NTNU
Norwegian University of Science and Technology

Information security as a countermeasure against cheating in video games

Kevin Kjelgren Mikkelsen

Master in Information Security
Submission date: June 2017
Supervisor: Erik Hjelmås, IIK
Co-supervisor: Simon McCallum, IDI

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Preface

This is a master thesis in Information security at NTNU that have been carried out over the course of a year and a half starting spring semester of 2016 and finished spring semester 2017. The goal of this paper is to prevent cheating in competitive video games. This paper assumes a background in software development and should be of interest for anyone working on a competitive video game or business solution with similar requirements in technology and performance.

The issue of players being able to modify their game clients or the data stream to the game server in order to gain unfair advantages is an increasing problem in video games as the stakes in competitive games grow higher. This is also an issue if businesses are to use the already existing technology in game engines for development of virtual business meeting solutions and similarly security dependent applications. The way this paper aims to reach this goal is by looking at video games from an information security point of view to figure out if these problems have been faced in other areas of software development where security have had more of a priority, how these issues have been handled and whether there are existing solutions that can be of use in a game development setting.

31-05-2017

Acknowledgment

I would like to thank supervisor Simon McCallum for continued support and input during this project with his help in providing an angle for the project as well as the help sorting out what areas to focus and game engines where this research might prove the most useful.

I would also like to thank the unreal engine developers for providing a widely used open source engine making the research for this project easier and useful for a larger group of developers, as well as some helpful people on their forum for providing input and pointers on the inner workings of the unreal game engine.

K.K.M.

Abstract

Most cheating in video games is possible due to information being accessible outside the intended frames of the game developer. The issue of protecting sensitive information have been handled in many areas outside of video games for a long time now. The goal of this paper is to review these information security solutions that are in use in more security concerned areas today and to potentially find transferable approaches that can help protect important and sensitive information in video games and that way help prevent cheating.

In this paper the current threat and approaches of cheating is investigated to find the most commonly used security breaches. Then potential ways to increase the security of these areas are reviewed in order to find transferable security measures. A promising approach using hardware assisted virtualization is presented, that by using the virtualization hypervisor will handle encryption and decryption of application memory as well as preforming security checks for interaction with the host operating system, This can potentially provide a safe environment that can be used for video games while maintaining the required performance.

Contents

List of Figures

1 Introduction

Video games is a large and growing entertainment market, with a total investment as of 2016 of over 91 billion USD[1] spread across different markets within the game industry, for several of whom where information security and players finding ways to gain an unfair advantage is causing problems. Cheating players takes away from the enjoyment of game communities themselves both for casual players and the big market that is based around watching video game tournaments and other videos through channels such as YouTube and Twhich.

Another market where cheating have a more direct impact is on competitive Esports and several professional players have been caught using third party software in order to cheat[2]. Precautions to prevent cheating is already being brought to the same level as conventional sports such as Drug testing being introduced after professional players admitted to using performance enhancing drugs when competing[3]. However software assisted cheating requires a different approach from what found in conventional sports and preventing this kind of cheating is an emerging problem that needs to be solved in order to grow Esports to its full potential.

Another large market gaining large investments the last couple of years is the Virtual Reality market, this is a market where the video game industry is leading the evolution of the technologies used. Virtual reality technologies are also gaining interests outside of the game industry, opening possibilities for uses such as long distance virtual business meetings replacing video conferences and long distance travel for face to face meetings[4]. However in order for such solutions to be feasible the security of the information shared through such technology need to hold the same level of confidentiality as the already used networks for sharing business information through Video conferences and file transfers.

1.1 The problem, and cheating as a business market

The issue with video games being susceptible to cheating and how much this is effecting the gaming market is an increasing problem. As more money is brought into the industry both in terms of entertainment investments and gambling but also through a massive increase in the price pools for winning Esports tournaments now reaching upwards of several million dollars [1]. All of this makes the incentives for winning higher, which again and provides the grounds for a million dollar business market based around producing, providing and using third party software for cheating in video games to grow.

The cheating business market is made up from a series of legitimate registered tax paying companies. These businesses range from one person companies making half to a million dollars a year, to larger companies with professional management and teams making around 1,5 million dollars a year. The global business market is estimated to be at about 100 Million USD as of 2016[5], making this a big lucrative market and making preventing cheating a larger issue than what can be solved by implementing minor difficulties.

The definitions used in this paper will be based on a presentation by Simon Allaeys and Aarni Rautava[2]. They held a talk during the Steam Developer Days 2016 and they work for the company Easy Anti-Cheat, who works on third party anti cheat software which is the current way to combat cheating in video games, more about the current technologies used for this will be presented in the Literature Review part of this paper. In their talk they separated the business of cheating into three categories, with profiles describing the skill set and motivation of the different people involved. These categories and profiles will be useful to have in mind when developing a video game and when considering the different approaches to preventing cheaters in a video game.

1.1.1 Cheaters

Cheaters are the players playing the game with an unfair advantage, there are many forms of cheating everything from drugs to match fixing similar to any other sport or competition. However within the context of this paper cheating will refer to people playing video games using third party software in order to make games easier or give the player access additional information that is hidden to other honest players. There are several reasons players choose to use these third party tools to cheat and we can split them up into cheater profiles.

Greifers

Griefers are cheaters mainly motivated by wanting to ruin the game for the other players and the and powerful feeling they get from breaking the game rules in obvious and openly visible ways. This is a minority group of cheaters and as they are

[1]Largest Overall Prize Pools in eSports: https://www.esportsearnings.com/tournaments

[2]Steam Dev Days: Anti-Cheat for Multiplayer Games https://www.youtube.com/watch?v=hI7V60r7Jco

easily detected by administrators and removed from the game. These people will buy several copies of games during sales or create multiple game accounts in order to keep playing a game on their own premises. This kind of cheaters can cause a large amount of damage to the community of a game and will cause honest players to stop playing if the game seems unfair and if these griefers are not taken care of effectively and therefore require large investments from the game developer.

Casual cheaters

The largest group of cheaters are the casual cheating players, these players cheat simply to make the game easier for themselves. If they feel the game is to difficult or that other players preform better then them they will find and use cheats to stay competitive or in order to get an edge on their fellow players. Their aim is not necessarily to ruin the game for others but can do this in order to stay on the same skill level as their friends. Such players generally don't feel any regret about cheating and will be likely to cheat in most games they play if possible.

Achievers

The achievers are the players who wants to be the best at and win everything they do. These cheaters will be highly secretive about using cheats in order to do so and will play in a way as to not make their cheating obvious to others making these the most difficult to catch. Its this group of cheater is likely to try entering tournaments for monetary as well as social gains with the use of cheats, something that can cause problems for video games as a sport if viewers never can be sure if the players are playing fair or not.

Vigilante or followers

If griefers or the other types of cheaters become too prominent in a gaming community rather than stop playing some people choose to pick up cheats for themselves in order to level the playing field or to "punish" the people who cheated against them. Making the game more a case of who are able to cheat the most without being caught. If a game reaches this state where cheaters are playing against cheaters it will prevent most new players from purchasing and playing. The game will receive negative reviews and that way removing most of the future profitability of a game, as well as giving a negative view for the game developer itself.

1.1.2 Hackers

Hackers are the people making the third party software that allows a player to cheat. These are the people finding and exploiting weaknesses in the software in order to get access to what they want. A hacker is a known term within information security but in the context of this paper we are focusing on the hackers who focus on video games. Their motivation for this as well as their skill set and background can be separated into three profiles.

Scripters

The vast majority that fits into the hacking group are known as scripters, they are often players with some technical background. The way they develop cheating software is typically by looking up code online, then editing and fitting this to the game they are interested in mostly through trial and error. The motivation of this group is mostly curiosity and a wish to test their skills in order to see what they are able to accomplish.

As major game engines become more common simple code used for cheating in one game can be easily modified to work for another game built using the same game engine. And as game engines release their source code as open source these cheats have the potential to become more advanced as deeper knowledge into the inner workings of a game become more easily accessible. However, most of the cheats created by this group are typically aimed to do small an simple tasks.

Researchers

The researcher group is people a bit similar to the scripters where the motivation is to see whats possible. However these are typically highly skilled individuals that will dig into and reverse engineer either a game or the anti cheat software itself in order to find security holes.

The researcher group wont necessarily create software for cheating or if they do it will be simple proof of concepts, rather than commercially viable cheating software. These finds are then typically released publicly just as much for the developer to find and fix as for other hackers to use.

Senior hackers

This group of hackers are typically payed and professional reverse engineers or programmers hired to make feature rich and easy to use cheating software. This group will have a professional approach to their software keeping it updated in order to avoid bugs and more importantly making sure it is not detected by a games cheat detection countermeasures.

1.1.3 Providers

There are a number of different ways cheaters can find the cheating software they use. As they typically do not develop these cheats themselves is a large market in distributing this kind of software. Sharing cheating software is done through different cheating communities, some more locked down and controlled than others.

Open communities

Open cheating communities are typically where both cheaters and game hackers will start out. On open forums and web pages free and open source cheating software will be shared and easily found through a web search. The cheating software here are often outdated and anti cheating software will typically quickly update their software to detect and lock accounts that are detected with openly available cheating software. The motivation for sharing cheating software in such an open and available form is to generate income from add revenue and lead players further into the cheating community

Payed subscription based communities

Another way of spreading cheating software are through payed subscription services. These services are also openly available through a web search but does require paying a monthly sum of money in order to be accessed, and that way restricting access for game developers who just want to protect their games from cheaters.

The software shared here are typically of a professional quality with nice well developed user interfaces as well as support for paying subscribers. The software are regularly updated in order to remain hidden from anti cheat countermeasures and are generally hard to prevent with today's approaches to prevent cheating.

