Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Jack Papaioannou / ID: 31253695
May 2023

Forensic Analysis Computer Tool For Video-Game Cheat Detection In-Game

Project supervisor: Ahmad Atamli
Second examiner: Frederic Gardes

A project report submitted for the award of Computer Science (MEng)

Abstract

This project aims to undertake in-depth research about the operation of video game cheats as well as anti-cheat systems along with the development of a tool to assist the understanding of this topic and also explore an anti-cheat strategy with a potential for future work. More specifically, dynamic link library (DLL) injection video game cheats that aim to alter game mechanics and attributes by changing the memory address values of that game. The tool provides an easy-to-use automation script for the existing library Volatility3, to help users obtain a quick and easy way to determine the existence of cheats in their system. So far, this has been accomplished in a way that is user-friendly for non-technical people, is light in terms of storage space and provides the user with clear detailed information about the script's process and the likelihood of the existence of cheats within that game process. This tool compared to existing systems, is a lightweight implementation with the ability to be integrated into existing systems or as a precaution detection scan before heavy processing and manual checking occurs. Testing resulted in detecting a strong presence of cheats 80% of the time with further testing needed for more accuracy. Future work includes integrating new detection methods for more cheating methods and creating a database of known cheat files, such as DLL file signatures and more, to identify a wider range of cheat files injected by cheaters and cheating methods used. This tool can help game developers create more secure and fair game experiences for players.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.
- I have acknowledged all sources, and identified any content taken from elsewhere.
- I have not used any resources produced by anyone else.
- I did all the work myself, or with my allocated group, and have not helped anyone else.
- The material in the report is genuine, and I have included all my data/code/designs.
- I have not submitted any part of this work for another assessment.

Acknowledgements

I would like to acknowledge the Volatility Foundation and its contributors for providing the Volatility Framework as an open-source memory extraction framework for the public to use. I also acknowledge the company Velocidex and their contributors, for the open-source memory acquisition driver WinPmem, which enabled me to acquire an easy way to generate memory dumps for study purposes. I would like to thank my supervisor Ahmad Atamli for his guidance and knowledge on this field and my project along with my family for their emotional support.

Contents

1 Problem

1.1 Introduction

Cheating in video games has been a documented case for many years dating back to 1992 with the Nintendo Entertainment System (NES) video game cheat cartridges. More specifically the Game Genie was a cheating product designed by the Codemasters company. It is a cartridge for the NES deck that would temporarily modify game data and thus change the game experience for the player [1]. Cheats have since then evolved and taken a software-based form that ranges from code injection to file injection and even memory manipulation. Those different types of cheats cause many negative effects on a game playing field that can cause it to become unfair, disrupting the game experience for all the players involved[16].

1.2 Cheats

For plenty of reasons, cheating has many negative consequences that can not only affect a user's experience and create an unfair playing field, but it can affect many more entities involved with the game, like the game suppliers or the developers and many more. Game suppliers can be impacted by lower sales of the game since cheating can lead to a more negative perception of the game, which can also cause revenue problems for the game developers and even more problems including negative publicity [16]. It is no wonder that the market overall can also be affected. A market that consists of billions of dollars of revenue every year and is predicted to increase every year as far as we know [2].

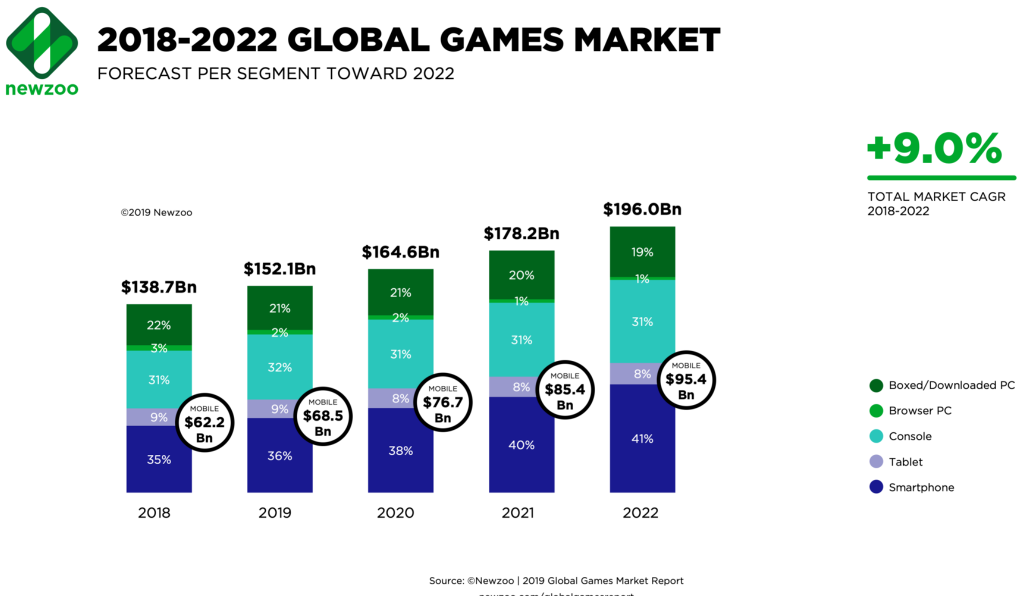


Figure 1: 2018-2022 Global Games Market Growth

Cheating has taken many forms and undergone various changes over time but the core concept for their existence has been the same, which is to alter the game experience for the player. This is the main purpose of video game cheats and it

will keep being like that for more years to come. The are many types of ways to cheat but we will discuss the most important ways to cheat, that are often used in a multitude of games.
Cheating related to virtual assets is a type of cheating where players in games that have an item system can attempt to bypass the right procedure to obtain an item. Usually, to obtain powerful items, players must spend time and effort to accomplish tasks that reward them with said items, something that not all people can or want to do. There is a range of reasons like not having the time to play the game this much or they don't have the skills necessary to accomplish those tasks and many more. As a result, players use other methods to obtain those items like purchasing them from other players or websites using real-life currency through different online marketplaces [27]. This in turn can shift the power balance between a game's player base making it hard to distinguish players that are good and players who falsify their skill by cheating their way to obtain power. Resulting in a game experience that is very unbalanced and not fun in general.
Cheating by modifying game software or data [3] like memory address values is another type of cheating that is a lot more technical than the previously mentioned one. Players in this scenario, attempt to use different cheating tools and programs, like Cheat Engine [4] or ArtMoney [5] to alter memory address values in real-time to change game values (health, ammunition), inject malicious code into the game process or inject dynamic link library (DLL) files that can change the game's values as well. This latter way of cheating will be the main focus of this project as we move further on [6].

1.3 Motivation

So why would someone want to cheat in a video game if it brings such destruction overall? There are many incentives that could lead someone to cheat in a video game and it is easier than ever today, with the enormous amount of resources and tools someone can use.
Reasons that may lead to someone cheating can include competitive motivation, the urge to win in a competitive setting, driving them to use any means necessary to achieve that victory even if that includes cheating and playing unfair [7].
Self-esteem-related reasons, provide a sense of fulfilment that increases confidence and the way cheaters look at themselves. They may seek out the thrill of adrenaline that comes from the sense of mastery or proficiency in a game, essentially deceiving themselves to change their image that may have been lowly looked at by themselves and other peers . Lastly, reasons linked to aggression, players that suffer from anti-social destructive psychological behaviour are prone to intentionally cause chaos and destruction not only in real life but to other players in-game, for the purpose of personal enjoyment.[8].
A combination of all those reasons can also lead to suggesting a reason for cheating that is not linked directly to the player but to the game's difficulty design. A player may be tempted to cheat if they are frustrated with the state of the game and its difficulty. If the game is too difficult by design for the player or the game possesses an overall player base of highly skilled individuals then it is safe to assume that the player will do everything they can to overcome that difficulty even if they have to cheat to accomplish that goal.

1.4 Anti-Cheat Tools

With all these factors that can create motivation for cheaters, what measures are set in place to combat cheating and protect the market? The solution is anti-cheats, this includes technology or software designed to detect and prevent cheating in online multiplayer games that many companies acquire either from third parties or manufactured in-house for their personal use. The main function of anti-cheats includes detecting and banning players who use cheats or apply malicious modifications to a game's process to disrupt the playing field for that game. The main goal of anti-cheat systems is to create an environment for players that is ensured to be fair without any disruptions that would otherwise cause disturbances to the playing field.
Many tools and programs to detect and combat cheaters have existed over the past years. The most notable anti-cheats include PunkBuster™, which is one of the first anti-cheat programs commercially available around the year 2000 [9]. The Valve Anti-cheat System (VAC) [10] is notorious for being a system produced by the company Valve mostly for their games built in the Valve Source engine and for some other games as well. Easy Anti-Cheat [11], is another famous anti-cheat system developed by Kamu, geared towards multiple games which was acquired in 2018 by Epic Games. Those are a couple of popular anti-cheat systems in the market right now that scan many games with a magnitude of daily players. We will focus on their core features, what they lack and how those missing attributes can be addressed. Most if not all of those systems share the fact that they operate on the kernel level, which is accomplished by including a kernel driver which allows those operations to happen, essentially giving the company that owns the tool kernel-level access to the player's computer.

