legitimate players. This cheat can be used for demonstration purposes, in explaining the issue with cheating and what need this project fulfils. The repository for this cheat software was added to the red team subdirectory as a git submodule (<https://github.com/JonathanBerkeley/Inertia-cheat>).

Additionally updated in this sprint is checking of external processes, the anti-cheat will now enumerate other processes and kill them if the process's image name matches known cheat programs. This is the first step toward detecting external cheat applications, such as Cheat Engine.

- 5.4 Database  
// todo
- 5.5 Backend  
// todo
- 5.6 Frontend  
// todo
- 5.7 Conclusion  
// todo

## 6 Testing

// todo

## Bibliography

Alder, D. (2020). *What kernel-level anti-cheat is and why you should care*. LEVVVEL. <https://levvvel.com/what-is-kernel-level-anti-cheat-software/>

Alexandru Telea, Heorhiy Byelas, & Lucian Voinea. (2009, March). *A Framework for Reverse Engineering Large C++ Code Bases*. ResearchGate; Elsevier. [https://www.researchgate.net/publication/220369791\\_A\\_Framework\\_for\\_Reverse\\_Engineering\\_Large\\_C\\_Code\\_Bases#pf12](https://www.researchgate.net/publication/220369791_A_Framework_for_Reverse_Engineering_Large_C_Code_Bases#pf12)

Atallah, M. J., Byrant, E. D., & Stytyz, M. R. (2004, November). *A Survey of Anti-Tamper Technologies*. <https://Citeseerx.ist.psu.edu/>. <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.642.1542&rep=rep1&type=pdf>

bashandslash. (2020). *BASHandSlash.com - gaming ethics: PunkBuster Part 3/3*. Bashandslash.com. [https://web.archive.org/web/20160322133643/http://bashandslash.com/index.php?option=com\\_content&task=view&id=258&Itemid=78](https://web.archive.org/web/20160322133643/http://bashandslash.com/index.php?option=com_content&task=view&id=258&Itemid=78)

BattleEye. (2022). *BattleEye – The Anti-Cheat Gold Standard*. Battleeye.com. <https://www.battleeye.com/>

Been-Lirn Duh, H., & Hsueh Hua Chen, V. (2009). *Cheating behaviors in online gaming*. <https://www.springer.com/Gp>. [https://link.springer.com/content/pdf/10.1007%2F978-3-642-02774-1\\_61.pdf](https://link.springer.com/content/pdf/10.1007%2F978-3-642-02774-1_61.pdf)

Berkeley, A. (2022). *Irish Artist | Alex Berkeley*. Alexandrasartwork. <https://www.alexandrasartwork.com/>

Bowyer, J. B. (1982). *Cheating: Deception in war & magic, games & sports, sex & religion, business & con games, politics & espionage, art & science*. St Martin's Press. <https://books.google.ie/books?id=kTPsHAAACAAJ>

Ciholas, P., Such, J. M., Marnerides, A. K., & Utz Roedig. (2020, July 7). *Fast and Furious: Outrunning Windows Kernel Notification Routines from User-Mode*. ResearchGate; unknown. [https://www.researchgate.net/publication/342727698\\_Fast\\_and\\_Furious\\_Outrunning\\_Windows\\_Kernel\\_Notification\\_Routines\\_from\\_User-Mode](https://www.researchgate.net/publication/342727698_Fast_and_Furious_Outrunning_Windows_Kernel_Notification_Routines_from_User-Mode)

Clayton Sterling Cyre. (2021, May). *Rainbow Six Siege players stage Ubisoft boycott over anti-cheat problems*. Game Rant; GameRant. <https://gamerant.com/rainbow-six-siege-boycott/>



Consalvo, M. (2009). *Cheating: Gaining advantage in videogames*. Mit Press.