Closed communities

The last kind of the cheating community is the closed communities where access is reputation based. These communities have been known for requiring video chat interviews and copies of official identification before potential members is given accepted into the community. All of this in order to prevent members leaking information and causing the cheats shared here to be detected

These communities are known to offer private personal cheats sold only to a very limited number of people at prices of 40USD a month and upwards. As well as other exclusive cheats sold to a single person or a competitive team for 500 to 1000USD who are supposed to be entirely undetectable. These cheats are typically small simple cheats but are hidden in ways allowing some of these to be installed and used for local tournaments, where the computers are provided by the host of the competition and the security is generally very high.

Examples of this is cheats installed through Steam workshop[2] simply by signing into a players account. As well as suspicions of cheats hidden in the system drivers for accessories such as mouse and keyboards where players in tournaments are allowed to bring their own accessories to the competition.

1.2 Forms of cheating

Cheating in video games can take many forms based on what the cheater want to accomplish, some cheating is more damaging to a video game community than others and in some game genres some areas of cheating is more accepted than others. The different ways of cheating also exploit different parts of the system and relies on being able to gain access to different information from within the game. We can split the cheating into categories that will be useful when discussing potential solutions for preventing them, as there is likely blanket solution to prevent all of these.

1.2.1 Exploits

The exploits category of cheating refers to players finding ways to make the game behave in ways that was not intended. This is most often simply by finding design errors in the game, this can be anything from an invisible wall left behind by a game developer that allows a player to jump up and reach a position that was not intended to be accessible, or players figuring out that they can use the physics engine or a game ability to break the gravity in the game to allow them to fly. The general theme for this category is that the cheater is not manipulating the games code itself and there is no real hacking involved but rather abusing game logic errors in ways not intended by the developer, and therefore is not that much of a focus for this paper rather than it being down to the developer to do conclusive testing and fix exploits as they show up.

1.2.2 Automation

Automation is another category where the cheater does not necessarily dig into the games code in order to cheat, there are more innocent ways to do automation that are generally not as frowned upon by the game community. This can be things like using keyboard shortcuts to have one button that activates all your abilities, or one button to write predefined message in the chat. This is generally done in order to save the player time or to automate repetitive tasks for some game. The capability of doing most of these things are now a general feature in a lot of gaming accessory software such as the driver for a mouse or keyboard and is in most cases accepted within the gaming community, and therefor not as much a priority in this paper.

There are however other more advanced ways of automation that relies on third party software digging into the code, memory, reading colors on the screen or the network packages to work. An example for this is typical Aim bots, where third party software will read the position of an enemy from the internal memory of the game and then automatically move the mouse to target this enemy and shoot, without requiring the player to do anything. It can work by listening to network packages waiting for a particular enemy to spawn then automatically moving the player to a position and killing it allowing a player to leave the game playing itself for extensive amounts of time and gaining rewards without having to play the game himself.

1.2.3 Overlays

Game overlays works much the same way as the previously mentioned more advanced form of automation. Except that rather than doing actions from the player, it will overlay information in over the game to give the cheating player access to more information than that which was intended by the game developer. This can be different visibility hacks where it will overlay enemy positions onto the screen through walls, or make them show up on a map. It can also be information through a message or sound to let the player know if an important item is available or a rare or important enemy have spawned into the world.

1.2.4 State Manipulation

The most intrusive way of cheating in video games is through state manipulation, where rather than just read and abuse information available from the game in unintended ways this will change how the game works for the cheating players. This can through third party software edit the game while running into reducing or disabling the gravity or game physics for the cheater allowing him to fly, jump higher or move through walls. It can also be used in less obvious ways such as changing the speed the player moves or teleport the player around the map by simply changing its positional values.

2 Literature Review

The available literature on video games and cheating in them are currently somewhat limited, and most discussions on these subjects are typically found on developer forums, blog posts and web pages as well as videos from developer conferences. Therefore the information reviewed in this section will comes from a variety of these sources as well as a few scientific papers discussing these subjects.

2.1 Third party anti-cheat software

Cheating in video games are a large problem en certain genres and even with the implementation of Network encryption there are still several other attack vectors left open. Making the use and need for other cheating countermeasures being necessary and widely used. The approach several approaches used by current Anti-Cheat software they are generally either looking at the statistics and data for players, searching through things like high score lists and point scores to detect impossibly high scores or looking at the movement of a player in order to detect if a distance moved is possible within a certain amount of time. Another approach is by logging statistics for each individual player, figuring out their play style and skill level and look out for large play style changes or abnormally large jumps in skill level.

This approach works well against very obvious cheating and players who don't try to hide the fact that they are using cheating software, there is however is some games quite a large overlap where highly skilled players and subtly cheating players generate the same sets of statistics, something that can lead to either false positives or cheaters being able to trick the system, there is also a problem with the logging of play styles where if a player simply have a friend over playing and playing the game on their computer could lead to a false positive if the skill level or play style is drastically different[5].

Another approach being used is similar to the one used by anti virus systems where signatures, where the anti cheat provider assemble a list hash signatures of known cheating software, it will then scan the computer of everyone playing the game in order to find any of these known signatures. This approach pretty much ensures no false positives, but it does have the drawback of having to know about all third party cheating software, and even a minimal change to a cheat will change the signature allowing it to pass by unnoticed by the anti cheat system.

A different approach is a heuristics based approach that aims to detect cheating by recognizing known behavioural patterns, and patterns in system calls made while the game is running. These systems don't search for specific cheating software but rather look at how the game engine behaves and where different system calls land, then comparing this to how other known cheating software works and if it find reasonable similarities files and logs will be collected from the players computer and then investigated by the anti cheat system. This ensures that most common cheats can be detected with a drawback of some computer viruses will behave similarly to cheating software so a player can risk getting locked out of their games for an unrelated virus infecting their system.

Most of the anti cheat systems is also offers a shared cheater database, meaning that if there are several games tied to the same user account system a player caught cheating in one video game using an anti cheat system will be prevented from playing competitively in any of the games tied to the cheaters account. The intention of this is to make the risk of being caught and having to replenish and buy all the users games over again in some cases being enough to prevent players from cheating in the first place.

There have unfortunately been incidents where these third party anti cheating have proved less than trustworthy, and news broke that a developers in a large competitive gaming network had included malware used for Bitcoin Mining in their anti cheat software[6]. A general issue with the current approach to anti cheating is that in order to prevent cheating anyone playing a competitive game have to provide this third party anti cheat software with access and privileges to scan and trace all the system calls made while a game is running, as well as the rights to send the information gathered to the Anti-Cheat companies servers for analysis.

Considering that most machines used for video games are private home computers, and the information scanned and gathered not necessarily being related to the video game the user have installer, this can be viewed as rather large breach of privacy, as information sent to the anti cheating server may contain personal information and files. This have in some cases been getting a lot of attention on community forums[7] and is generally giving bad press to the companies that implements these cheating countermeasures. This is costing companies potential sales from people who wont allow this anti cheat software to run on their computer making game developers having to choose between anti cheat security or the potential sale loss.

2.2 Current Information security used in video games and game engines

The current technology used to add security to video games currently, mostly focused around preventing illegal downloads and protecting assets used in the game from theft. This security is implemented differently dependent on how and what the aim is to protect. A lot of games use account based access systems to make sure that the game is played by a someone who have acquired the game legally. An for these account authentication systems the general information security guidelines is are implemented and followed with encrypted connection between the client machine and the game companies account server as to not transfer information in plain text over the network. Package encryption to protect the art assets as well as the program files for video games are also something that is used in some cases.

Security for the run time information on the game however is rarely used, the same goes for securing the continuous data stream transferred over the network between the host game server and the client. There is starting to show up network encryption in some areas and game engines[8], This typically implements a combination of RSA for key exchange and AES encryption to contiguously protect the game data and effectively limiting the some potential attack vectors used in cheating software. These technologies are not activated by default but can be used by the game developer if they choose.

For other game engines such as the Unreal engine there are community developed network encryption plugins[9] [10] that can be attached to specific variable types so with some vigilant use a game developer can secure the most confidential information against being openly available on the network. and by doing so they will be sacrificing some potential performance for security.

2.3 Hiding techniques for cheating software

Since most anti-cheating approaches used for games currently are based around searching for software and files on the players computer that are recognized as known cheating software it has become necessary for cheating software to hide their influence and presence on the cheating players system. It is also a priority to make cheating software obscure in order to make reverse engineering harder in order to make it harder for an anti cheat company to break apart the cheat and learn their scanners to recognize them.

To avoid being detected and to work cheating software will usually go through the operating systems kernel mode directly accessing a computer systems memory and influencing system interrupts and service hooks in order to make a game easier for the cheater. The cheating software itself will usually utilize virtual address descriptor hiding, by hooking their memory into other legitimate system services or applications or the game itself in order to not be detected by a anti-cheat scanner.

Advanced and expensive private cheats will often also implement a back end system observing and logging the usage of a cheat, aimed to detect suspicious activity such as if a cheat have been ran 3 times consecutively, is shared and run simultaneously on several systems or if there is a debugger running simultaneously on the system running the cheat. This is done in part in order to be able to lock down the cheating software in case someone is trying to reverse engineer it, but also to protect the hackers and providers creating these cheats from users sharing their software for free to people who have not payed for the software.

2.4 Other approaches currently used to prevent cheating

There are no perfect way to protect and deal with cheating in video games. Typical anti cheat approaches makes cheating harder but not impossible and if the potential reward in doing so cheats will be developed. There are a number of decisions and limitations placed on the game developer in order to prevent cheating in their game. The approaches that have proved the most efficient does not rely on additional security to work but rather designing the game around making cheats less valuable or possible.

