Prevention vs Detection in Online Game Cheating

University of Oulu
Information Processing Science
Bachelor's Thesis
Waranyoo Ronkainen
04.11.2021

Abstract

Cheating is a major problem in online games, but solving this would require either a complicated architecture design, costly third-party anti-cheat, or both. This paper aims to explore the differences between preventive and detective solutions against online game cheating. Specifically, it explores solutions against software-based cheatings, what kind of cheats there are, and what proposed and implemented solutions there are. This paper was conducted using literature reviews as methodology, using relevant papers from databases such as ResearchGate, ACM, and IEEE.

In this paper, it was concluded that a good prevention strategy during the game development phase is adequate to mitigate and prevent cheating but will require appropriate anti-cheat software to maintain fairness during the lifetime of the game. The importance of an online game's network architecture choice in preventing cheating became apparent within this paper after comparing the benefits of each type side-by-side. Results showed that peer-to-peer architecture not having a trusted centralized authority means that the game needs to rely more on an anti-cheat software to prevent and detect cheating. This paper could not conclude what an appropriate anti-cheat software is because the topic is outside of the scope of this paper and lacks public data. Still, it does raise the question of whether a more aggressive anti-cheat strategy is suitable for a game or not.

Contents

1. Introduction

What is the difference between cheating prevention and detection in any online game, which solves the cheating problem more effectively, and which one provides a more cost-effective solution? This paper will aim to answer the questions mentioned above by conducting a literature review on software-based cheats and solutions.

Online games have become one of the largest entertainment types in the world, and it has changed how people enjoy entertainment, and for some, it has changed how they live their life. One of those life-changing effects is esports. Esports, short for electronic sports, are a type of sport played on a video game platform, and like with any kind of sport, there is a form of competition. Like all sports that have passionate fans worldwide, these competitions can become quite fierce. Most of the time, esports takes the form of an organized, multiplayer, tournament-style video game competition. The prize money for the winner or winners can be as high as multiple million USD. The successful players in these competitions can earn their living by playing games and are thus called professional gamers. In some countries, these professional gamers are treated similarly to well-known celebrities from other entertainment industries or sports. Non-professional players hope to be recognized by a professional gaming team and live out their dream life. This recognition is one of the reasons why the competition among non-professional players can also be quite intense due to the aspiration of becoming a professional gamer. The aspirations and seriousness in these competitions create the need to ensure that the game is played fairly since some players might be motivated to cheat to win. With so many players competing, it is quite difficult to prevent and detect every player using cheats.

Cheating has always been an issue in online games, but it has become a major problem in recent years. It has become so much of a problem that many games have opted to use third-party anti-cheat software or solutions. (Cano, 2016; EAC, 2018; Warren, 2020) These anti-cheat software are intruding on players' privacy by scanning and analyzing the processes currently running on the player's computer. However, some have gone beyond this by having functions to send files back to the anti-cheat's server and remotely run shellcodes. (Battleye, 2016; The & Khanh, 2010.) In online gaming, the factors of large participation groups and high monetary reward entices cheating. Developers of online game cheats are benefiting from this economic opportunity. The practice of selling online game cheats may be an unethical way of earning money; it has nonetheless become a possible way to earn a living. (Maiberg, 2014.) Cheating has troubled many games and game companies because it ruins the experience of playing games and destroys the fairness of competition. Cheating makes some players leave the game, and as a result, the gaming company suffers financial loss, which eventually leads to the game's decline. (Chalk, 2021; Warren, 2020.) Because of this, it is essential to prevent cheating and detect cheats and/or cheaters.

Preventing cheating before it occurs would be the optimal solution, but realistically it is impossible to prevent every cheat and their usage before they happen. Thus, cheat detection is needed to prevent further damage after the cheating has already occurred. The development of anti-cheat software has come far, but so have cheating techniques, especially in modern online games (Cano, 2016). This results from modern online games being a software distributed to players who then interact with each other over the internet. With the increasing complexity within a game's software and the interaction between its players, online games become more vulnerable to many different types of cheats (Mönch et al., 2006). Most types of cheats involve reading or tampering with the game's code,

memory, and configuration data separately or any combination of the three on the cheater's side of the game client (Yan & Randell, 2005).