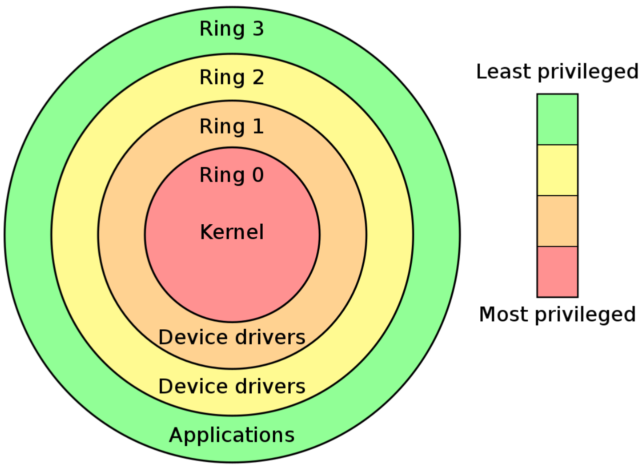


Figure 2: Privilege Rings on a x86 architecture system
(Source: Wikipedia.com)

Companies having that kind of invasive access to user systems has been a controversial topic for many years and is regarded as wrong by many people. It is wrong for a variety of reasons, the most notable one being that it brings cause for security and privacy concerns to those system owners [12].
Security wise it can be very detrimental if that access the company has can fall into the wrong hands. Attackers will obtain access to the highest level of a user's

system to act as they please. That includes access to sensitive system resources, which can be exploited by attackers to gain control of the system. Privilege escalation can be used to obtain access to inter-networked systems and resources that the system might be connected to and the list goes on with the vast array of attacks that can happen under kernel-level access on a system [12].
In terms of privacy, the aforementioned access an attacker can have led to many ways that a user's privacy can be breached. Sensitive personal data and files can be collected and accessed from a system, and monitoring of a user's activities can also be placed consisting of a major attack on a person's privacy with many more ways to do so.
Whys is kernel level access needed then If it brings so many complications for the developers of those systems? Most cheats operate on the kernel level within drives and different executable programs and this is required for the anti-cheat system to have kernel-level access to block those malicious entities from activating those cheats [12].
This is such a big issue, that countless journalists, media, and content creators cover and criticize regularly, whenever a company decides to produce an anti-cheat system with kernel-level access [13]. There are many instances where huge companies have come under fire and have had to make statements about their systems to alleviate user's concerns, companies like Valve and their VAC system[10], Riot and its Vanguard [14] and many more that are constantly brought into the light [15]. The goal of this project is to tackle this issue and provide a proposed solution, a tool that will function as an anti-cheat system that aims to detect cheating in a way that reduces privacy concerns. The details of how it accomplishes that goal are discussed at a later point.

2 Goals

The main goals of this project include conducting research into how cheats work, current methods of detecting them and how they can be improved. This is in combination with the goal of creating a tool that proposes a contributing strategy to help develop anti-cheat systems.
The development of the tool is focused mainly on creating a script based in the programming language Python, that aims to automate the usage of the Volatility3 [17] Python library as well as add a weighting system, to identify the possibility of a cheat being present in a game process within a memory dump. The weighting system works by assigning an integer number to each operation ran by the script. The size of the number depends on the result of the operation and how strongly it determines the presence of a cheat. Those weights then add up to output a log message to the user informing them of the likelihood for a cheat to be inside that memory dump. It ranges from not likely/non-existent to very likely/cheat exists. A memory dump is a file that contains information that was stored in the random access memory (RAM) and it is usually generated before a system failure so that it can be used to identify the cause of the crash [20]. It can also be generated manually for research purposes.
Python was chosen as the programming language for the script's development for a variety of reasons. It is easy to use, and easy to learn with plenty of resources that can be found online. Python is a very flexible language with many

supported third-party libraries that broaden the type of uses someone might have for the language [18]. Lastly, the Volatility3 is a memory forensics library written in Python, that can be used to help us detect cheats. Volatility3 was chosen to conduct memory forensic analysis since it is very flexible, having plugins available more multiple platforms and it can aid with many functions that could help indicate the presence of a cheat. The tool and the associated components involved in it were based on the Windows 10 operating system, which is the most popular choice for desktop users[19].
The final outcome of these goals is to contribute to the research against cheating and propose a work-of-progress tool that can help improve strategies and methods used in current and future anti-cheat systems. Developers and game companies can use this tool as an initial easy and efficient way to scan memory dumps sent by their system to detect cheats and flag cheaters before further heavy processing can be done that drains resources.

3 Literature Review

Research on cheat detection methods and approaches tends to fall under three general categories. Some methodologies focus on using artificial intelligence and machine learning methods, to help detect cheating patterns or behaviours through the use of those methods. Memory forensic approaches are used to analyze memory maps of a system or specific regions of a system's memory. One method is focused more on the active side of cheats whereas the other method focuses on the more technical side and the effects cheat file injections have on a system. There is also a third approach that is usually not as common where novel approaches are used that usually fall close to resembling one of the aforementioned methods. This project analyzes all three methods and reviews their advantages and disadvantages as well as how they relate to this project's development.

3.1 Machine learning based anti-cheat methods

3.1.1 Robust Vision-Based Cheat Detection in Competitive Gaming

There is research focused on a vision-based cheat detection method conducted by many authors affiliated with NVIDIA, a famous technology company with a main focus on designing graphics processing units (GPU) and supplying artificial intelligence hardware and software as well as many more activities [30]. The paper introduces a vision-based cheat detection method that involves the development of a deep neural network (DNN) based tool that uses machine learning based algorithms, to help analyze visual data extracted from gameplay frames. This method is tested and based on a popular multiplayer first-person shooter game called Counter-Strike: Global Offensive. The tool functions by capturing the frame buffer's final state and detecting game interface overlays that are assistive to cheating activity.
By using this method and training the tool, there are certain advantages to using this method. User privacy and system security are minimized since it is run independently of the game and does not require access to kernel-level data. It is

accurate in terms of detecting visual-based cheats (aimbots, wallhacks etc) that operate within an overlay. Robustness is another advantage the tool has, since it can detect cheats that do not cover a big part of the gameplay and the tool can be re-trained to help adapt to changes that can occur in cheating methods. And lastly, the tool can be automated to operate and re-train itself when necessary, eliminating the need for human intervention.
Some disadvantages and limitations are coupled with this method, a handful of them are acknowledged whereas others are not. Ease of deployment may be of concern for game developers that are not knowledgeable in machine learning and computer vision. The detection method is limited to only detecting visual-based cheats. What happens when cheats are not visual-based and don't contribute visual data anymore? This is talked upon, with a possible migration of other types of data like input/output (I/O) from peripheral devices like mice and keyboards, network traffic linked to different game actions and even game logs. This would also help identify different types of cheats that are not solely visual-based. False positives are not entirely eliminated leaving behind a possibility for users to be falsely flagged for cheating. Training is another aspect that may hinder system deployment. How often will the tool need re-training and how cost-intensive is it to keep re-training the tool? The game limitation is another concern since the tool has been tested on a limited amount of games and modes. There is a need for more flexibility to cover different game modes of the same game or overlay updates that may replace a standard overlay. The tool may be adapted easily to cover first-person shooter (FPS) games but what happens to games of different genres, that behave in a completely different manner with other overlays?
Overall, this paper proposed an efficient approach that recognises the need for it to be used not in isolation but in conjunction with other anti-cheat systems to maximize the efficiency of detecting cheats [31].

3.1.2 A Novel Approach to the Detection of Cheating in Multiplayer Online Games

There is another research conducted at the department of computer science at the University of York that uses machine learning based on player behaviour analysis as well as statistical techniques to determine cheating behaviour in-game. The proposed method is tested and built around the Valve Source engine, the same engine that is aforementioned in the previous paper.
It operates by analyzing data retrieved from "trace" entities that reflect what a player is looking at. The analysis would then be conducted to categorize the traces in different categories depending on different metrics, like an illegal trace (pointing the camera at a player through a wall) or trace distance to world material and more. Afterwards, it would compare how often the player conducted those traces that resembled a cheater's behaviour and depending on a formula that sums up the frequency of those traces it would determine if the number was high enough to flag a player as a cheater.
Like the above research, this method is efficient at detecting successfully a visual-based cheat. It shares a lot of common advantages with the previous method like automation and adaptability in recognizing cheat behaviour of similar vision-based cheats.

This method unfortunately is a lot more restricted, since there is no safeguard for preventing the method from becoming outdated, there is a limitation to detect only Source engine games, false positives are present and performance may be impacted from the sheer amount of data analyzed in real-time.
As a conclusion, this paper recognizes its need for future work to develop heterogeneity for both games it can be applied to and in-game cheats it can detect. There is also recognition for the method's need to be used in conjunction with other tools or systems to detect cheats[32].

3.2 AI-Based anti-cheat method

3.2.1 Cheat Detection in a Multiplayer First-Person Shooter Using Artificial Intelligence Tools

An article published by the University of Johannesburg in South Africa researches a method that is based on the use of artificial intelligence (AI) tools to detect cheating behaviour in first-person shooter games. The method proposed in this article involves the use of AI and machine learning techniques that construct models to analyse gameplay data and effectively classify normal and cheating behaviour based on metrics related to a specific game. The three techniques introduced are decision trees, support vector machines (SVMs) and Naïve Bayes classifiers. Those three models were then compared to see how their effectiveness varies in detecting cheating behaviour. The data used to determine cheating came from keyboard input as well as mouse input with game logs added along with it.
The use of decision trees yielded high accuracy results with a 95.45% accuracy in detecting player behaviour when not pruning a tree that uses keystroke data and a 100% accuracy while pruning a tree that utilizes console log data. The negative side of this method occurs when players press unbound keys arbitrarily, producing a substantial number of false positives. Thus, limiting the use of this method to be most efficient in distinguishing normal gameplay behaviour.
Support vector machines were next in use. Ensemble learning with the use of feature pair SVMs resulted in the highest average accuracy amongst the methods tested in this article. This was shown in all the types of data analyzed with the highest accuracy being in keystroke data.
Naïve Bayes classifiers was the last method applied in this article, and it resulted in being the least effective method to determine cheating behaviour. This was because the classifiers have a dependency on categorical data and console logs were the only form of data that could be grouped categorically. It was effective in analyzing that metric but with no further details provided beside the fact that further research would be required to transform Naïve Bayes classifiers into a more effective method.
Just like machine learning methods, similar or the same advantages are shared with this approach. One of those advantages is accuracy at detecting specific cheating behaviour which in our case would be non-human generated input and enhanced gameplay performance through the use of cheats. Adaptability by configuring those methods to be used for first-person shooter games that have common attributes.
Limitations associated with those AI-driven methods include a lack of variety in the detection of cheats. Visual-based cheats can go undetected since they do not

rely on input data or exhibit anomalous game log activity if the cheater doesn't abuse those cheats. It can abuse the cheat just enough to exhibit gameplay behaviour that resembles a good player rather than a cheater. False positives are another negative aspect that is commonly shared with machine learning methods that can lead to falsely tagging players as cheaters when in fact they are not.
The article presents some promising artificial intelligence-based methods that can be implemented within anti-cheat systems and tools, to help aid cheat detection through analysis of a player's behaviour in-game. Those do come with limitations that are acknowledged as a need for future research to be applied with the aim of improving the efficiency of those methods [33].