The main part of these approaches is by having a trusted server making a number of decisions, such as the server making the decisions about whether or not two players can see each other, and then only sharing the information about other players positions when the server decides that they should be able to see each other. This will limit the usefulness of cheats such as radar/map hacks where enemies will be visible on the cheating players map, and could also potentially help prevent wall hacks.

The issue with having the server making all decisions such as this is firstly that it requires a secure trusted server. Another is that this server needs to be substantially more powerful as it needs to do all the calculations required for the game to run. The biggest challenge with using this approach however is the connection speed, all the information need to travel through the network and will in some cases cause fast paced and particularly VR games feel slow and unnatural and therefore a completely authoritative server is not always a viable solution.

There are also some other options for cheat detection that can be implemented by a game developer that makes cheating more difficult and might catch out some of the more casual cheaters using basic cheating software that have not been targeted towards one game in particular. This can be done by implementing code and variable obfuscation, changing the standardized variables used to store important information such as integers, floats and Boolean variables into proprietary variables that are not recognized easily by a cheater. This will require the cheaters to obtain more inn depth knowledge about how the game works before they are able to cheat.

Other approaches include creating so called "Mouse Trap" variables, this will be variables that store information that seemingly is relevant to gameplay such as having an extra variable tracking a players health as a standard integer value while having a different obscured value that is the real variable tracking the same information. Then implementing regular checks to make sure this mouse trap variable matches the actual variable and if a change of the trap variable is detected you are able to label the player as a cheater and react accordingly. There are typicaly plugins for easy implementation for these solutions available for popular game engines such as Unity[11] and Unreal Engine[12] as some examples.

3 Methods

3.1 Cheating software, How it works

This section is partially based on the research and testing done in a specialization project conducted before this one. The paper on this has been included in this paper as Appendix A: Finding insecurities that can be used for cheating in online games. Further information on the testing and research behind this section can be found there, however this section will summarize the most important aspects that is used as a basis for this paper. Some further testing focused on the networking side and what information is available through listening to network communications was conducted as a part of this paper and will also be covered here.

For this section the focus is on technical cheats that works by accessing or changing information used running the game, rather than the exploitation of bugs that are left in game by developer. Automation where a player uses keyboard shortcuts and macros to preform functions not directly connected to a game or game engine is also not a focus, as this kind of automation is generally accepted within the game community. Therefore the priority of this paper is looking at ways to prevent other more damaging types of cheating.

3.1.1 Overlays

Overlays is as mentioned in the introduction a way for a cheater to be shown more information in a game than what was intended by the game developers, the implementations can work in a couple different ways depending on the kind of game but generally the information gathered by the cheating software will shown overlaid onto the screen over the video game graphics or by manipulating the game graphics themselves into showing more information than normal. The most common information this aims to provide is the position of the enemy players, either overlaid on a map or making them visible through walls.

The information used to be displayed with overlay cheats are gathered from the memory used by the running game application. Full information on how this is typically done can be found in Appendix A, but in short the cheater will launch the game through a cheat application hooking it into the game's running processes and by going through kernel mode gain access to the random access memory used by the application. Then the cheat software will narrow down the memory used by the application in order to find the variables containing interesting information such as player positions, health amount and so on by doing a number of searches based on whats happening within the game.

To narrow down the available memory variables the cheater will have to manipulate the game, an example would be record all available variables while not moving in the game, then narrow it down by removing all variables that changed while players are standing still. Then when players start moving narrow it down again by removing variables not changing and continuing to do so until the desired variables are found. This can be done with multipurpose cheating software and made to work for a lot of different video games. Other more advanced cheats made for a game in particular will be able to recognize the important variables based on knowledge about the game itself removing a lot of trial and error but ultimately will work the same way with reading the variables used by the game.

Another slightly less used way of gathering information for an overlay cheat is through listening to networking packages received as these will contain updated information on how other players are moving and where they are within a game arena. This approach is harder to detect by anti cheat measures that look for unknown software tied into the game processes as with this approach the cheat requires no direct interaction with the game itself.

When the cheating software have found the information required it will typically overlay it directly onto the screen of the cheater and draw enemies at the correct position either on a in game map or make the game drawn enemies in the game world even when there are walls separating the cheater and the other players.

3.1.2 Automation

Automation depends on the same basic approach to accessing information from a video game process. For these cheats it is typically gathered from scanning network packets for information or for packages containing a particular pieces of information. This can also be implemented to used information from the application memory as explained in the Overlays section, the difference being that rather than showing the cheating player the information for the player to act on automation cheats will simulate keyboard or mouse input to the game on the cheaters behalf. This can be again by reading enemy player positions from memory or a networking package, then it will provide the mouse input required to target any enemies within range and click the shoot button.

Some automation cheats will also use screen recording as a way to gather information, an example of how this works would be to search for the color of an enemy player on screen and use where it is detected on screen to provide simulated mouse input to target and shoot at an enemy player. This will generally react way faster than any honest player would be able to react. To get this to work some cheats require game resources and textures to be edited, such as making the textures of enemy player completely red in order for them to stand out from the surroundings[13].

3.1.3 State Manipulation

While the other cheats covered here are mainly focused on reading information about the game and use this information to give a player an unfair advantage they don't change the game or how the game is played. Sate manipulation however is where the cheater accesses the information similar to the previously mentioned ways either through accessing network packages or directly in the memory of the cheating players application. This can be done with different goals in mind, if a player wants to become invincible finding the variable for the players health points and simply locking this will prevent it from being updated when the player takes damage. Due to this being too easy for a cheater however a lot of the more important variables is handled by the server.

Other more relevant cases where state manipulation is used to cheat is for games where the developers will leave debugging or spectating game modes in the game code with a simple activation check at the start of the game, deciding if they are supposed to be active or not. Manipulating this after a cheating player have connected to a server however can allow the cheater to make his game run in one of these modes mid game giving him access to flying or making him able to view all players though walls similar to how someone spectating the game would view it. Therefore in popular Esport focused games where the viewer experience is a focus there is a possibility that they can be abused by a cheater providing him with a nice already polished way of cheating using the code already within the game.

Other ways of cheating using state manipulation is by redirecting system calls that a game will use to either external libraries or to the operating system as a way to implement their own code into the game. This will in theory allow the cheater to send custom information to the game server or make other feature rich cheats where he can get his game client to change most rules not enforced and validated by the server.

3.2 Concept game

Previously in the pre-project for this paper a concept game was developed in Unity, as this is one of the most popular game engines in use today. However in the time between the pre-project and the research for this paper began Unreal Engine another popular game engine made its source code open source. As this would be a big advantage for the researched planned for this project a new and basic multiplayer concept game was developed. This was in part to do more analysis of what security measures where implemented and again to see what information was available both through the application memory as well as capturing and analyzing network packages. The other reason for developing this concept game again in an open source engine was in order to be able to implement potential information security measures directly on an engine level and use this concept game to test the effectiveness of potential solutions as well as measuring performance loss caused by added security.

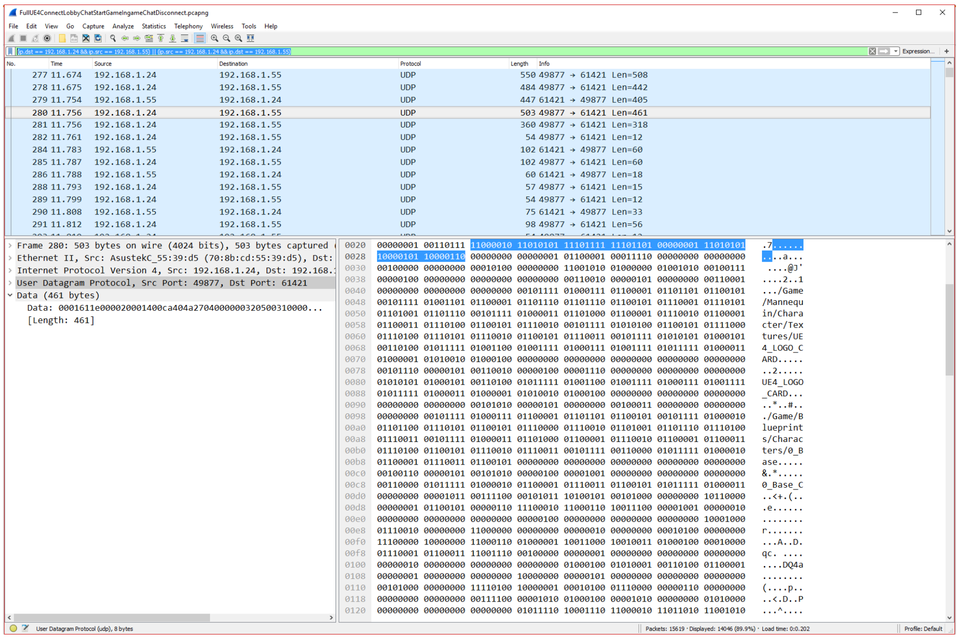
The concept game included shared variables and texture files for different player characters that would be replicated between server and client as well as variables for player positioning around in a game world. A player selected user image was also implemented as a way of testing how files would be communicated over the network. An in game text chat was also added into the concept game in order to test how these messages shared within the game would appear going over the network between server and client. These last tests would be interesting in order to see how viable a the basic chat solution of a game engine would be in cases where security would be more a concern such as a virtual reality business meeting. And what precautions that would be required in order to make this communication secure from potential man in the middle attacks.