2. Modern online games

There are many types of modern online games, but the most popular types can be categorized as Massively Multiplayer Online Game (MMOG), Multiplayer Online Battle Arena (MOBA), First-person Shooter (FPS), and Real-time Strategy (RTS). All the types mentioned above of online games share the same basic qualities that make them successful: low communication delay, scalable architecture, and cheat and/or cheater prevention and detection. (Baughman et al., 2007; Webb & Soh, 2007.) While they all share the same qualities to be successful, the requirements to meet those qualities are different depending on the type of the game. For example, MMOG is a type of game where hundreds or even thousands of players share the same persistent world instances, hence the need for robust networking and scalable architecture (Farlow & Trahan, 2014; Webb & Soh, 2007). Whereas MOBA is a type of fast-paced game that has 2 – 10 players share the same world instances, which resets every time the match ends, but there is a massive amount of these instances, hence the need for low communication delay (Cassar et al., 2014). However, to meet those requirements depends a lot on the game's architecture, therefore making the architecture of the game the single most important aspect of a successful game. Client/server (C/S) architecture and Peer-to-peer (P2P) architecture are two common architectures used in online games. (Webb & Soh, 2007.)

2.1 Client/server

C/S architecture has been popular among online games, especially in MMOGs. The advantages of this architecture lie in the control over the game and in security. C/S architecture has the ability for important things to be run and calculated from a trusted, centralized server. These servers send enough information for the client to render and update all performed actions, world instances, and the player's character. A drawback in C/S architecture is that it has poor scalability due to the performance and network overhead increases in proportion to the number of concurrent players in the game. This flaw becomes apparent in modern online games where the players might be geographically far from each other. Therefore, a high network bandwidth and fast networking speed from the server is required. In C/S architecture, the playing experience is largely dependent on the server's condition. If the server has high latency, is overloaded, or suffers from a hardware or system failure, the players will not be having a pleasant playing experience. Mirrored server environments aim to mitigate this problem, but it does not change the core fact that scalability is poor. (Baughman et al., 2007; Farlow & Trahan, 2014.)

Mirrored server architecture contains many identical copies of the game servers, and players connect to one of them, ideally the one best suited for them. These mirrored servers connect to each other via internal network to sync their data across the servers. The latency problem is largely mitigated with the mirrored servers' internal network being a lot faster than the player's network. This will mitigate the issue of players being geographically far from each other while playing in the same game world instance. If there are enough servers scattered around the world or enough servers concentrated at the location with the heaviest network traffics, this will also mitigate server overload issues. However, having an abundant number of servers does not fix the whole problem. Instead, it creates an additional load known as server-to-server communication. Not only do mirrored servers need to send updates about players and the game world to the players, but it also needs to send the same updates to other servers to maintain consistency. Doing so creates an additional load on the network, thus increasing the cost of

operating/maintaining the game. (Baughman et al., 2007; Farlow & Trahan, 2014; Webb & Soh, 2007.) Compared to MMOG, there are no real needs to update changes about the world with MOBA, FPS, and RTS games because the game world instances are not persistent or shared with only a small number of players, making scalability issue smaller for those types of games.

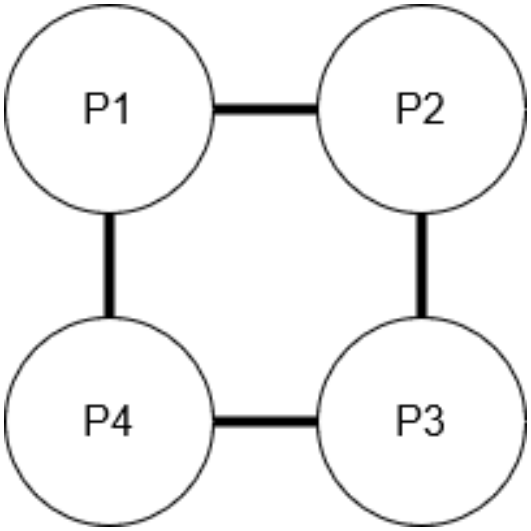

Figure 1A – Peers having maximum of 2 connections to other peers forming chain of P2P network.