3.3 Memory forensic approach to anti-cheats

3.3.1 A method for preventing online games hacking using memory monitoring

An article written in the electronics and telecommunications research institute (ETRI) journal in Korea, proposes an approach that differs from the previous analyses that were focused on the domain of artificial intelligence and machine learning. It focuses on a more system-based approach that involved monitoring and analyzing system memory.
The method is separated into three different stages. It starts by collecting information from all the game client systems that are a part of the game server. It then separates the collected information depending on the operating system and its version, into safe and unsafe categories depending on the information. Finally, it analyzes the memory information of the game in real-time. If any modifications occur in the areas most likely to be affected by DLL injection, they will be analyzed to determine their legitimacy of them being a result of a hacking process. In the event that it is a malicious modification, action can be taken to shut down that game client's connection to the server and issue a ban.
This method proves to be efficient since it can provide real-time monitoring of memory modifications. It can accurately determine DLL injections as seen in the experiments conducted. Since there is memory monitoring related to operating system info, the method can be adapted to also identify specific operating system vulnerabilities. The method leaves open the possibility for modification along with process automation that does not require human intervention.
Limitations are associated with this method. Most recognizable and acknowledged by the authors is the need to establish further study and research, since memory monitoring of a specific area varies on the occasion. Performance costs of real-time monitoring are not taken into consideration for this method and may be of substantial impact when applying this method in a real setting. The method is limited to DLL injection modifications, there are no attempts at detecting different cheating methods, limiting the tool to a singular purpose. False positives can also be present when flagging legitimate modifications.
The research conducted with this method presents an effective strategy that can contribute to the detection of DLL injections within a game process. It acknowledges some limitations and imperfections that need honing through further research and study with a potential for future development [6].

3.4 Probabilistic Approach

3.4.1 Probabilistic Approaches to Cheating Detection in Online Games

A research paper published at the institute of electrical and electronics engineers (IEEE) conference in 2010, approaches cheat detection through a probabilistic approach combined with machine learning. This method takes the form of a framework, with the purpose of modelling games and the cheating behaviour associated with those games, for the use of cheat detection. The framework is designed to be applied to online multiplayer games of the first-person shooter (FPS) and the massively multiplayer online (MMO) genre. Two probabilistic frameworks are used to identify cheating behaviour.

The first framework is defined as an extension to the Bradley–Terry model, which functions by comparing player performance to their average game rank performance, which is sourced from other players' performance data in that rank. Afterwards, a probabilistic model is constructed that focuses on the results of the games that suspected cheaters play versus other players of the same rank. It can then determine if a cheater is playing according to their rank, or if their performance has been inflated through cheats to a rank that does not match the corresponding gameplay performance.

The second framework used is built upon multiple limit theorems which include the central limit theorem as well. This framework works similarly to the first one with the added assumption that a player's rank is known and has the expectation of it growing over time. Both of those frameworks work well in conjunction since the first one can be used to generate an estimated rank that can be used as a reference when using the second framework that has the real rank of that player.

This approach can prove to be very scalable, by applying those frameworks on even bigger datasets that can span entire regional game servers. Flexibility can be easily implemented to analyze different games with a ranking system that belongs to the method's covering genre (FPS, MMO). The analysis conducted on this data can also be used efficiently by the game developers to gain key insights into player cheating behaviour on a large scale. Real-time application of the method can be used for the detection of cheaters live in-game.

The negative aspects of this approach include, the need for rank information for the use of one of the frameworks. There are still possibilities of the method producing false positives and false negatives. This would be a high probability in games where smurfing is present. Smurfing is the act of a high-ranked player that is using another account usually of lower rank to play the game. This is not technically cheating since there is no violation in most games although it is unfair. Since the player's estimated rank would be higher than their actual rank, the framework can classify them as a cheater player

Summarizing this approach, it is a variation of similar methods discussed above that focus on machine learning. The difference here is the focus on the element that is detected, which in this case is the player performance and its relation to the average performance someone of that rank exhibits. This contrasts the other methods where the focus was on the similarities that behaviour shared with cheats behaviour [34].

4 Approaches

4.1 First Approach

Taking into consideration the above review of existing methodologies, the approach this project chose was using a memory forensics approach to detect cheats inside a memory dump. The majority of anti-cheat systems that exist, choose to operate on the kernel level. It is one of the most effective ways to detect and prevent the usage of cheats and that is why this tool will also operate on that same level. It will do so, by minimizing the privacy issues as much as possible to help steer away from the same path most anti-cheat systems take. The tool integrates into its design the possibility of it being used in conjunction with other tools since this is a key finding of the literature review on different approaches.
At the beginning, the first approach aimed to implement a script designed to single-handedly produce a memory dump and have written code to analyze the game process located within the dump. This proved to be ineffective since generating the memory dump not only added to the time of operation but also increased the cost in terms of space the script would need. Reducing the script to just analysis helped negate those added costs and improved the efficiency of the script. The next step was writing code to establish a way to scan through the memory dump and then analyze it to find cheats.

4.2 Second Approach

It was meaningless to come up with new code to perform actions that other libraries and frameworks accomplish more effectively, libraries and frameworks that have been developed and supported for many years by many skilled individuals. So the project opted to use the existing Python-based framework Volatility3 to scan and analyse memory dumps [20]. More specifically, volatility's Windows plugins were used with power shell command inputs in order to analyze the presence of cheats.
For analyzing purposes only, memory dumps were generated multiple times using the tool WinPmem [21]. Pslist was the first plugin used to determine the presence of cheating programs that may have been used at the time of the dump generation. Pslist [22]outputs a list of processes running at the time of the memory dump recording. Using pslist as a way to detect cheats, proved to be very superficial and in no way was reliable to identify the presence of cheats. The detection of a program that is known to be a cheat does not inherently mean that a cheat was applied, not only that but process names can also be changed, rendering the use of pslist as the sole method to detect cheats inefficiently.

```
Volatility 3 Framework 2.4.0

PID PPID ImageFileName Offset(V) Threads Handles SessionId Wow64 CreateTime ExitTime
File output

4 0 System     0x850455e7f040 284 -N/A False 2023-04-04 12:18:22.000000  N/A
Disabled
172 4 Registry 0x850455f06080 4 -N/A False 2023-04-04 12:18:19.000000  N/A Disabled
584 4 smss.exe 0x85045db92040 2 -N/A False 2023-04-04 12:18:22.000000  N/A Disabled
720 704 csrss.exe 0x8504604b1140 14 -0 False 2023-04-04 12:18:23.000000  N/A
Disabled
816 704 wininit.exe 0x850461a24080 1 -0 False 2023-04-04 12:18:25.000000  N/A
Disabled
824 808 csrss.exe 0x850461a2a140 20 -1 False 2023-04-04 12:18:25.000000  N/A
Disabled
```

Figure 3: Volatility3 Pslist output example

The next plugin used for the project was Dlllist [22]. It functions by retrieving the dynamically linked list of the specified process from the memory dump for the script to analyse. This allowed for checking if new DLL files were injected but not removed after function execution. It was a step forward but still not enough for it to be the only method used to detect cheats.

```
Volatility 3 Framework 2.4.0

PID Process Base Size Name Path LoadTime File output

9924 Cuphead.exe 0x7ffeeed50000 0x1f8000 ntdll.dll C:\WINDOWS\SYSTEM32\ntdll.dll
2023-04-04 13:00:54.000000  Disabled
9924 Cuphead.exe 0x7ffeed760000 0xbf000 KERNEL32.DLL C:\WINDOWS\System32\KERNEL32.DLL
2023-04-04 13:00:54.000000  Disabled
9924 Cuphead.exe 0x7ffeedac0000 0xae000 ADVAPI32.dll C:\WINDOWS\System32\ADVAPI32.dll
2023-04-04 13:00:54.000000  Disabled
9924 Cuphead.exe 0x7ffeed5f0000 0x9e000 msvcrt.dll C:\WINDOWS\System32\msvcrt.dll
2023-04-04 13:00:54.000000  Disabled
```

Figure 4: Volatility3 Dlllist output example

4.3 Final Approach

Malfind was afterwards chosen to attempt to detect cheats [22]. Malfind works by showing portentously malicious processes based on their memory page permissions and virtual address descriptor (VAD) tags. It improved cheat detection since it was a more in-depth approach that could be targeted at specific processes, there were some drawbacks though. Malfind did output a lot of false positives that could indicate activity of cheating being present even though there may not have been any cheats present. Again rendering it ineffective for it to be functioning isolated.

```
Volatility 3 Framework 2.4.0

PID Process Start VPN End VPN Tag Protection CommitCharge PrivateMemory File output
Hexdump Disasm

9924 Cuphead.exe 0x3e0000 0x3eefff VadS PAGE_EXECUTE_READWRITE 15 1 Disabled
89 4c 24 08 48 83 ec 38 .L$.H..8
e9 8b 31 02 00 ff 25 00 ..1...%.
00 00 00 00 a2 a4 8b fe ........
7f 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
```