3.3 Exploring the information channels exploited by Cheating software

From the research on how cheats generally work and where they get the information they use in order to function it is clearly two approaches that are being used. That is the information stored in the application memory, and the information transferred over the network between clients and the game server. A more in depth view of how information is accessed from the application memory can be found in Appendix A where a concept game was created and the commonly used Cheat Engine is used to locate, lock and edit the application memory while the concept game is running to show how a potential cheater might manipulate the game through this approach.

As we already have the results from the memory manipulation from the Unity test project the focus this time was to analyze what information that was openly available through the networking packages sent between client and server. The new Unreal Engine based concept game was used for the analysis and network packages was captured on both the server and client in order to see what information was available using the open source packet analyzer Wireshark. The information found on both clients and the server was similar so only one of the logs where analyzed deeper for information

When analyzing the networking packages a challenge was the fact that unreal engine uses non primitive data types for a lot if its operations rather than the typically recognized integers and strings, making recognizing messages and information stored in these using wire shark non trivial. However other information such as file paths and names for the player images and character textures was openly transferred as String variables as shown in bottom right of the Wireshark Capture.


FullUE4ConnectLobbyChatStartGameIngameChatDisconnect.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
(ip.dst == 192.168.1.24 && ip.src == 192.168.1.55) || (ip.src == 192.168.1.24 && ip.dst == 192.168.1.55)
Expression...
User Datagram Protocol (udp), 8 bytes
Packets: 15619 · Displayed: 14046 (89.9%) · Load time: 0:0.202
Profile: Default

```
No.  Time    Source          Destination     Protocol  Length Info
277 11.674   192.168.1.24    192.168.1.55    UDP       550 49877 → 61421 Len=508
278 11.675   192.168.1.24    192.168.1.55    UDP       484 49877 → 61421 Len=442
279 11.754   192.168.1.55    192.168.1.24    UDP       447 61421 → 49877 Len=405
280 11.756   192.168.1.24    192.168.1.55    UDP       503 49877 → 61421 Len=461
281 11.756   192.168.1.24    192.168.1.55    UDP       360 49877 → 61421 Len=318
282 11.761   192.168.1.24    192.168.1.55    UDP        54 49877 → 61421 Len=12
284 11.783   192.168.1.55    192.168.1.24    UDP       102 61421 → 49877 Len=60
285 11.787   192.168.1.24    192.168.1.55    UDP       102 49877 → 61421 Len=60
286 11.788   192.168.1.55    192.168.1.24    UDP        60 61421 → 49877 Len=18
288 11.793   192.168.1.24    192.168.1.55    UDP        57 49877 → 61421 Len=15
289 11.799   192.168.1.24    192.168.1.55    UDP        54 49877 → 61421 Len=12
290 11.808   192.168.1.55    192.168.1.24    UDP        75 61421 → 49877 Len=33
291 11.812   192.168.1.24    192.168.1.55    UDP        98 49877 → 61421 Len=56
```

```
> Frame 280: 503 bytes on wire (4024 bits), 503 bytes captured
> Ethernet II, Src: AsustekC_55:39:d5 (70:8b:cd:55:39:d5), Dst:
> Internet Protocol Version 4, Src: 192.168.1.24, Dst: 192.168.
> User Datagram Protocol, Src Port: 49877, Dst Port: 61421
v Data (461 bytes)
    Data: 0001611e000020001400ca404a270400000003205003100000...
    [Length: 461]
```

```
0020  00000001 00110111 11000010 11010101 11101111 11101101 00000001 11010101   .7......
0028  10000101 10000110 00000000 00000001 01100001 00011110 00000000 00000000   ....a...
0030  00100000 00000000 00010100 00000000 11001010 01000000 01001010 00100111    ....@J'
0038  00000100 00000000 00000000 00000000 00110010 00000101 00000000 00110001   ....2..1
0040  00000000 00000000 00000000 00101111 01000111 01100001 01101101 01100101   .../Game
0048  00101111 01001101 01100001 01101110 01101110 01100101 01110001 01110101   /Mannequ
0050  01101001 01101110 00101111 01000011 01101000 01100001 01110010 01100001   in/Chara
0058  01100011 01110100 01100101 01110010 00101111 01010100 01100101 01111000   cter/Tex
0060  01110100 01110101 01110010 01100101 01110011 00101111 01010101 01000101   tures/UE
0068  00110100 01011111 01001100 01001111 01000111 01001111 01011111 01000011   4_LOGO_C
0070  01000001 01010010 01000100 00000000 00000000 00000000 00000000 00000000   ARD.....
0078  00101110 00000101 00110010 00000100 00001110 00000000 00000000 00000000   ..2.....
0080  01010101 01000101 00110100 01011111 01001100 01001111 01000111 01001111   UE4_LOGO
0088  01011111 01000011 01000001 01010010 01000100 00000000 00000000 00000000   _CARD...
0090  00000000 00000000 00101010 00000101 00000000 00100011 00000000 00000000   ..*..#..
0098  00000000 00101111 01000111 01100001 01101101 01100101 00101111 01000010   ./Game/B
00a0  01101100 01110101 01100101 01110000 01110010 01101001 01101110 01110100   lueprint
00a8  01110011 00101111 01000011 01101000 01100001 01110010 01100001 01100011   s/Charac
00b0  01110100 01100101 01110010 01110011 00101111 00110000 01011111 01000010   ters/0_B
00b8  01100001 01110011 01100101 00000000 00000000 00000000 00000000 00000000   ase.....
00c0  00100110 00000101 00101010 00000100 00001001 00000000 00000000 00000000   &.*.....
00c8  00110000 01011111 01000010 01100001 01110011 01100101 01011111 01000011   0_Base_C
00d0  00000000 00001011 00111100 00101011 10100101 00101000 00000000 10110000   ..<+.(..
00d8  00000001 01100101 00000110 11100010 11000110 10011100 00001001 00000010   .e......
00e0  00000000 00000000 00000000 00000100 00000000 00000000 00000000 10001000   ........
00e8  01110010 00000000 11000000 00000000 00000010 00000000 00010100 00000000   r.......
00f0  11100000 10000000 11000110 01000001 10011000 10010011 01000100 00010000   ...A..D.
00f8  01110001 01100011 11001110 00100000 00000001 00000000 00000000 00000000   qc. ....
0100  00000010 00000000 00000000 00000000 01000100 01010001 00110100 01100001   ....DQ4a
0108  00000001 00000000 00000000 10000000 00000101 00000000 00000000 00000000   ........
0110  00101000 00000000 11110100 10000001 00010100 01110000 00000110 00000000   (....p..
0118  00000000 00000000 00111100 00001010 01000100 00001010 00000000 01010000   ..<.D..P
0120  00000000 00000000 00000000 01011110 10001110 11000010 11011010 11001010   ...^....
```

Figure 1: Capture from wireshark showing file information in clear text sent between server and clients. In our case the "UE 4 LOGO CARD" being the user image selected by a client and the "0 Base" being the player character selected.

In the Wireshark capture you can see references to the image files and file paths transmitted in clear text over the network. Therefore it is reasonable to believe that by default there are no encryption used for the network layer of Unreal Engine, The information shared through the in game text chat is also likely available in these packages but are not as easily as easily read using wire shark as this only recognizes the primitive data types. Software specifically searching for the data types used by the unreal engine however would likely be able to find and read the information shared by client and server openly.

3.4 Areas of focus

From the research on cheats and the software used for cheating there is generally two vulnerabilities that are abused to make the majority of cheats work. Therefore information security solutions who handle networking security and who secure application memory will be the main area of research. The goal is finding how security in these areas have been dealt with in business markets where security have been a major concern from the ground up. These solutions will be reviewed based on how they are implemented, what additional security they provide in a video game setting, as well as what requirements they need to be effective. This is required in order to judge whether or not they can be of use in an environment where performance and the responsiveness of the software is vital, and where any sort of delay and required processing time will be a trade off from the more marketable graphics, gameplay and system requirements for a video game.

3.5 Memory Focused security

Memory protection was the area that received the most focus while working on this project as this is the vulnerability that are most easily and commonly used by cheaters while also being the area where there are less options available for current game developers to protect their games. This focus did however prove a challenge as there where no simple ways of implementing and testing potential solutions. And the solutions commonly used to provide memory security to applications relies on functionality provided by the operating system or hardware support to work. Some approaches that have been considered however might prove useful if a game engine where to be developed around them, something that potentially could lead to significantly higher resistance against cheating than any other engine on the market.

3.5.1 Sandboxing

In computer security a sandbox is a mechanism used for separating running software from the rest of a computer system. The goal here is typically to be able to run unverified and therefore not trusted software with a layer of protection to make it unable to access processes outside of the sandbox, and that way prevent the operating system or other applications becoming infected. This is not directly useful in preventing cheating in video games as running a game within a sandbox wont protect it from the outside as the cheater have control of the operating system.

However in commercialized cloud computing there have been research done Two-Way Sandboxing where both the application is protected from the untrusted platform running it, while at the same time the operating system and other applications on the host system is protected from untrusted applications[14]. The approach discussed for two way sandboxing is by using the virtualization hypervisor to encrypt and isolate the application memory. Working at a lower level than the host operating system all the information that will be passed through the operating system will be encrypted thus significantly reducing the attack surface of any game running within this sandbox.

The difficulty with implementing this solution is with providing a game running within the sandbox access to necessary graphical driver for supporting some of the more advanced technologies while still keeping the interfacing between operating system and game code to a minimal and manageable level. This might require game developers or the game engine to manage these interfaces together with hardware developers to make the necessary calls available without them having to interact in clear text through the operating system, without sacrificing too much of the systems performance.