Corliss, C. (2019, December 29). *The rise and fall of cheat codes*. Game Rant; GameRant. <https://gamerant.com/rise-fall-cheat-codes/>

cppreference. (2022). *C++ compiler support - cppreference.com*. Cppreference.com. [https://en.cppreference.com/w/cpp/compiler\\_support#cpp20](https://en.cppreference.com/w/cpp/compiler_support#cpp20)

CS:GO. (2021, July 7). *Fair play guidelines*. Twitter. <https://twitter.com/csgo/status/1412560338508152833>

DeepSource. (2020). *DeepSource*. DeepSource. <https://deepsourc.io/glossary/static-analysis/>

Delfy. (2020, November 28). *TF2 exploits 2010-2020*. Wwww.youtube.com. <https://youtu.be/azXXrgNY1dE>

DigiCert. (2021). *Kernel-mode code signing certificates*. DigiCert.com. <https://www.digicert.com/dc/code-signing/kernel-mode-certificates.htm>

Dnb. (2022). *EasyAntiCheat*. Dnb.com. [https://www.dnb.com/business-directory/company-profiles.easyanticheat\\_oy.250be03e7be2b8476739b62092dbe167.html](https://www.dnb.com/business-directory/company-profiles.easyanticheat_oy.250be03e7be2b8476739b62092dbe167.html)

Easy Anti-Cheat. (2022a). *Easy Anti-Cheat*. Easy.ac. <https://www.easy.ac/en-us/partners/>

Easy Anti-Cheat. (2022b). *Easy Anti-Cheat*. Easy.ac. <https://www.easy.ac/en-us/>

Fowler, M. (2020, April 19). *Riot offers hackers up to \$100,000 reward to expose Valorant anti-cheat exploits - IGN*. IGN; IGN. <https://www.ign.com/articles/riot-offers-hackers-up-to-100000-reward-to-expose-valorant-anti-cheat-exploits>

Franceschi-Bicchierai, L. (2021). *“Warzone” Bans 100,000 Cheaters in Largest Ban Wave Yet*. Vice.com. <https://www.vice.com/en/article/k7898v/warzone-bans-100000-cheaters-in-largest-ban-wave-yet>

FTI Consulting. (2021). *The rise in isolation gaming and cheating*. Fticonsulting.com. <http://documents.jdsupra.com/330d66dc-b29c-408c-bc8a-dba30684c9c4.pdf>

gameranx. (2020). *Why Do People CHEAT In Video Games? [YouTube Video]*. In *YouTube*. [https://www.youtube.com/watch?v=w8O3\\_7mLPuQ](https://www.youtube.com/watch?v=w8O3_7mLPuQ)

Higgins, C. (2016, April 30). *Valve VAC bans users of prolific TF2 hack, catches pros in the tidal wave*. PCGamesN; PCGamesN. <https://www.pcgamesn.com/team-fortress-2/valve-vac-bans-users-of-prolific-tf2-hack-catches-pros-in-the-tidal-wave>

Irdeto. (2018a). *Irdeto global gaming survey report*. Irdeto.com. <https://resources.irdeto.com/media/irdeto-global-gaming-survey-report?page=%2Firdeto-global-gaming-survey&widget=6052143d4d28bf314681cca4>

Irdeto. (2018b, May). *New global survey: Widespread cheating in multiplayer online games frustrates consumers - Irdeto*. Irdeto. <https://irdeto.com/news/new-global-survey-widespread-cheating-in-multiplayer-online-games-frustrates-consumers/>

Irdeto. (2020). *E-book: Grand Theft Gaming - Does the video games industry have a \$29bn cheating problem?* Irdeto.com. <https://go.irdeto.com/ebook-grand-theft-online-gaming-cheating-problem/>

ISOCPP. (2022). *C++ core guidelines*. Github.io. <https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#inaims-aims>

Johnson, L. (2016, December 7). *Why is it so hard to stop cheating in videogames?* <https://www.pcgamer.com/why-is-it-so-hard-to-stop-cheating-in-videogames/>; PC Gamer. <https://www.pcgamer.com/why-is-it-so-hard-to-stop-cheating-in-videogames/>

Johnson, N. (2020, December 9). *CSGO dev says trusted accounts see few cheaters, players disagree*. WIN.gg. <https://win.gg/news/csgo-dev-says-trusted-accounts-see-few-cheaters-players-disagree/>

Kaiser, E. L., Feng, W., & Schluessler, T. (2009). *Fides: remote anomaly-based cheat detection using client emulation*. ResearchGate. [https://www.researchgate.net/publication/220269455\\_Fides\\_remote\\_anomaly-based\\_cheat\\_detection\\_using\\_client\\_emulation](https://www.researchgate.net/publication/220269455_Fides_remote_anomaly-based_cheat_detection_using_client_emulation)