Figure 5: Volatility3 malfind output example from Cuphead

Vadyarascan [22] was the last plugin the project incorporated within the script. Based on a ".yar" extension file that included the rules of searching for the plugin. The plugin scans the virtual address descriptor (VAD) tree of a process and identifies any regions of the memory that are marked as executable but are not related to any loaded modules. This plugin is as effective as the rules we have defined in the ".yar" extension file. For it to be effective at detecting cheats, we need to supply it with uniquely identifiable information about the cheat were looking for, which could be in the form of bytes, strings or file headers associated with those known cheats. For the tool's purpose, the unique identifiers that were supplied were known DLL cheat file names in the form of a string and bytes. This identifier can easily be changed so it is not the only identifier that was supplied. The module version ID (MVID) was also supplied as a unique identifier that represents the file. Something that is not as easily modified and is more reliable to use.
Threading was also implemented as a final change to help reduce processing time by assigning the tasks to different threads, resulting in a more optimised version of the tool. This change helped reduce processing time by an entire minute. The execution time is hard to approximate since many factors play into the execution of the tool like the memory dump contents or its size will affect exeuction time of the plugins increasing the overall time it takes to run the tool.

```
Volatility 3 Framework 2.4.0

Offset PID Rule Component Value

0x1fab1113a2e 9924 DLL_Detection $dllInj 41 00 73 00 73 00 65 00 6d 00 62 00 6c 00 79
00 2d 00 43 00 53 00 68 00 61 00 72 00 70 00 5f 00 65 00 78 00 70 00 2e 00 64 00 6c 00
6c 00

0x1fab111466e 9924 DLL_Detection $dllInj 41 00 73 00 73 00 65 00 6d 00 62 00 6c 00 79
00 2d 00 43 00 53 00 68 00 61 00 72 00 70 00 5f 00 65 00 78 00 70 00 2e 00 64 00 6c 00
6c 00
```

Figure 6: Volatility3 vadyarascan output example

To establish and develop the efficiency of the tool and to produce an effective cheat detection method, The aforementioned plugins were all combined under a weight system that was discussed early on in the paper. The most accurate plugin in terms of detecting a cheat would be vadyarascan. That is why we assigned it a weight of 7 out of 10. It reflects the importance of the results if any occur during the operation of vadyarascan. Malfind has a tendency of producing false positives so it was assigned a weight of 3 out of 10, reflecting the fact that if any results are present they should be investigated further on but it is not of serious importance. Dlllist is effective when it comes to lingering files that have not been unloaded after their execution, thus knowing the original DLL file count of a process can be effective when using this plugin. It yielded a weight of 6 out of 10 which reflects that fact.
The application of this strategy effectively narrows down the possibilities of a cheat being present in a dump. This project is limited by the lack of active data being fed into the script in real-time along with the fact that the testing is running on a game with a local client. Consequently, this allows cheaters to manipulate the game in a multitude of ways leaving us only with the ability to narrow down the possibility of a cheat being active without a 100% guarantee that a cheat was present inside that process within the memory dump that was analyzed.

5 System Implementation

5.1 Planning

The planning phase of the production of this tool took into consideration many factors that decided the development of this tool. How accessible can it be, how efficiently it will execute its main function and how easy is it to use. These three factors were the main considerations behind the development of this tool.
To help improve accessibility, the tool opted to use software that is not only widely supported but also widely used. For that reason, Python was chosen as the main programming language and the script was built around Windows 10 operating systems. This made the tool accessible to a wide array of users. The efficiency was determined by the nature of the tool, as it is an automation script

developed on already-existing plugins. This, in turn, helped with ease of use for a lot of users who may not be highly technical. The nature of this tool minimizes the system requirements needed to run it by only requiring someone to have a functioning computer system with Windows 10 as the designated operating system with the ability to download, install and run the latest version of Python. Another aspect of planning for the tool's development was determining which type of cheats would the tool be focused on detecting and why. Single-player game cheats and more specifically dynamic link library (DLL) file injection cheats would be the main focus of detection for the tool. Those cheats were chosen for the simple reason that they are an ideal environment that can harbour cheats. Since there is so much freedom to modify single-player games without penalties, it allowed us to test multiple games with multiple cheats which are further analyzed in the testing section.
With the development plan and the testing environment setup, the last part of planning was to improve and optimize the script by testing it rigorously and reducing time and space costs where possible.

5.2 Design

The design of this tool takes the form of an automation script that operates as an anti-cheat tool. This tool is primarily accessible to Windows 10 operating systems with the potential for functionality in different operating systems. The tool's main advantage in design is its ease to use as well as its limited use costs. It costs little in terms of both time and space as well as it is easy for it to be run by most people due to its user-friendly nature. It provides users with a basic idea of how anti-cheat systems work and the possibility to detect cheats that are present in memory dumps, with a possibility of an added user interface to help improve the tool's ease of use. The tool is designed in a way to support already existing anti-cheat systems and be used in conjunction with those systems or other tools. It can be easily integrated into existing workflows or automation pipelines depending on the purpose of its use. The tool can also function as a preemptive measure to quickly identify cheats, in hopes of reducing computational and staff work costs. The efficiency of this tool can also be improved in the future through a community and its user contributions by providing identifiable signatures of cheats that are known or found, to help the plugins used within the script. Customization can also be integrated in the future according to the needs of the system and also the game process being scanned in question.

5.3 Testing

As previously referenced, testing has been focused on single-player games and more specifically dynamic link library file injection cheats. Single-player games were chosen over multiplayer games for their ease of testing cheats without any hindrances. Multiplayer games operate on a client-to-server model making it difficult to test cheats since it would not only cause a ban of use due to existing anti-cheat systems in place but there would be a breach of fair play of that game causing negative consequences for the player base of that game. Single-player games were also chosen because of their endless possibility to be modified in many ways without penalties. Three games were chosen in particular for testing,

Half-Life 2 [28], Valheim [35] and Cuphead [29]. Half-Life 2 was chosen as a game that has been around for decades with multiple cheating methods that can be used to apply cheats to it. The game was developed using the Source engine made by Valve, an engine that was used to also make a popular competitive first-person shooter game called Counter-Strike: Global Offensive. A game that many players tend to cheat in due to its highly competitive nature of gameplay. Cuphead and Valheim were chosen as games that are newer and closely resemble more modern games these days. They are also difficult games that have enticed players to manufacture cheats for them, which are similar.
Testing was conducted in a Windows 10 virtual machine for the purpose of limiting malicious files that could sneak through the installation of the cheats and cheat tools. Cheating was applied through two methods, CheatEngine [4] was used to test dynamic link library (DLL) injection and the second method included files being manually replaced with modified versions of them (Assembly-CSharp.dll). Those cheats were further analyzed through the use of output logs generated from the tool and written to text files for testing references. Testing resulted in the detection of the DLL files that were either injected, executed and remained in the system or not executed and still dormant in the system through injection. Ten different files were tested for Cuphead (see Appendix E), five for Valheim (see Appendix F) and one for Half-Life 2 (see Appendix G) with the most popular ones being listed in the yara rule file. Half-Life 2 was the exception since the game is so old that most people opted in for a .exe program to apply cheats in real-time. DLL injection file cheats were very rare to acquire and the ones that were found and applied did not have a module version id, were non-functional or outdated, resulting in very few indications from the tool pointing to a cheat.
The output of the tool 80% of the time was strongly indicating the presence of cheats in those processes with a string output of ”There are many indications that point to cheats being present in the memory dump”, which only occurs when the sum of weights is at its highest point. The goals of testing aimed to reliably and consistently detect, the injection of DLL files in a game process, located inside a memory dump. It is safe to say that goal was accomplished but that can only be said for these two specific methods of cheating.

5.4 Maintenance

Maintenance of the tool is quite straightforward, keeping the necessary software needed constantly updated is enough for the tool to function for the foreseeable future. A form of user input on known unique identifiers for cheats will be needed for the vadyarascan plugin, to prevent the tool being massively outdated and restricted to a small sample size. Development of the tool can also continue in order to expand for different operating systems and to cover many more ways of cheating.

5.5 Time Management

Time management is a crucial aspect of this project. Carefully managing time allocation to tasks through the use of a Gantt chart proved to be an effective method of gauging not only progress over time but also any deviations from the project schedule (see Appendix C, D). Throughout the course of the project, the Gantt chart was updated to reflect changes made in the schedule and to correct inconsistencies. Changes were made in the second half of the project timeline to correct the time estimates. Tasks related to the report writing were moved, more specifically the literature review was moved towards January to complete the collection of literature and also write down the findings of it. Verification of memory dumps took less time than anticipated. It was not as difficult as expected to generate memory dumps and verify them. Lastly, tool development and functionality took longer to complete. This was because more methods were being explored to incorporate into the tool as well as optimization and testing was changing process throughout the project. Overall those deviations did not affect the progress of the project and were taken into consideration through the risk assessment table (see Appendix B).

6 Conclusion

6.1 Progress & Results

Summarizing the progress of this project, the tool has accomplished most of the goals of this project, although it has not quite reached them due to the limitations of the project. Testing single-player video game cheats on a local client-side process, with no exchange of information between a server and a client can be a problem. Giving a free environment to a cheater, to manipulate the game process in any way they see fit without any drawbacks, makes the goal of this project very hard to obtain. Adding to the fact that there is no real-time information being fed to the tool since the games tested are single-player, makes it a lot harder. Since the possibilities for the cheater to apply the cheats are endless, the best thing that can be done is to gauge with different methods to give an informed guess at the likelihood of a cheat being present in a memory dump. Comparing this tool to existing methodologies and systems out there, it has implemented the ability to be integrated into other anti-cheat systems, since that is a core aspect that can be inferred from the literature review. The tool is flexible, by being able to detect cheats in many different games without any restriction to game or genre. Since the main operation of the tool is to detect DLL injection, it can detect multiple cheats that can occur through that injection, varying the number of cheats that the tool can detect.