3.5.2 Hardware assisted virtualization

Most modern computers have hardware to assist with virtualization built in, that being especially the virtualization hypervisor, made to provide virtual operating systems a more direct and efficient access to the hardware without having to route all system calls through the host operating system on the machine. The previously suggested sandboxing approach was also based around using the hypervisor both to provide application security from the host operating system as well as protecting the host from anything running within the sandbox making up two layers of security.

As performance is important it might be sufficient from a game developers perspective to only focus the protection on the video game itself with outward facing security. There have been a number of papers and research done in the area, focused on protecting a virtual machine typically running a web server from the system running it, as in shared systems there will typically be several untrusted virtual machines running on he same hardware it is a priority to protect each individual virtual machine from potential infections on the others.

When looking at performance metrics done on implementations of hypervisor protected virtual machines tasks requiring processing power but limited memory usage the performance hit of this security measure is near inconsequential[15]. However for tasks where transferring and processing large amount of data is required the resource consumption is significantly higher to upwards of a 50% reduction in performance for a pure file transfer tasks[16].

A large benefit of a virtualization approach is that it could be able to run most software without modification of either the host operating system or the game application itself, its entire development would be focused on a virutalization layer. In the references found[17] this layer, referred to as a "Shim" is responsible for managing all the interactions between the clacked application and the operating system. This allows for relatively complex OS specific system calls and operations to be handled without the need of entering kernel mode through the host operating system, and provides a convenient place to add custom or OS-specific functionality without modifying the host operating system.

The shim memory consists of both encrypted and unencrypted memory regions providing the potential for increased performance if implemented in a way where a video games heavier files such as sound and textures are stored and handled outside of the protected memory while keeping program critical memory protected against access from the the outside. This have the potential of making such an approach viable in the performance heavy environment as it can keep the size of the allocated protected memory to a minimum and thus avoiding unnecessary encryption and decryption of the larger files required in modern video games.

In the Overshadow[17] project the shim are made to handle nearly all system calls supported by the Linux 2.6 kernel interface, and thus was able to run a large amount of Linux software without requiring the applications to be modified. How an implementation developing such and implementation for windows based applications however could prove a challenge, but in today's market would be necessary to reach the majority of video game developers. However in the future this market could potentially change to where Linux will increase in popularity, with big video game companies such as Valve known for the dominant Steam gaming platform investing big in making Linux a free and more openly available gaming platform through the development of Steam OS[18].

For an implementation of such a system to be deployed together with a video game however would place a lot of additional development time for a developer as it would essentially be shipping a video game bundled together with a virtual machine manager as well as a proprietary operating system to handle the communication between the application and the client computer hardware. This is not realistic from the view of a game developer, however it could be a potential approach and motivation to develop a game engine that offer easy game development similar to that of Unreal engine or the Unity engine but bundled with the proprietary operating system and virtualization manager. This even just being an option for deploying the finished product with the benefit of added security. Leaving the only decisions the game developer have to consider during the game development being what information should be placed in the secured and encrypted part of memory and what can be safely used directly in order to maintain performance at a reasonable level.

Another alternative for developing and deploying such a solution could be in the form of a game distribution platform, so that rather than just being a library, storefront, and community hub it would also provide a virtualization layer for running the games sold on the platform. This would be a benefit for both game developers and customers on the platform being able to ensure increased resistance against cheaters being a large potential selling point both for players looking for a fair game and developers who would rather focus on improving their games than dealing with anti cheating measures after deployment.

3.6 Network security approaches

As the second most common vulnerability used for cheating in video games being the ability to capture and read networking packages, this would also be an important area of focus in order to make video games more resilient against cheating. This is an area that have received a lot more attention generally when discussing security outside of the game industry, this is due to the system running an application and the user with access to that system is generally considered to be trusted while potential attackers will have to attack from the outside.

After starting research on the topic of networking security in video games there are actually some networking security available for use in video games in the most common game engines. Most of witch is focused around account management and for transferring user passwords and account information between different sign inn systems and the account server rather than use for the game information between clients and the game server.

The way this is implemented is so that the game designer can assign certain information to be handled securely typically through the HTTPS standard for communication with web servers. There are also implementations that use RSA for asymmetric key exchange and then AES either as a block cipher or a deviation of AES used as a stream cipher. In the closed source unity engine it is available for use by any game developers who chose to activate full network security for their game covering all network communications by default and functionality for turning it of for specific packages if desired. This solution does however not preform any kind of authentication of the server.

As for the unreal Game engine there are code for encryption within the engine that according to the limited information available on the subject are used by some online subsystems but are not currently available for use by game developers out of the box. The engine source code does currently contain code both for asymmetric encryption using RSA, as well as symmetric encryption through AES, BlowFish, TwoFish as well as an XOR based stream and block encryption[19]. It is however only recently other game engines have began implementing network encryption as a standard and this could potentially be a work in progress to soon be able to provide for game developers.

In addition to encryption network encryption supported by default by game engines there are third party plugin solutions offered for the most common game engines. This will give game developers options and the possibility to help protect their games against certain approaches to cheating. And this trend is hopefully something that only will gain more support and features going forwards, and making game engines more suitable for potential the business solutions making use of Virtual reality features, where network encryption is the major concern to protect the information shared in virtual business meetings from being vulnerable to man in the middle attacks.

4 Results

The results found in researching security solutions for a video game environment is that the challenge of protecting against a cheating players differs fundamentally from most other security concerned applications in that malicious user already have the full access to both the hardware and administrator privileges within the operating system. This would be comparable to a root kit on the operating system level. Most information security is aimed at stopping malicious software before it can reach such a critical access level within a system, preferably to keep hostile code from running on a system altogether, but a few potential and applicable solutions have been found.

The problem of video games being vulnerable to cheating is mainly due to two different attack vectors for extracting and modifying information, the first being the application run time memory, the other being through capturing network packets. The later of which with security measures being implemented and made more easily accessible to game developers using popular engines. However without securing the application memory video games will still be left vulnerable to certain kinds of attacks.

Implementing application memory protection will require a major change in how video games are run on a system, and is likely not something that can be handled by a game developer alone. In order for the potential solutions suggested in this paper to be implemented it will need to be approached from a game engine perspective or that of a game distribution platform, and in this way create a virtual machine environment where video games can run separated off from the host operating system in order to achieve a higher level of security.

5 Discussion

The original goal of this paper was to experiment and test different security implementations in order to provide statistics about different solutions and their performance. Then test common cheating software to figure out how the different solutions preform against attacks and how they would be helpful in the video game concept game developed in the pre-project. However during the course of the project researching cheating and how security concerned software handled similar situations there currently are no technology that are directly transferable.

The way that cheating in video games become a different challenge to that of other security sensitive application is the environment surrounding the application as well as where the attacks are originating. Typically a system would be considered compromised if the adversary gains physical or root access to a system. However when considering a cheater in a video game in most cases will be using a private home computer with admin privileges the challenge becomes quite different.

In typical cases where the user of a system is not trusted a hierarchy of trust is implemented and enforced on the operating system level. This will typically be a computer in an office environment where there are trusted system administrators who have access to the root system of the operating system, while regular users are limited into only having access to the system components they require to do their job. As soon as the untrusted user owns the system and is able to run as the administrator the same level of security becomes impossible to uphold without limiting the user in the use of their own privately owned computer.

In most current approaches to prevent cheating it is mainly focused around making the potential reward for cheating less valuable than the effort required to cheat. This is done either through obfuscation of application memory requiring a cheater to reverse engineer the application or preform a large amount of trial and error to find the information necessary. This will cause most casual cheaters to be deterred from attempting to cheat, but if a games audience grows and there becomes a higher and more competitive community around a game the value of cheat software will increase to a point where someone will go through the trouble of doing the research and providing cheating software to those willing to pay for it.

Other approaches to anti cheating being the more proactive anti cheat scanners require high system privileges of the private computer systems they need to scan to the point of being considered a threat to privacy[7], and have proven to be potential way for malware to enter a computer system[6] if a developer proves to be untrustworthy. Both of which are good reason for the video game market to consider other approaches cheat prevention.

The approaches suggested in this paper concerning application memory protection will require a different approach to that of a traditional game development. To implement the approaches suggested here would require the development of virtualization technologies together with a proprietary operating system to act as a layer between the host system that are potentially managed by the cheating player and the game application itself. A solution such as this would likely also not be completely cheat proof but it could help significantly by being able to secure the information channels past the level of the operating system will significantly reduce potential for cheats to be created as they would have to be run within the same virtual machine and proprietary operating system as the game to gain access to confidential information.

With the creation of a virtualized run time environment for video games it have the potential of being created in a way that already excising video games could be ported and run within the environment without requiring changes to be made. However some optimization could potentially be necessary in order to direct what element of the memory that would require this protection and what could be left outside for the sake of the performance of the game.

6 Conclusion

The approach suggested to prevent cheating in video games in this paper could have potential to greatly reducing the most commonly abused vulnerabilities. It does however require a virtualization layer to be developed with its own functions for handling system interactions commonly sent to the operating system to prevent information leakage. This layer will then manage application memory access as well as encryption and decryption of the memory used by the game application during run time. This approach would also be able to increase the effectiveness of the networking encryption used when the application can be able to store keys securely within the virtual environment rather than it being openly accessible in system memory.

I also believe this increased security could be a good marketing point for a game engines or distribution platforms to be able to claim increased application security and protection against cheating that does not pose a threat to their users privacy. Together with the potential for easy portability of excising game titles it could be a worthwhile investment. Adding to that the potential for business solutions using this protection for Virtual reality business meeting or other not gaming related application in need of this kind of security it is an interesting area for development and further research.