Lehtonen, S. (2020). *Comparative Study of Anti-cheat Methods in Video Games*. [https://helda.helsinki.fi/bitstream/handle/10138/313587/Anti\\_cheat\\_for\\_video\\_games\\_final\\_07\\_03\\_2020.pdf?sequence=2&isAllowed=y](https://helda.helsinki.fi/bitstream/handle/10138/313587/Anti_cheat_for_video_games_final_07_03_2020.pdf?sequence=2&isAllowed=y)

Li, V. (2021, May 30). *The Anti-Cheat Arms Race - SUPERJUMP*. Medium; SUPERJUMP. <https://superjumpmagazine.com/the-anti-cheat-arms-race-c19343be1dd>

Linfo. (2005a). *Kernel mode definition*. Linfo.org. [http://www.linfo.org/kernel\\_mode.html](http://www.linfo.org/kernel_mode.html)

Linfo. (2005b). *User mode definition*. Linfo.org. [http://www.linfo.org/user\\_mode.html](http://www.linfo.org/user_mode.html)

Meyers, S., & O'Reilly Media. (2018). *Effective modern C++ : [42 specific ways to improve your use of C++11 and C++14]* (Twelfth Release, pp. 1–2). O'reilly.

Microsoft. (2021a, July 21). *Hardware ID - Windows drivers*. Microsoft.com. <https://docs.microsoft.com/en-us/windows-hardware/drivers/install/hardware-ids>

Microsoft. (2021b, August 18). *User mode and kernel mode - Windows drivers*. Microsoft.com. <https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode>

Microsoft. (2021c, September 2). *MSVC C++20 and the /std:c++20 switch - C++ team blog*. C++ Team Blog. <https://devblogs.microsoft.com/cppblog/msvc-cpp20-and-the-std-cpp20-switch/>

Microsoft. (2021d, December 15). *Kernel-mode code signing requirements - Windows drivers*. Microsoft.com. <https://docs.microsoft.com/en-us/windows-hardware/drivers/install/kernel-mode-code-signing-requirements--windows-vista-and-later->

Microsoft. (2021e, December 15). *Portable drivers - Windows drivers*. Microsoft.com. <https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/portable>

Microsoft. (2022, January 7). *Loading test signed code - Windows drivers*. Microsoft.com. <https://docs.microsoft.com/en-us/windows-hardware/drivers/install/the-testsigning-boot-configuration-option>

Mikkelsen, K. (2017). *Information security as a countermeasure against cheating in video games*. [https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2448953/18113\\_FULLTEXT.pdf?sequence=1&isAllowed=y](https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2448953/18113_FULLTEXT.pdf?sequence=1&isAllowed=y)

Minna Adel Rubio. (2020, April 16). *Valorant's mandatory Vanguard anti-cheat found to be highly invasive*. Daily Esports; Daily Esports. <https://www.dailyesports.gg/valorants-mandatory-vanguard-anti-cheat-found-to-be-highly-invasive/>

Owler. (2022a). *BattlEye top competitors or alternatives*. Owler; Owler Inc. <https://www.owler.com/company/battleye>

Owler. (2022b). *Easy top competitors or alternatives*. Owler; Owler Inc. <https://www.owler.com/company/easy17>

Paoletti, M. (2021, October 18). *VALORANT Anti-cheat: Fall 2021 Update*. Playvalorant.com; Riot Games, Inc. <https://playvalorant.com/en-us/news/game-updates/valorant-anti-cheat-fall-2021-update/>

Pochwatko, G., & Giger, J.-C. (2015, December). *Cheating in video games causes and some consequences*. Research Gate. [https://www.researchgate.net/publication/292151847\\_Cheating\\_in\\_video\\_games\\_-\\_causes\\_and\\_some\\_consequences](https://www.researchgate.net/publication/292151847_Cheating_in_video_games_-_causes_and_some_consequences)

Savage, P. (2013, May). *ESEA release malware into public client, forcing users to farm Bitcoins [updated]*. Pcgamer; PC Gamer. <https://www.pcgamer.com/esea-accidentally-release-malware-into-public-client-causing-users-to-farm-bitcoins/>