6.2 Future Work

In conclusion, there is great potential for future development of this tool to further expand it both in terms of usefulness as well as applications available for the tool. Creating a public open-source database to help feed into the script, particularly for the vadyarascan plugin, could help tremendously to detect known cheats by their unique identifiers by having a vast selection of them available. Alterations to the tool can be made, to add an extra function to check if in-game values have been modified. This is accomplished by checking static pointer address values that are linked to game values. As an example, there can be a game that has attributes related to the player like health, ammo and speed. If those values are known beforehand, then simply checking the value of the static pointer that links to those attributes is enough to know if the malicious modification has occurred. This does limit the tool since every game has different pointer objects associated with in game attributes that have different address values. There is also a possibility for remaking the tool into a form more suitable for productization and release to a general user market. This can be achieved by introducing a user interface that would be friendly in regards to non-technical people.

6.3 Cheat Use Ethics

There are plenty of debates regarding the ethics related to the use of cheating that can be discussed about this project and the tool that has been created[23]. Should cheating be acceptable in some cases? Is the modification of single-player games for more content, that doesn't violate fair play, something that should not be accepted? There are endless discussions around cheating and its form (game modification in single-player games) that many have debated about. Why should the modification of single-player games be penalized? When it can breathe life into a game that was forgotten only for more people to enjoy without any negative consequences.
A prime example of this is Garry's Mod [24][25], a game modification made by a company for one of their already existing games. A modification so popular[26] it constituted an entirely new game and a sandbox platform for other players to make their games within.

6.4 Limitations

With the inability of testing cheats and their detection in multiplayer games, the tool can only be left in a state that would require further development and testing on multiplayer games either designed with the ability of cheat use and modification or on existing games that would provide a sandbox environment isolated from real players to test cheats without violating the integrity of the fair play. The limitation of cheating methods the tool can detect is another problem that can be solved with more future research for incorporating other cheat detection methods. Lack of testing samples can be solved by further testing a larger variety of games, where DLL injection is a common method to cheat. This will help quantify the efficiency of the tool since there would be a larger collection of testing samples to analyze from.

7 References

[1] Game Genie (1992)(Codemasters) : Free Download, borrow, and streaming (1992) Internet Archive. Available at: archive.org (Accessed: April 2, 2023).

[2] IRDETO Global Gaming Survey Report (no date) Irdeto Resources Hub. Available at: irdeto.com(Accessed: April 2, 2023).

[3] Jeff Yan, J. and Choi, H.J. (2002) "Security issues in online games," The Electronic Library, 20(2), pp. 125–133. Available at: doi.org.

[4] Cheat engine (no date) Cheat Engine. Available at: www.cheatengine.org (Accessed: April 13, 2023).

[5] ArtMoney game cheater - powerful and easy to use game cheat engine utility. (no date) ArtMoney Game Cheater - Powerful and easy to use game cheat engine utility. Available at: www.artmoney.ru (Accessed: April 13, 2023).

[6] Lee, C.S. et al. (2020) "A method for preventing online games hacking using memory monitoring," ETRI Journal, 43(1), pp. 141–151. Available at: doi.org.

[7] Fragster (2022) The top 10 cheaters in esports who have been caught, Fragster. Available at: fragster.com (Accessed: April 13, 2023).

[8] Lee, S.J. et al. (2021) "Why do some users become enticed to cheating in competitive online games? an empirical study of cheating focused on competitive motivation, self-esteem, and aggression," Frontiers in Psychology, 12. Available at: doi.org.

[9] (no date) Even balance, inc. - punkbuster online countermeasures. Available at:
evenbalance.com (Accessed: April 6, 2023).

[10] Valve Anti-Cheat (VAC) and game bans (Steamworks documentation) (no date) Steamworks. Available at: steamgames.com (Accessed: April 5, 2023).

[11] Don't bear with the Cheaters (no date) Easy Anti-Cheat. Available at: easy.ac(Accessed: April 5, 2023).

[12] Maario, A. et al. (2021) "Redefining the risks of kernel-level anti-cheat in online gaming," 2021 8th International Conference on Signal Processing and Integrated Networks (SPIN) [Preprint]. Available at: doi.org.

[13] Menegus, B. (2022) What's the deal with Anti-Cheat Software in online games?, Wired. Conde Nast. Available at: wired.com(Accessed: April 7, 2023).

[14] What is Vanguard? – valorant support (no date). Available at: support-valorant.riotgames.com (Accessed: April 7, 2023).

[15] Hruska, J. (2020) Riot Games' new Anti-Cheat system runs at system boot, uses kernel driver, ExtremeTech. Available at: extremetech.com (Accessed: April 8, 2023).

[16] Blaukovitsch, R. (2022) Cheating in video games: The A to Z, Irdeto Insights. Available at: irdeto.com (Accessed: April 9, 2023).

[17] Volatilityfoundation (no date) Volatilityfoundation/Volatility3: Volatility 3.0 development, GitHub. Available at: github.com (Accessed: April 9, 2022).

[18] K. R. Srinath (2017) "Python – The Fastest Growing Programming Language," International Research Journal of Engineering and Technology (IRJET), 04(12), pp. 354–357.

[19] Operating system market share worldwide (2022) StatCounter Global Stats. Available at: statcounter.com (Accessed: April 9, 2023).

[20] Kirvan, P. (2023) What is a memory dump? – TechTarget definition, WhatIs.com. TechTarget. Available at: techtarget.com (Accessed: April 9, 2023).

[21] Velocidex (no date) VELOCIDEX/WinPmem: The multi-platform Memory Acquisition Tool., GitHub. Available at: github.com (Accessed: April 10, 2023).

[22] Volatilityfoundation (no date) Command reference, GitHub. Available at: github.com (Accessed: April 10, 2023).

[23] Bilińska-Reformat, K., Dewalska-Opitek, A. and Hofman-Kohlmeyer, M. (2020) "To mod or not to mod—an empirical study on game modding as customer value co-creation," Sustainability, 12(21), p. 9014. Available at: doi.org.

[24] Donnelly, J. (2015) The making of: Garry's mod, PCGamesN. Available at: pcgamesn.com (Accessed: April 12, 2023).

[25] Valve Corporation. (2006). Garry's Mod [Video game]. Steam. Available at: store.steampowered.com (Accessed: April 12, 2023).

[26] Garry's Mod - Steam Charts. (n.d.). Steam Charts - Tracking What's Played. Available at: steamcharts.com (Accessed: April 12, 2023).

[27] Buy WoW Items: Warcraft Gear & Mounts — PlayerAuctions. (n.d.). PlayerAuctions. playerauctions.com (Accessed: April 12, 2023).

[28] Valve Corporation. (2004). Half-Life 2 [Video game]. Steam. Available at: store.steampowered.com (Accessed: April 12, 2023).

[29] Studio MDHR Entertainment Inc. (2017). Cuphead [Video game]. Steam. Available at: store.steampowered.com (Accessed: April 12, 2023).

[30] World Leader in AI Computing. (n.d.). NVIDIA. nvidia.com(Accessed: April 12, 2023).

[31] Jonnalagadda, A. et al. (2021) "Robust vision-based cheat detection in competitive gaming," Proceedings of the ACM on Computer Graphics and Interactive Techniques, 4(1), pp. 1–18. Available at: doi.org.

[32] Laurens, P. et al. (2007) "A novel approach to the detection of cheating in multiplayer online games," 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007) [Preprint]. Available at: doi.org.

[33] Spijkerman, R. and Marie Ehlers, E. (2020) "Cheat detection in a multiplayer first-person shooter using Artificial Intelligence Tools," 2020 The 3rd International Conference on Computational Intelligence and Intelligent Systems [Preprint]. Available at: doi.org.

[34] Chapel, L., Botvich, D. and Malone, D. (2010) "Probabilistic approaches to cheating detection in online games," Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games [Preprint]. Available at: doi.org.

[35] Valheim on steam (no date) Valheim on Steam. Available at: store.steampowered.com (Accessed: April 16, 2023).

8 Appendices

8.1 Appendix A - Brief

Problem

In today's day and age, the gaming industry has become a multifaceted capital machine worth millions, even billions. With its growth, it harbours danger and attackers that try to capitalize on or disrupt that industry. More specifically, people are trying to disrupt the integrity and fun that comes with games by using hacks to gain an unfair advantage against other players on the game they are attacking. Hacking to gain an unfair advantage can majorly disrupt and damage the integrity and availability of casual and competitive multiplayer games. It is a problem of serious importance, that affects people, and this project will try to solve it.The approach to this problem will start by trying to find a game that will enable a simulated attack to be performed in order to have a testing ground for the tool. Afterwards, the development of the tool and the technical specifications need to be set to create the tool and figure out how it will work and be used. Finally, detection of the third party software used to hack the game will need to be detected by the tool.

Goals

This project, aims to create a tool that will be used in combination with games in order to scan a memory dump and identify if third party software is used to interfere with the integrity of a multiplayer game and create an unfair advantage against other players. A simulation of an attack on a game will be created, which the tool will try to flag as a potential hacking attack. This will be done be checking the memory dump and cross-reference it against a database of known hacking methods.