7 Further work

For future work on information security in video games the suggested solution from this paper would require a lot of work in order to get to a functional level. Further testing would be required to know whether or not the solution holds any real merit for use in a video game environment. It is however a promising and if not potentially a bit overly complex to undertaken easily. Potential driver support for graphics and other process intensive tasks could prove a challenge within a virtual environment leaving multiple areas of potential failure for such a solution that would be interesting topics for further testing and development.

7.1 Neural network game logic

A completely different approach that was considered shortly when looking into obfuscation during this project was the idea of using neural networks to handle parts of the video game logic. The general though of which being that developing the game normally but then having a game engine able to train a neural network to preforming parts of the game logic provide some unknown complexity to the game logic that would be difficult for a potential cheater to predict and modify in a predictable and beneficial way. This however was not an approach that was deeply researched during this project but could potentially be an area of further investigation.

Bibliography

[1] SACCO, D. 2016. Superdata's 2016 report's top 6. http://www.esports-news.co.uk/2016/12/21/esports-market-size-2016-superdata/. Accessed: 2017-04-20.

[2] Lahti, E. 2014. Cs:go competitive scene in hacking scandal. http://www.pcgamer.com/csgo-competitive-scene-embroiled-in-hacking-scandal-as-three-players-are- Accessed: 2017-04-20.

[3] Lahti, E. 2015. After drug scandal, esl says "esports needs to mature". http://www.pcgamer.com/after-drug-scandal-esl-says-esports-needs-to-mature/. Accessed: 2017-04-20.

[4] Zakrzewski, C. 2016. Wall street journal: Virtual reality takes on the video conference. https://www.wsj.com/articles/virtual-reality-takes-on-the-videoconference-1474250761. Accessed: 2017-05-18.

[5] Simon Al-laeys, A. R. 2016. Steam dev days: Anti-cheat for multiplayer games. https://www.youtube.com/watch?v=hI7V60r7Jco. Accessed: 2017-02-8.

[6] Savage, P. 2013. Esea release malware into public client, forcing users to farm bitcoins. http://www.pcgamer.com/esea-accidentally-release-malware-into-public-client-causing-users-to-far Accessed: 2017-05-20.

[7] Reddit:Douggem. 2015. Battleye is sending files from your hard drive to its master server. https://www.reddit.com/r/arma/comments/2750n0/battleye_is_sending_files_from_your_hard_drive_to/. Accessed: 2017-05-20.

[8] UnityTechnologies. 2017. Unity documentation: Network security. https://docs.unity3d.com/ScriptReference/Network.InitializeSecurity.html. Accessed: 2017-05-20.

[9] Github:Maxenko & SalihBalkan. 2016. Sodiumue4: public and private cryptography plugin for unreal engine 4 based on libsodium. https://github.com/maxenko/SodiumUE4. Accessed: 2017-05-20.

[10] http://lowentry.com/. 2016. Le encryption. https://www.unrealengine.com/marketplace/low-entry-encryption. Accessed: 2017-05-20.

[11] Yukhanov, D. 2017. Code stage: Anti-cheat toolkit plugin. https://forum.unity3d.com/threads/anti-cheat-toolkit-stop-cheaters-easily.196578/. Accessed: 2017-05-17.

[12] Leite, B. X. 2016. Scue4 anti-cheat solution plugin. https://www.unrealengine.com/marketplace/scue4-anti-cheat-solution. Accessed: 2017-05-17.

[13] Mpgh.net:NeSucks. 2015. Color aimbot. http://www.mpgh.net/forum/showthread.php?t=792406. Accessed: 2017-05-20.

[14] Li, Y., McCune, J. M., Newsome, J., Perrig, A., Baker, B., & Drewry, W. 2014. Minibox: A two-way sandbox for x86 native code. In *USENIX Annual Technical Conference*, 409–420.

[15] Dewan, P., Durham, D., Khosravi, H., Long, M., & Nagabhushan, G. 2008. A hypervisor-based system for protecting software runtime memory and persistent storage. In *Proceedings of the 2008 Spring Simulation Multiconference*, SpringSim '08, 828–835, San Diego, CA, USA. Society for Computer Simulation International. URL: http://dl.acm.org/citation.cfm?id=1400549.1400685.

[16] Silakov, D. V. 2012. Using virtualization to protect application address space inside untrusted environment. *Programming and Computer Software*, 38(1), 24–33. URL: http://dx.doi.org/10.1134/S0361768812010069, doi:10.1134/S0361768812010069.

[17] Chen, X., Garfinkel, T., Lewis, E. C., Subrahmanyam, P., Waldspurger, C. A., Boneh, D., Dwoskin, J., & Ports, D. R. 2008. Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, 2–13, New York, NY, USA. ACM. URL: http://doi.acm.org/10.1145/1346281.1346284, doi:10.1145/1346281.1346284.

[18] ValveCorporation. 2013. Steam os. http://store.steampowered.com/steamos/. Accessed: 2017-05-20.

[19] EpicGames. 2017. Unreal engine - encryption components. https://github.com/EpicGames/UnrealEngine/tree/release/Engine/Source/Runtime/PacketHandlers/EncryptionComponents. Accessed: 2017-05-20.

Appendix A: Finding insecurities that can be used for cheating in online games

1

Finding insecurities that can be used for cheating in online games

Student nr: 100889, Gjøvik University College - Supervisor: Simon McCallum

This paper will take a look at a recommended basic approach to networking for real time multiplayer games in the Unity game engine. A concept game is developed in order to test how secure these recommended approaches are against players giving themselves an unfair advantage through cheating/hacking. This basic implementation uses the Unity Networking library and in the test implementation no additional care was taken with concerns to security and cheat prevention.

The second part of this research project was to gain basic knowledge on how cheating works, and to what extent cheating software can be effective against multiplayer games. In the case of this research project Cheat Engine was chosen for the tests. This is a all purpose cheating software running only on the cheating players computer with a focus on altering the memory used by the game as well as the speed the game that is running on the system. This was chosen due to it being the most common approach to cheating.

Categories and Subject Descriptors: 0.0.0 [**Cheating / Hacking in video games**]

General Terms: Cheating / Hacking in video games

Additional Key Words and Phrases: Cheating, Hacking, Video, Computer, games, Multiplayer, Online, Unity, Networking

1. INTRODUCTION

This project began with researching cheating in video games in general, what was common in different genres, and what parts of the client - server architecture is targeted by cheaters. What was found was that most common was cheating entirely based on the cheating clients system. Some exceptions was found where proxies where used to preform a ”Man in the middle” attacks in order to manipulate network packets[McGraw and Hoglund 2007].

Some other articles was also found where the server was directly attacked by a client. As information about these cases are limited and no well known commercial or open source software that are based around this approach was found. Therefore in this project it server targeted cheats was considered rare cases that probably targeted specific weaknesses in a specific game's servers rather than a widespread issue and will not be a focus of this project.

1.1. Multiplayer concept game

The other and main part of this project is to use the information gathered from this research to make a concept game based on the commonly recommended approaches to networking in multiplayer games. Unity was chosen due to its popularity and ease of access, making research here valuable to a high amount of developers. Then by following popular guides and approaches using the included networking tools in unity, the concept was made to be a realistic implementation of a basic multiplayer game. Another priority for the concept game was keeping it simple and modular enough to use in future testing of Cheating countermeasures.

2. CHEATING IN MULTIPLAYER GAMES

From the research done the different kinds of common cheats was separated up into different categories based on what advantage a cheating player can gain from using them. This information was then used to figure out what the concept game should contain in order to test some of the more common kinds of cheating in a high number of genres.

2.1. Value manipulation

One of the most common way of cheating is based around manipulating the different variables used by a game to gain an unfair advantage. Commonly this can be by increasing or locking the players values for their health, ammunition or speed, making them more powerful or immortal. This is typically done by targeting the memory of the client with software such as Cheat Engine. A different less common approach is also targeting the network transmitted data between the server and client to manipulate such values.

Another common use for this is to enable debugging parameters within a game where that you know have them, allowing for a simple way of enabling visual cues that the player is intended to have available during a fair multiplayer game. When connecting to a server the debugging variables will typically be turned off and not accessible through the game client, however accessing these directly in the memory of the clients computer and changing them works for certain multiplayer games.

2.2. Speed manipulation

Another way of cheating is by the tick speed of the clients game and that way making it run at a faster or slower speed than the other players. This can give the cheating player increased speed in the game by increasing the tick speed or alternately more time to react to things happening within the game by lowering their games tick speed.

2.3. Information visualisation

This category of cheating is based around all cheats that aims to give the cheating player access to more information than what was intended by the developer. This is typically done by accessing the game clients memory and then using the information found there to visualize it for the cheating players in beneficial ways, this can be by marking important items or enemy players on a map or showing where other players and items are through walls. In some cases this require additional HUD elements to be created and used by the player and overlaid onto the game or using this information on a map on a second monitor.

2.4. Predicting the future

Randomness in an important element in a wide range of video games and information security in general, In certain games predictable randomness have been used and in some cases where either the source code was accessed a cheater could use this predictable randomness to gain an unfair advantage. This is however not the most common way of cheating, but made this list as it was a big problem in a case for online poker where actual money was made by cheaters abusing this issue in the game code[McGraw and Hoglund 2007].

2.5. Automating gameplay

This last category covers cheating that aims to automate the actions required by the player. This is especially common for games such as MMORPG's where players can make progress by repeating an action many times. This kind of cheating generally does not rely on abusing the game client but rather automating key and mouse clicks with timings and delays allowing tasks to be preformed automatically. Other uses for this can be in cases where a certain combination of buttons must be combined potentially with a set timing for optimal play, such automation can be used to continue the combination whenever the players starts it. Cheating such as this however is much more of a gameplay development challenge than something that can be solved using information security, and as such wont be a priority within this project.