Seppala, T. J. (2016). *Valve will ban Steam cheaters via their linked phone numbers*. Engadget; Engadget. [https://www.engadget.com/2016-04-30-valve-anti-cheat-phone-number-ban.html?guccounter=1&guce\\_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLnNvbS8&guce\\_referrer\\_sig=AQAAAFoCbyA6SQAhsTNQQFezMx4eA4Mh7CTyLCOFrXUv23oJF9OYnNcTGN28TdU\\_RmTqpUcieeiD\\_RcnzJ98EnRL9fzCH9OmrH5aJmyltuMbyn9oiikV8s\\_aKN\\_2rsKAy0hpMN1ZXdbAWSB274VafZ5J1HN7PHruwTStbsrZvzdR2M-3](https://www.engadget.com/2016-04-30-valve-anti-cheat-phone-number-ban.html?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLnNvbS8&guce_referrer_sig=AQAAAFoCbyA6SQAhsTNQQFezMx4eA4Mh7CTyLCOFrXUv23oJF9OYnNcTGN28TdU_RmTqpUcieeiD_RcnzJ98EnRL9fzCH9OmrH5aJmyltuMbyn9oiikV8s_aKN_2rsKAy0hpMN1ZXdbAWSB274VafZ5J1HN7PHruwTStbsrZvzdR2M-3)

Stroustrup, B. (1988). What is object-oriented programming? *IEEE Software*, 5(3), 10–20. <https://doi.org/10.1109/52.2020>

Thorn, E. (2018, September 14). *CS:GO - How to get Prime*. Metabomb.net; Metabomb. <https://www.metabomb.net/csgo/gameplay-guides/cs-go-how-to-get-prime>

time2win. (2019). *time2win.net*. Time2win.net; time2win.net. <https://www.time2win.net/guides/hwid-ban/>

Unknowncheats. (2015). *Legit cheating guide*. UnKnoWnCheaTs - Multiplayer Game Hacking and Cheats. <https://www.unknowncheats.me/forum/counterstrike-global-offensive/143177-legit-cheating-guide.html>

Unknowncheats. (2018). *FAQ - Anti-Cheats, Bypasses and Hardware IDs*. UnKnoWnCheaTs. <https://www.unknowncheats.me/forum/anti-cheat-bypass/266433-faq-anti-cheats-bypasses-hardware-ids.html>

Unknowncheats. (2020). *Easy anti-cheat does not check IP*. UnKnoWnCheaTs - Multiplayer Game Hacking and Cheats. <https://www.unknowncheats.me/forum/anti-cheat-bypass/404466-eac-check-ip.html>

Valve Anti-Cheat team. (2016). *Reddit forums*. Reddit.com. [https://www.reddit.com/r/GlobalOffensive/comments/5u2xly/eli5\\_why\\_are\\_spinbots\\_not\\_autodetected\\_or\\_atleast/ddr7ydq/](https://www.reddit.com/r/GlobalOffensive/comments/5u2xly/eli5_why_are_spinbots_not_autodetected_or_atleast/ddr7ydq/)

Vape. (2019, March 29). *The BanWaves*. Vape Forums; Vape Forums. <https://forums.vape.gg/index.php?threads/the-banwaves.6928/>

Vaughan-Nichols, S. J. (2019, September 23). *Static vs. Dynamic IP Addresses*. Static vs. Dynamic IP Addresses; Avast. <https://www.avast.com/c-static-vs-dynamic-ip-addresses>

vmcall. (2020a, April 17). *Why anti-cheat software utilize kernel drivers*. Secret Club. <https://secret.club/2020/04/17/kernel-anticheats.html>

vmcall. (2020b, July 6). *BattlEye client emulation*. Secret Club.  
<https://secret.club/2020/07/06/bottleye.html>

Warren, T. (2021, October 13). *Call of Duty's new anti-cheat system includes a kernel-level driver to catch PC cheaters*. The Verge; The Verge.  
<https://www.theverge.com/2021/10/13/22724037/call-of-duty-ricochet-anti-cheat-system-kernel-level-driver>

Yin-Poole, W. (2021, April 14). *Activision confirms it issues hardware bans to "repeat or serial" Call of Duty: Warzone cheaters*. Eurogamer.net; Eurogamer.net.  
<https://www.eurogamer.net/articles/2021-04-14-activision-confirms-it-issues-hardware-bans-to-repeat-or-serial-call-of-duty-warzone-cheaters>