Scope

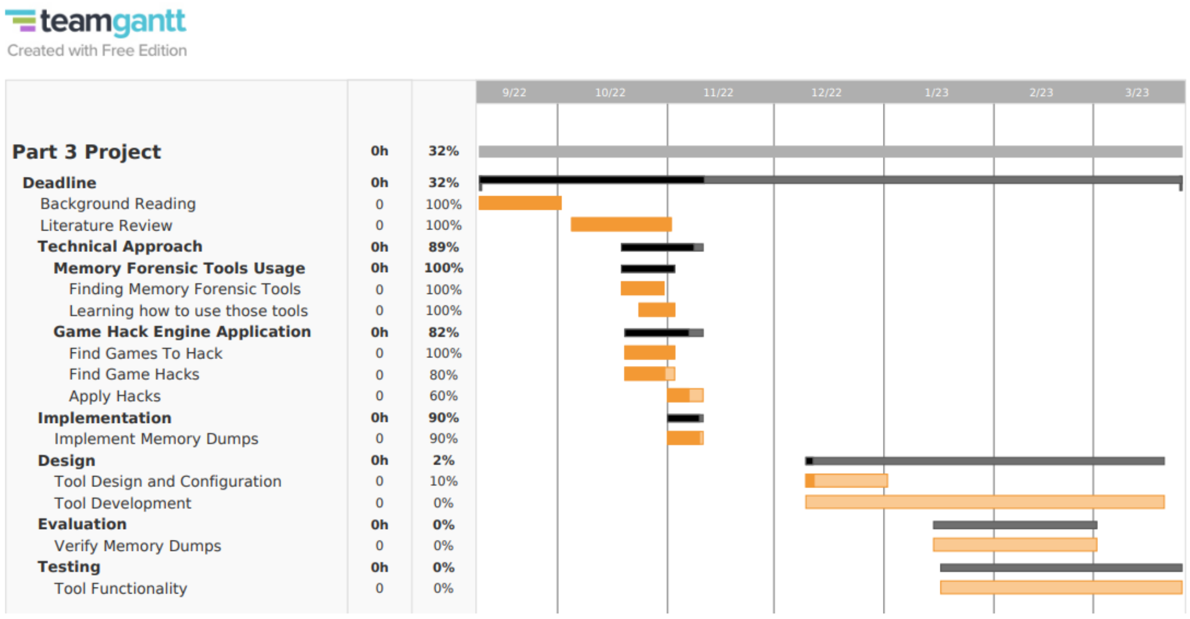
The scope of this project lies in the cybersecurity domain relating to malicious software used to alter the integrity of software. Game design is also included in the scope of this project, since the tool will be tested with video games as the foundation of the tool. The field of forensics is also within the scope since the tool uses application of computer science to detect criminal activity within a game.

8.2 Appendix B - Risk Assessment

Risk	**Likelihood (1-5)**	**Impact (1-5)**	**Risk Score (Likelihood x Impact)**	**Response Strategy**
Equipment Failure	1	5	5	Store project files in cloud for easy access with different equipment
Technical Issues	3	4	12	Make sure everything within the project is compatible with each other and thoroughly check software used
Bad Time Management	2	4	8	Plan out activities and stick to a routinely schedule
Lack of Knowledge	4	4	16	Research and access as many resources possible related to the project tool development
External Issues	5	3	15	Try to not get affected in both terms of time and mental health
Quality Risks	3	4	12	Quality testing and improvements as often as possible

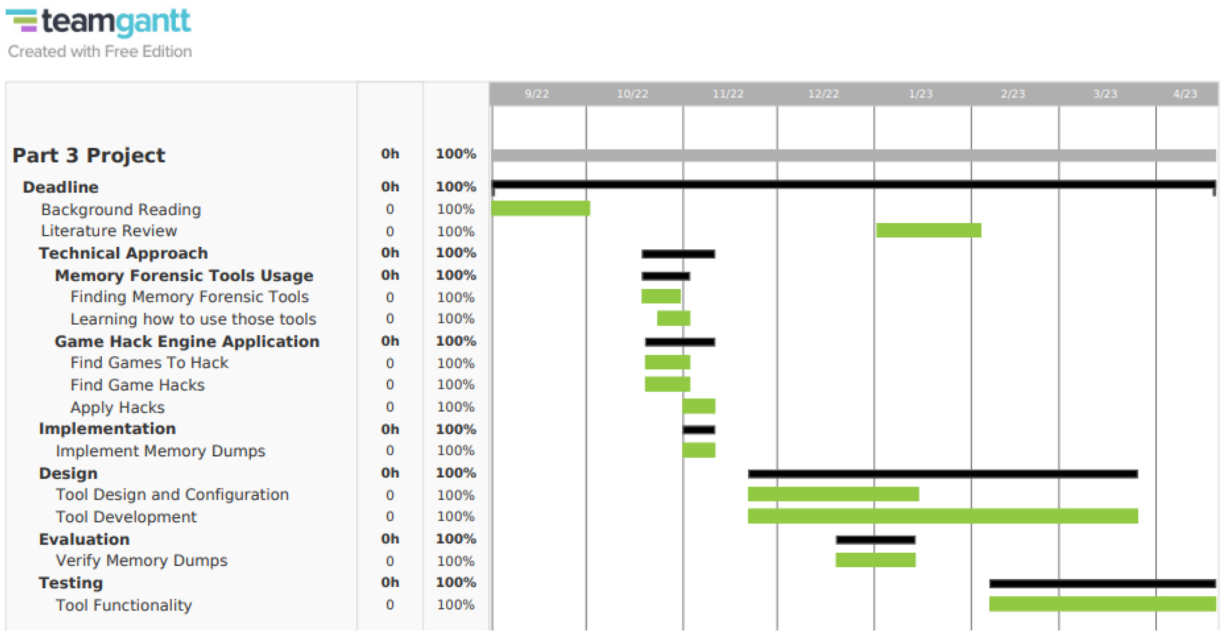
Table 1: Risk Assessment Table

8.3 Appendix C - Old Gantt Chart



Part 3 Project	**0h**	**32%**
Deadline	**0h**	**32%**
Background Reading	0	100%
Literature Review	0	100%
Technical Approach	**0h**	**89%**
Memory Forensic Tools Usage	**0h**	**100%**
Finding Memory Forensic Tools	0	100%
Learning how to use those tools	0	100%
Game Hack Engine Application	**0h**	**82%**
Find Games To Hack	0	100%
Find Game Hacks	0	80%
Apply Hacks	0	60%
Implementation	**0h**	**90%**
Implement Memory Dumps	0	90%
Design	**0h**	**2%**
Tool Design and Configuration	0	10%
Tool Development	0	0%
Evaluation	**0h**	**0%**
Verify Memory Dumps	0	0%
Testing	**0h**	**0%**
Tool Functionality	0	0%

8.4 Appendix D - Updated Gantt Chart



Part 3 Project	**0h**	**100%**
Deadline	**0h**	**100%**
Background Reading	0	100%
Literature Review	0	100%
Technical Approach	**0h**	**100%**
Memory Forensic Tools Usage	**0h**	**100%**
Finding Memory Forensic Tools	0	100%
Learning how to use those tools	0	100%
Game Hack Engine Application	**0h**	**100%**
Find Games To Hack	0	100%
Find Game Hacks	0	100%
Apply Hacks	0	100%
Implementation	**0h**	**100%**
Implement Memory Dumps	0	100%
Design	**0h**	**100%**
Tool Design and Configuration	0	100%
Tool Development	0	100%
Evaluation	**0h**	**100%**
Verify Memory Dumps	0	100%
Testing	**0h**	**100%**
Tool Functionality	0	100%

8.5 Appendix E - Cuphead Test Logs

```
Volatility 3 Framework 2.4.0

PID Process Base Size Name Path LoadTime File output

16512 Cuphead.exe 0x7ff681410000 0xa4000 Cuphead.exe F:\SteamLibrary\steamapps\
common\Cuphead\Cuphead.exe 2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84f070000 0x1f8000 ntdll.dll C:\WINDOWS\SYSTEM32\ntdll.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84e020000 0xbf000 KERNEL32.DLL C:\WINDOWS\System32\KERNEL32.DLL
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84c840000 0x2da000 KERNELBASE.dll C:\WINDOWS\System32\KERNELBASE.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84ef00000 0xae000 ADVAPI32.dll C:\WINDOWS\System32\ADVAPI32.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84e3b0000 0x9e000 msvcrt.dll C:\WINDOWS\System32\msvcrt.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84def0000 0x9c000 sechost.dll C:\WINDOWS\System32\sechost.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84db80000 0x126000 RPCRT4.dll C:\WINDOWS\System32\RPCRT4.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff806f20000 0x1602000 UnityPlayer.dll F:\SteamLibrary\steamapps\
common\Cuphead\UnityPlayer.dll 2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84e580000 0x19d000 USER32.dll C:\WINDOWS\System32\USER32.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84cb20000 0x22000 win32u.dll C:\WINDOWS\System32\win32u.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84afe0000 0xd000 HID.DLL C:\WINDOWS\SYSTEM32\HID.DLL
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84c150000 0xc000 CRYPTBASE.DLL C:\WINDOWS\SYSTEM32\CRYPTBASE.DLL
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84d6c0000 0x2b000 GDI32.dll C:\WINDOWS\System32\GDI32.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84cfc0000 0x110000 gdi32full.dll C:\WINDOWS\System32\gdi32full.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84ccb0000 0x9d000 msvcp_win.dll C:\WINDOWS\System32\msvcp_win.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84cec0000 0x100000 ucrtbase.dll C:\WINDOWS\System32\ucrtbase.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84e450000 0x12a000 ole32.dll C:\WINDOWS\System32\ole32.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84d1e0000 0x354000 combase.dll C:\WINDOWS\System32\combase.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84dfc0000 0x55000 SHLWAPI.dll C:\WINDOWS\System32\SHLWAPI.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff845ef0000 0xa000 VERSION.dll C:\WINDOWS\SYSTEM32\VERSION.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84d710000 0x468000 SETUPAPI.dll C:\WINDOWS\System32\SETUPAPI.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84ce00000 0x4e000 cfgmgr32.dll C:\WINDOWS\System32\cfgmgr32.dll
2023-04-30 18:15:49.000000  Disabled
16512 Cuphead.exe 0x7ff84c810000 0x27000 bcrypt.dll C:\WINDOWS\System32\bcrypt.dll
2023-04-30 18:15:49.000000  Disabled
```