A different part of the automating gameplay category is cheats seeking to automate aiming and potentially shooting, and that way give the cheating player potentially inhuman reaction times and accuracy. The way these are implemented are many, the most common is using information gathered from the client systems memory and that way knowing where enemies are located and that way automatically target them. Other ways is targeting based on color or patterns, where a third party software scans the screen for a set color or pattern and then automatically target it. The last option have become less useful on its own since graphics with lighting and reflections now are more difficult to distinguish, as well as how prone this approach was to errors where it would automatically target dead enemies or patterns in the environment, combined with modifications to the game however where the color of the environment to be easily distinguished it can be both useful and hard to detect.[TCQ-Review 2009]

3. AREAS OF FOCUS

With an overview of the most common areas targeted by cheating it is clear that there are several areas of game development that need to be secure in order to be resistant to cheaters. Too many to all be covered by this paper, and as this paper will act as a basis for further research on information security solutions to counteract cheating some categories are more relevant than others.

The main parts that was interesting from a information security stand point is the area of memory manipulation as well as network based man in the middle base cheating. The other one that was chosen for further testing in this paper is the tick speed changing cheats. Memory manipulation was chosen as a focus for our concept game and for the tests preformed in this paper due to the large amount of cheats where this is the part of the system exploited. A tick speed test was also preformed as well as this is an easily accessible way of cheating that is easily exploited without having to tailor third party software towards a particular game.

4. CONCEPT GAME

As the main reason for doing this project was to create a concept game where both cheating and potential countermeasures could be tested the research into creating as well as the implementation of this game took the major part of this project. This part started up by looking into how a general unity networking multiplayer implementation was done. After looking into a couple alternatives i ended up using guide for creating a ”Simple multiplayer example[Unity3d 2016]”.

This was a simple and clean implementation of basic multiplayer functionality that works great as a basis for the cheating tests that was required for this project. Offering the most basic functionality for testing health, movement and and shooting that need to be synchronised between all the connected clients.

How the different parts of this test concept works is that the health variables for both player and non player characters was managed by the server, and that way be difficult for a client to interact with and change. Movement is implemented in the recommended way where it is managed by each individual client and then reported back to the server and other clients. This is commonly recommended to keep games responsive and avoid issues where the networking connection makes movement feel sluggish for the player. Another reason for this is to allow the client to handle collisions to take that strain of the server. This last part was not part of the concept test but could be an area of future research

The shooting for the concept game works so that the client notifies the server about its gun being fired and the trajectory of the shot and it is then handled by each individual game client to calculate what the bullet hits and does. This part of the gives the possibility for some additional tests but was not used for the tests conducted in this paper.

5. CHEAT ENGINE

Cheat engine was chosen to test our concept game, this tool was chosen because it is a fairly flexible multipurpose open source tool. Cheat engine works by bundling it to a running process and that way gaining access to the processes memory. The main part of the cheat engine software is a powerful search and compare engine to help you locate the interesting variables of a running game.

To find interesting variables you can search for variables based on variable type, initial value or just simply search for a variable with no prior knowledge. After the first search is conducted you can filter out variables by sorting out variables that have increased, decreased or remained the same, or simply changed between searches. This can help you locate variables such as health easily by finding all variables, then loose some health, search up all variables that have decreased, or avoid loosing health and search for the variables that have remained the same. After a number of such searches the correct variable is probably found and this allows cheat engine to either lock the variable so it wont change, and that way making a player invincible. or change it to am unnaturally high number and that way gaining an unfair advantage.

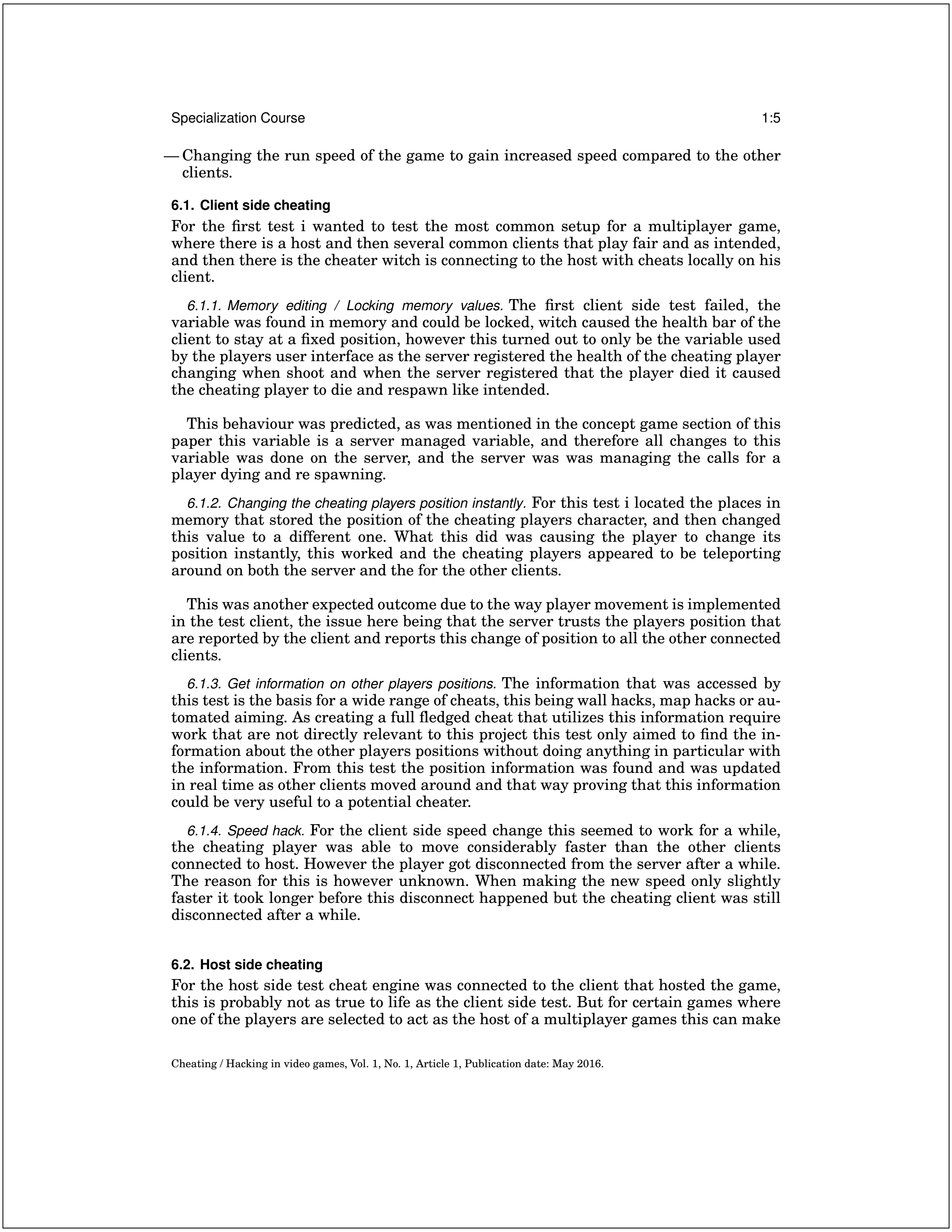
Another possibility with cheat engine is its basic included ”speed hack” where you can simply change the tick speed of the bundled process and making it run either faster or slower than intended. Cheat engine also contains several far more advanced features that can allow for code insertion or disabling or changing certain commands a game is using to make it behave differently. These more advanced features was not necessary for our test however so these wont be relevant for this project.

6. CHEAT TEST

For the cheat testing in this paper a focus was put on the memory, and tick speed based cheats. For this cheat engine was used as this is a good multipurpose and open source tool commonly used by cheaters. I tested the concept game twice, once where cheat engine was only connected to the client, and then tested again with cheat engine connected to the host. This was done to see what differences this made and what kind of cheating was accessible where.

A selection of 3 different kinds of cheating was chosen for our tests

— Find the health variable of the cheating player and lock it so that it wont change and that way making the cheater invincible.
— Finding the other players connected to the servers positions and that way always know where they are in the game world, even when not visible on the screen.
— Changing the position of the cheating player, and that way allow the player to instantly change the position of the player.



— Changing the run speed of the game to gain increased speed compared to the other clients.

6.1. Client side cheating

For the first test i wanted to test the most common setup for a multiplayer game, where there is a host and then several common clients that play fair and as intended, and then there is the cheater witch is connecting to the host with cheats locally on his client.

6.1.1. Memory editing / Locking memory values. The first client side test failed, the variable was found in memory and could be locked, witch caused the health bar of the client to stay at a fixed position, however this turned out to only be the variable used by the players user interface as the server registered the health of the cheating player changing when shoot and when the server registered that the player died it caused the cheating player to die and respawn like intended.

This behaviour was predicted, as was mentioned in the concept game section of this paper this variable is a server managed variable, and therefore all changes to this variable was done on the server, and the server was was managing the calls for a player dying and re spawning.

6.1.2. Changing the cheating players position instantly. For this test i located the places in memory that stored the position of the cheating players character, and then changed this value to a different one. What this did was causing the player to change its position instantly, this worked and the cheating players appeared to be teleporting around on both the server and the for the other clients.

This was another expected outcome due to the way player movement is implemented in the test client, the issue here being that the server trusts the players position that are reported by the client and reports this change of position to all the other connected clients.

6.1.3. Get information on other players positions. The information that was accessed by this test is the basis for a wide range of cheats, this being wall hacks, map hacks or automated aiming. As creating a full fledged cheat that utilizes this information require work that are not directly relevant to this project this test only aimed to find the information about the other players positions without doing anything in particular with the information. From this test the position information was found and was updated in real time as other clients moved around and that way proving that this information could be very useful to a potential cheater.

6.1.4. Speed hack. For the client side speed change this seemed to work for a while, the cheating player was able to move considerably faster than the other clients connected to host. However the player got disconnected from the server after a while. The reason for this is however unknown. When making the new speed only slightly faster it took longer before this disconnect happened but the cheating client was still disconnected after a while.

6.2. Host side cheating

For the host side test cheat engine was connected to the client that hosted the game, this is probably not as true to life as the client side test. But for certain games where one of the players are selected to act as the host of a multiplayer games this can make a difference in what cheats are available.

The results for the testing done on the host player was for most cases the same as when conducted on the client. The only exceptions being that when running on the host it was possible to lock the health variable, making this player truly invincible. The other difference from the client side test was that in the speed manipulation test it worked for a longer period, but after a while all the other clients connected to the host was disconnected while the host could keep playing with increased speed.

7. AVAILABLE CHEATING COUNTERMEASURES

For the last part of this project research was done on what cheating countermeasures are currently available to Unity game developers, i found some tools and recommendations that potentially have the ability to help. For this project i have not conducted any tests with these suggested solutions, or looked deeply into how they work but they aim to cover some of the holes that a cheating player would use so these solutions would definitely make cheating more difficult.

7.1. Authoritative Networking

One suggested solution to counteract cheating is basing the game design on Authoritative Networking this described in the now legacy Unity manual[Unity3d a]. The way this works is by only having the server control the game, while limiting the players interaction with the game world to watching and sending the host information about their button clicks. This is a solution that does have the potential to be effective as proven by the our test that showed us that the cheating client was not able to modify and abuse this variable.

There are however some downsides to consider with this approach in that it would require a very powerful server, and the network response times could become an issue for certain genres of games and cause them to feel unresponsive. However for slower paced games without the need for quick response times this can be a viable solution for increased resistance towards cheating.

7.2. Secure connections

Unity provides an option network layer security for developers who wish to use them. This is done through the use of AES encryption, and aims to prevent unauthorized read and prevents ”man in the middle” replay attacks. It Adds CRCs(Redundancy checks) so that data tampering can be detected. Adds randomized, encrypted SYNCookies to prevent unauthorized logins and uses RSA encryption to protect the AES key. Most games will want to use secure connections[Unity3d b]. However they do add up to 15 bytes per packet and take time to compute will be effective at countering networking attacks towards a game.

7.3. Anti-Cheat Toolkit

The last and maybe most relevant to this project is the Anti-Cheat Toolkit, available for purchase in the Unity Asset store[focus 2016]. In the words of the developer this toolkit aims to:

> Anti-Cheat Toolkit (ACTk) designed to let you add some extra pain to the guys who cheat / hack / crack something in your game.

This toolkit claims to add security to several different areas of a game, memory protection being one of them by adding ”Obscured” variables to replace standard and some unity variables. For our research however we were unable to find any good explanations on how these variables where obscured, how much of a performance impact this would have on the application or how much security this would add to the application. This might be grounds for further research however to see how good of a solution this toolkit offers in regards to memory protection.

8. DISCUSSION

Cheating in video games clearly target many different areas of the client-server structure, there if security is important to a multiplayer game it needs to be considered throughout development to be effective. There are some offerings that aims to prevent cheating, or at the very least make it more difficult.

From the research done for this paper it seems most of the counter cheating options aims to obscure the information needed by the cheating software. Outside of the communication layer of security there does not seem to be many solutions using proven techniques based on encryption and cryptography, at least not where the approach is openly shared. What exact approach solutions such as the Anti Cheat Toolkit includes have not been researched thoroughly in this paper however so the efficiency of this solutions is unknown.

9. CONCLUSION

Based on the tests conducted in this paper it is clear that in order to prevent cheating special care is required during development. The most common and suggested solutions does not take this security into consideration, and therefore it would be up to each developer to put the required effort into securing their game if they wish to prevent cheating.

As all of the cheat prevention approaches found researching this paper does require additional system resources and development time it should be considered on a case by case basis, how important countering cheating is for a particular game. Offering proven and good security options however could be an important step in order to make developers consider security from an early stage of game development.

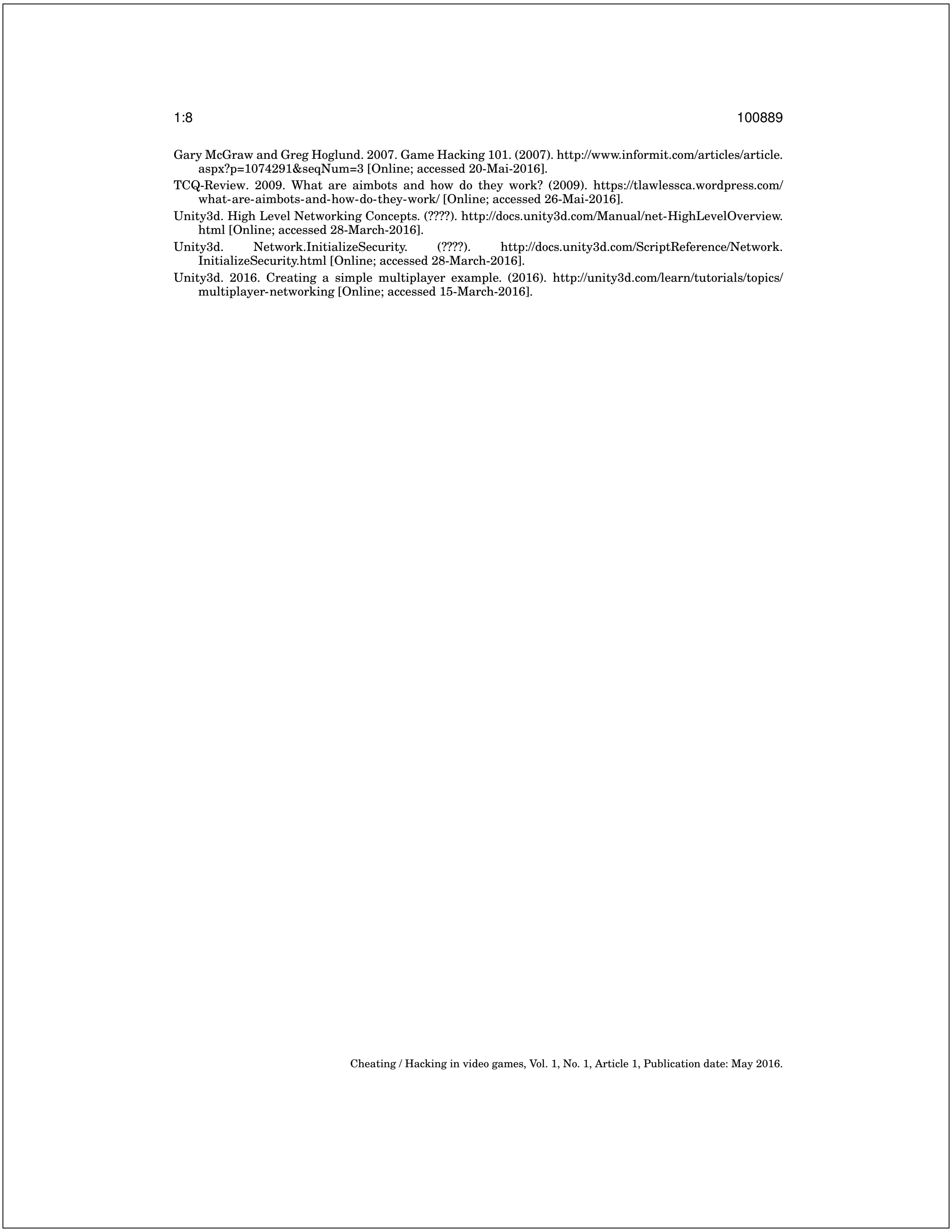
10. FURTHER WORK

The focus of this project was to gain a deeper understating of how cheating in video games is done, and how cheating software is able to alter how the rules of a game for one player so that it differs from those of the other players connected to the same server. This project was mostly focused on how client side memory editing is done and works. For further research there are other areas of video game cheating that could be an interesting area of focus.

The other major part of this project was the development of a basic concept game where cheating software could be tested, this will be useful for my further work on my master's thesis where the focus will be on how to prevent such cheats. This concept game will be the base for implementing and testing potential solutions. This concept is developed to be clean and modular and easily edited, changed and tested during further work in this area.

REFERENCES

focus. 2016. Anti-Cheat Toolkit (ACTk). (2016). https://www.assetstore.unity3d.com/en/#!/content/10395 [Online; accessed 15-March-2016].



Gary McGraw and Greg Hoglund. 2007. Game Hacking 101. (2007). http://www.informit.com/articles/article.aspx?p=1074291&seqNum=3 [Online; accessed 20-Mai-2016].

TCQ-Review. 2009. What are aimbots and how do they work? (2009). https://tlawlessca.wordpress.com/what-are-aimbots-and-how-do-they-work/ [Online; accessed 26-Mai-2016].

Unity3d. High Level Networking Concepts. (????). http://docs.unity3d.com/Manual/net-HighLevelOverview.html [Online; accessed 28-March-2016].

Unity3d. Network.InitializeSecurity. (????). http://docs.unity3d.com/ScriptReference/Network.InitializeSecurity.html [Online; accessed 28-March-2016].

Unity3d. 2016. Creating a simple multiplayer example. (2016). http://unity3d.com/learn/tutorials/topics/multiplayer-networking [Online; accessed 15-March-2016].