Figure 7: Volatility3 dlllist output snippet from modified Cuphead test

```
Volatility 3 Framework 2.4.0

PID Process Start VPN End VPN Tag Protection CommitCharge PrivateMemory File output
Hexdump Disasm

16512 Cuphead.exe 0x24400280000 0x24400280fff VadS PAGE_EXECUTE_READWRITE 1 1
Disabled
9c 50 51 52 53 54 55 56 .PQRSTUV
57 41 50 41 51 41 52 41 WAPAQARA
53 41 54 41 55 41 56 41 SATAUAVA
57 48 83 ec 20 48 8b 0d WH...H..
29 00 00 00 ff 15 2b 00 ).....+.
00 00 48 83 c4 20 41 5f ..H...A_
41 5e 41 5d 41 5c 41 5b A^A]A\A[
41 5a 41 59 41 58 5f 5e AZAYAX_^
0x24400280000: pushfq
0x24400280001: push rax
0x24400280002: push rcx
0x24400280003: push rdx
0x24400280004: push rbx
0x24400280005: push rsp
0x24400280006: push rbp
0x24400280007: push rsi
0x24400280008: push rdi
0x24400280009: push r8
0x2440028000b: push r9
0x2440028000d: push r10
0x2440028000f: push r11
0x24400280011: push r12
0x24400280013: push r13
0x24400280015: push r14
0x24400280017: push r15
0x24400280019: sub rsp, 0x20
0x2440028001d: mov rcx, qword ptr [rip + 0x29]
0x24400280024: call qword ptr [rip + 0x2b]
0x2440028002a: add rsp, 0x20
0x2440028002e: pop r15
0x24400280030: pop r14
0x24400280032: pop r13
0x24400280034: pop r12
0x24400280036: pop r11
0x24400280038: pop r10
0x2440028003a: pop r9
0x2440028003c: pop r8
0x2440028003e: pop rdi
0x2440028003f: pop rsi
16512 Cuphead.exe 0x3e0000 0x3eefff VadS PAGE_EXECUTE_READWRITE 15 1 Disabled
89 4c 24 08 48 83 ec 38 .L$.H..8
e9 8b 31 02 00 ff 25 00 ..1...%.
00 00 00 00 a2 7a 14 f8 .....z..
7f 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
```

Figure 8: Volatility3 malfind output snippet from modified Cuphead test

```
Volatility 3 Framework 2.4.0

Offset PID Rule Component Value

0x24406ad6940 16512 DLL_Detection $expModMVID 39 45 41 37 45 35 36 39 2d 43 33 36 45 2d 34 30
42 43 2d 41 38 36 32 2d 30 46 41 32 37 44 43 32 38 43 32 32
0x24406b7d686 16512 DLL_Detection $trainerModName 43 00 75 00 70 00 68 00 65 00 61 00 64 00 54
00 72 00 61 00 69 00 6e 00 65 00 72 00 2d 00 76 00 31 00 2e 00 64 00 6c 00 6c 00
0x24406b7d5c6 16512 DLL_Detection $accessModName 43 00 75 00 70 00 68 00 65 00 61 00 64 00 5f
00 41 00 63 00 63 00 65 00 73 00 73 00 69 00 62 00 69 00 6c 00 69 00 74 00 79 00 2e 00 64 00 6c
00 6c 00
0x24406cd6da1 16512 DLL_Detection $accessModName 43 75 70 68 65 61 64 5f 41 63 63 65 73 73 69
62 69 6c 69 74 79 2e 64 6c 6c
0x24406cfc686 16512 DLL_Detection $trainerModName 43 00 75 00 70 00 68 00 65 00 61 00 64 00 54
00 72 00 61 00 69 00 6e 00 65 00 72 00 2d 00 76 00 31 00 2e 00 64 00 6c 00 6c 00
0x24406cfc206 16512 DLL_Detection $accessModName 43 00 75 00 70 00 68 00 65 00 61 00 64 00 5f
00 41 00 63 00 63 00 65 00 73 00 73 00 69 00 62 00 69 00 6c 00 69 00 74 00 79 00 2e 00 64 00 6c
00 6c 00
0x24406cfcf86 16512 DLL_Detection $accessModName 43 00 75 00 70 00 68 00 65 00 61 00 64 00 5f
00 41 00 63 00 63 00 65 00 73 00 73 00 69 00 62 00 69 00 6c 00 69 00 74 00 79 00 2e 00 64 00 6c
00 6c 00
0x24406d5105a 16512 DLL_Detection $trainerModName 43 75 70 68 65 61 64 54 72 61 69 6e 65 72 2d
76 31 2e 64 6c 6c
0x24406d51290 16512 DLL_Detection $accessModName 43 75 70 68 65 61 64 5f 41 63 63 65 73 73 69
62 69 6c 69 74 79 2e 64 6c 6c
0x24406e28e74 16512 DLL_Detection $accessModName 43 00 75 00 70 00 68 00 65 00 61 00 64 00 5f
00 41 00 63 00 63 00 65 00 73 00 73 00 69 00 62 00 69 00 6c 00 69 00 74 00 79 00 2e 00 64 00
6c 00 6c 00
0x24406e28fb4 16512 DLL_Detection $accessModName 43 00 75 00 70 00 68 00 65 00 61 00 64 00 5f
00 41 00 63 00 63 00 65 00 73 00 73 00 69 00 62 00 69 00 6c 00 69 00 74 00 79 00 2e 00 64 00 6c
00 6c 00
0x24407307164 16512 DLL_Detection $trainerModMVID 30 00 39 00 38 00 37 00 39 00 34 00 33 00 63
00 2d 00 38 00 37 00 35 00 30 00 2d 00 34 00 33 00 31 00 34 00 2d 00 38 00 34 00 33 00 65 00
2d 00 39 00 30 00 35 00 33 00 61 00 64 00 66 00 36 00 35 00 63 00 62 00 35 00
0x24407307b74 16512 DLL_Detection $trainerModMVID 30 00 39 00 38 00 37 00 39 00 34 00 33 00 63
00 2d 00 38 00 37 00 35 00 30 00 2d 00 34 00 33 00 31 00 34 00 2d 00 38 00 34 00 33 00 65 00
2d 00 39 00 30 00 35 00 33 00 61 00 64 00 66 00 36 00 35 00 63 00 62 00 35 00
0x244073c2864 16512 DLL_Detection $accessModMVID 31 00 33 00 32 00 30 00 64 00 31 00 34 00 30
00 2d 00 32 00 33 00 32 00 39 00 2d 00 34 00 63 00 35 00 62 00 2d 00 38 00 64 00 34 00 39 00
2d 00 37 00 64 00 61 00 32 00 62 00 32 00 32 00 34 00 34 00 62 00 34 00 35 00
0x244073c2f64 16512 DLL_Detection $accessModMVID 31 00 33 00 32 00 30 00 64 00 31 00 34 00 30
00 2d 00 32 00 33 00 32 00 39 00 2d 00 34 00 63 00 35 00 62 00 2d 00 38 00 64 00 34 00 39 00
2d 00 37 00 64 00 61 00 32 00 62 00 32 00 32 00 34 00 34 00 62 00 34 00 35 00
0x2440741a2b4 16512 DLL_Detection $accessModMVID 31 00 33 00 32 00 30 00 64 00 31 00 34 00 30
00 2d 00 32 00 33 00 32 00 39 00 2d 00 34 00 63 00 35 00 62 00 2d 00 38 00 64 00 34 00 39 00
2d 00 37 00 64 00 61 00 32 00 62 00 32 00 32 00 34 00 34 00 62 00 34 00 35 00
```

Figure 9: Volatility3 vadyarascan output snippet from modified Cuphead test

```
====================================================
|            Starting Anti-Cheat Script            |
====================================================

[13:43:20] Running windows.pslist to get the memory dump process list....
[13:44:06] Pslist finished succesfully

[13:44:06] Scanning processes.txt to find PID for Cuphead....
[13:44:06] Running windows.dlllist to get the DLL list for Cuphead....
[13:44:06] Running windows.malfind to identify memory permissions....
[13:44:06] Running windows.vadyarascan to scan known file signatures....
[13:44:53] Dlllist finished succesfully

[13:44:53] Malfind found suspicious VAD tags
[13:44:53] Malfind finished succesfully

[13:45:18] Vadyarascan found file signatures belonging to cheats
[13:45:18] Vadyarascan finished succesfully

[13:45:18] There are many indications that point to cheats being present in the
process
```

Figure 10: Tool output from modified Cuphead test

8.6 Appendix F - Valheim Test Logs

```
Volatility 3 Framework 2.4.0

PID Process Base Size Name Path LoadTime File output

26312 valheim.exe 0x7ff6642a0000 0xa5000 valheim.exe C:\Program Files (x86)\Steam\steamapps\
common\Valheim\valheim.exe 2023-05-01 18:40:40.000000  Disabled
26312 valheim.exe 0x7ff84f070000 0x1f8000 ntdll.dll C:\WINDOWS\SYSTEM32\ntdll.dll
2023-05-01 18:40:40.000000  Disabled
26312 valheim.exe 0x7ff84e020000 0xbf000 KERNEL32.DLL C:\WINDOWS\System32\KERNEL32.DLL
2023-05-01 18:40:40.000000  Disabled
26312 valheim.exe 0x7ff84c840000 0x2da000 KERNELBASE.dll C:\WINDOWS\System32\KERNELBASE.dll
2023-05-01 18:40:40.000000  Disabled
26312 valheim.exe 0x7fffa6a90000 0x1c12000 UnityPlayer.dll C:\Program Files (x86)\Steam\
steamapps\common\Valheim\UnityPlayer.dll 2023-05-01 18:40:40.000000  Disabled
26312 valheim.exe 0x7ff84e580000 0x19d000 USER32.dll C:\WINDOWS\System32\USER32.dll
2023-05-01 18:40:40.000000  Disabled
26312 valheim.exe 0x7ff84cb20000 0x22000 win32u.dll C:\WINDOWS\System32\win32u.dll
2023-05-01 18:40:40.000000  Disabled
26312 valheim.exe 0x7ff84d6c0000 0x2b000 GDI32.dll C:\WINDOWS\System32\GDI32.dll
2023-05-01 18:40:40.000000  Disabled
26312 valheim.exe 0x18fa63a0000 0xe2000 assembly_valheim.dll C:\Users\jackp\Desktop\Valheim\
Valheim ShieldSupporter\Valheim\assembly_valheim.dll 2023-05-01 18:51:26.000000  Disabled
26312 valheim.exe 0x18fa6910000 0xe2000 assembly_valheim.dll C:\Users\jackp\Desktop\Valheim\
assembly_valheim.dll 2023-05-01 19:16:05.000000  Disabled
26312 valheim.exe 0x18f7fbc0000 0xc000 SharpMonoInjector.dll C:\Users\jackp\Desktop\Valheim\
SharpMonoInjector.dll 2023-05-01 19:16:13.000000  Disabled
26312 valheim.exe 0x18fa6a00000 0x8000 shieldsupporter.dll C:\Users\jackp\Desktop\Valheim\
shieldsupporter.dll 2023-05-01 19:16:19.000000  Disabled
26312 valheim.exe 0x18fa6a10000 0xfc000 UnityEngine.CoreModule.dll
C:\Users\jackp\Desktop\Valheim\
UnityEngine.CoreModule.dll 2023-05-01 19:16:29.000000  Disabled
26312 valheim.exe 0x18fa6b10000 0x2c000 UnityEngine.IMGUIModule.dll
C:\Users\jackp\Desktop\Valheim\
UnityEngine.IMGUIModule.dll 2023-05-01 19:16:32.000000  Disabled
26312 valheim.exe 0x18fa6b40000 0x10000 Duranda.dll C:\Users\jackp\Desktop\Valheim\
Duranda.dll 2023-05-01 19:29:22.000000  Disabled
```

Figure 11: Volatility3 dlllist output snippet from Valheim test

```
Volatility 3 Framework 2.4.0

PID Process Start VPN End VPN Tag Protection CommitCharge PrivateMemory File output
Hexdump Disasm

35940 hl2.exe 0x1d9a0000 0x1d9a0fff VadS PAGE_EXECUTE_READWRITE 1 1 Disabled
00 00 00 00 80 0f 9a 1d ........
03 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
0x1d9a0000: add byte ptr [eax], al
0x1d9a0002: add byte ptr [eax], al
0x1d9a0004: or byte ptr [edi], 0x9a
0x1d9a0007: sbb eax, 3
0x1d9a000c: add byte ptr [eax], al
0x1d9a000e: add byte ptr [eax], al
0x1d9a0010: add byte ptr [eax], al
0x1d9a0012: add byte ptr [eax], al
0x1d9a0014: add byte ptr [eax], al
0x1d9a0016: add byte ptr [eax], al
0x1d9a0018: add byte ptr [eax], al
0x1d9a001a: add byte ptr [eax], al
0x1d9a001c: add byte ptr [eax], al
0x1d9a001e: add byte ptr [eax], al
0x1d9a0020: add byte ptr [eax], al
0x1d9a0022: add byte ptr [eax], al
0x1d9a0024: add byte ptr [eax], al
0x1d9a0026: add byte ptr [eax], al
0x1d9a0028: add byte ptr [eax], al
0x1d9a002a: add byte ptr [eax], al
0x1d9a002c: add byte ptr [eax], al
0x1d9a002e: add byte ptr [eax], al
0x1d9a0030: add byte ptr [eax], al
0x1d9a0032: add byte ptr [eax], al
0x1d9a0034: add byte ptr [eax], al
0x1d9a0036: add byte ptr [eax], al
0x1d9a0038: add byte ptr [eax], al
0x1d9a003a: add byte ptr [eax], al
0x1d9a003c: add byte ptr [eax], al
0x1d9a003e: add byte ptr [eax], al
35940 hl2.exe 0x22e80000 0x22e80fff VadS PAGE_EXECUTE_READWRITE 1 1 Disabled
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
```

Figure 12: Volatility3 malfind output snippet from Valheim test

```
Volatility 3 Framework 2.4.0

Offset PID Rule Component Value

0x18ee89dd270 26312 DLL_Injection_Detection $durandaModMVID 38 32 46 36 31 41 31 45
2d 33 39 43 34 2d 34 30 43 32 2d 42 38 30 41 2d 39 37 45 38 30 44 33 31 31 36 45 38
```

Figure 13: Volatility3 vadyarascan output from Valheim test

```
====================================================
|            Starting Anti-Cheat Script            |
====================================================

[20:39:19] Running windows.pslist to get the memory dump process list....
[20:40:08] Pslist finished succesfully

[20:40:08] Scanning processes.txt to find PID for valheim....
[20:40:08] Running windows.dlllist to get the DLL list for valheim....
[20:40:08] Running windows.malfind to identify memory permissions....
[20:40:08] Running windows.vadyarascan to scan known file signatures....
[20:40:54] Dlllist found extra DLL files

[20:40:56] Malfind found suspicious VAD tags
[20:40:56] Malfind finished succesfully
[20:43:37] There are many indications that point to cheats being present in the process
```

Figure 14: Tool output from modified Valheim test

8.7 Appendix G - Half-Life 2 Test Logs

```
Volatility 3 Framework 2.4.0

PID Process Base Size Name Path LoadTime File output

35940 hl2.exe 0xea0000 0x21000 hl2.exe F:\SteamLibrary\steamapps\common\
Half-Life 2\hl2.exe 2023-04-02 12:26:27.000000  Disabled
35940 hl2.exe 0x7ff84f070000 0x1f8000 ntdll.dll C:\WINDOWS\SYSTEM32\ntdll.dll
2023-04-02 12:26:27.000000  Disabled
35940 hl2.exe 0x7ff84e290000 0x59000 wow64.dll C:\WINDOWS\System32\wow64.dll
2023-04-02 12:26:27.000000  Disabled
35940 hl2.exe 0x7ff84dcb0000 0x83000 wow64win.dll C:\WINDOWS\System32\wow64win.dll
2023-04-02 12:26:27.000000  Disabled
35940 hl2.exe 0x77ba0000 0xa000 wow64cpu.dll C:\WINDOWS\System32\wow64cpu.dll
2023-04-02 12:26:27.000000  Disabled
```

Figure 15: Volatility3 dlllist output snippet from Half Life 2 test

```
====================================================
|            Starting Anti-Cheat Script            |
====================================================

[15:14:01] Running windows.pslist to get the memory dump process list....
[15:14:52] Pslist finished succesfully

[15:14:52] Scanning processes.txt to find PID for hl2....
[15:14:52] Running windows.dlllist to get the DLL list for hl2....
[15:14:52] Running windows.malfind to identify memory permissions....
[15:15:36] Dlllist finished succesfully

[15:15:45] Malfind found suspicious VAD tags
[15:15:45] Malfind finished succesfully

[15:16:09] Vadyarascan finished succesfully

[15:16:09] There are little to no indications that cheats are present in the process
```

Figure 16: Tool output from Half Life 2 test

```
Volatility 3 Framework 2.4.0

PID Process Start VPN End VPN Tag Protection CommitCharge PrivateMemory File output
Hexdump Disasm

26312 valheim.exe 0x18fb96b0000 0x18fb96bffff VadS PAGE_EXECUTE_READWRITE 16 1
Disabled
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
00 00 00 00 00 00 00 00 ........
0x18fb96b0000: add byte ptr [rax], al
0x18fb96b0002: add byte ptr [rax], al
0x18fb96b0004: add byte ptr [rax], al
0x18fb96b0006: add byte ptr [rax], al
0x18fb96b0008: add byte ptr [rax], al
0x18fb96b000a: add byte ptr [rax], al
0x18fb96b000c: add byte ptr [rax], al
0x18fb96b000e: add byte ptr [rax], al
0x18fb96b0010: add byte ptr [rax], al
0x18fb96b0012: add byte ptr [rax], al
0x18fb96b0014: add byte ptr [rax], al
0x18fb96b0016: add byte ptr [rax], al
0x18fb96b0018: add byte ptr [rax], al
0x18fb96b001a: add byte ptr [rax], al
0x18fb96b001c: add byte ptr [rax], al
0x18fb96b001e: add byte ptr [rax], al
0x18fb96b0020: add byte ptr [rax], al
0x18fb96b0022: add byte ptr [rax], al
0x18fb96b0024: add byte ptr [rax], al
0x18fb96b0026: add byte ptr [rax], al
0x18fb96b0028: add byte ptr [rax], al
0x18fb96b002a: add byte ptr [rax], al
0x18fb96b002c: add byte ptr [rax], al
0x18fb96b002e: add byte ptr [rax], al
0x18fb96b0030: add byte ptr [rax], al
0x18fb96b0032: add byte ptr [rax], al
0x18fb96b0034: add byte ptr [rax], al
0x18fb96b0036: add byte ptr [rax], al
0x18fb96b0038: add byte ptr [rax], al
0x18fb96b003a: add byte ptr [rax], al
0x18fb96b003c: add byte ptr [rax], al
0x18fb96b003e: add byte ptr [rax], al
26312 valheim.exe 0x18d4e4e0000 0x18d4e4effff VadS PAGE_EXECUTE_READWRITE 16 1
Disabled
55 48 8b ec 48 81 ec 40 UH..H..@
```

Figure 17: Volatility3 malfind output snippet from Half Life 2 test