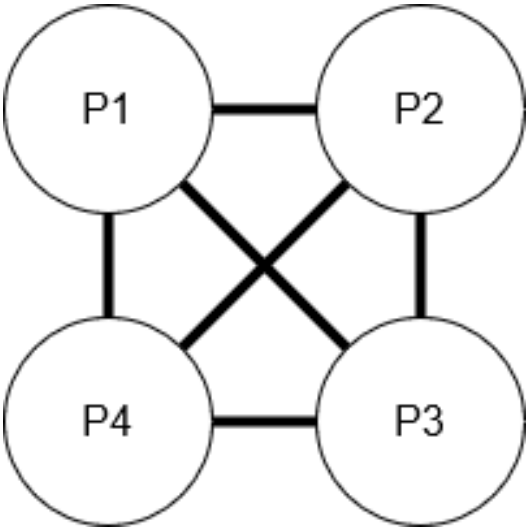

Figure 1B – Peers connect to all peers forming mesh of P2P network.

2.2 Peer-to-peer

P2P architecture is an architecture without a centralized server where players, i.e., peers, are network nodes that directly send updates to each other. This is also sometimes called serverless architecture. P2P architecture increases the scalability and performances significantly because each connected peer increases the overall network bandwidth and computational power. Communication delays depend on how the P2P architecture is implemented; figures 1A and 1B illustrate this. In Figure 1A, each player is connected to a maximum of two other players, forming a ring between all players. For this scenario, the communication delay is determined by the player with the slowest connection speed. In Figure 1B, every player is connected, thus, making the communication delay the average of the connection speed of every player. (Cassar et al., 2014.) While the previously mentioned P2P techniques solve the scalability issue, it introduces new problems. One problem faced is the network strain on the player's side when the network's performance is tied to either the weakest link or the computed average among all peers. A new P2P technique is now being used to mitigate these new problems introduced by the traditional P2P architecture. This technique is called supernode or superpeer technique. (Schiele et al., 2007.)

Superpeer technique is where a game assigns a virtual "area of interest." This area could be a game area separated by a loading screen or the area around the specific player(s). Inside the area of interest, peers would act according to the traditional P2P architecture by sending updates to each other. Additionally, the game would then assign one player inside the area to be a superpeer. This superpeer would be the only peer who sends updates to the outside of the area of interest, thus creating a connection to all the other instances of areas of interest. So, instead of all the peers connecting to each other, the superpeers connect to one another and distribute the updates to the peers inside the area of interest. This technique gives an advantage over the traditional P2P technique in performance because the game can choose the player with the best connection speed to be a superpeer.

(Cassar et al., 2014; Schiele et al., 2007.) However, this technique introduces a new connectivity issue when the superpeer is suddenly lost or disconnected. This issue can be mitigated with additional design to the architecture to account for the sudden drop of the superpeer. (Schiele et al., 2007.)

All the benefits mentioned above do come with drawbacks because of how the sensitive and critical information is stored in a player's device, and all calculations are done locally. It is quite easy to modify these values, and these modified values can then be sent to other players as valid and truthful values. Since there is no centralized authority, the detection is difficult because there is no way to determine which player data to trust (Prather et al., 2017). Consistent algorithms can be implemented to prevent errors in the game and player states, but this will at best mitigate the problem (Webb & Soh, 2007). This is especially true in the case of when superpeer technique is being used because the superpeer will essentially hold authority over other peers in the area of interest.

2.3 Hybrid

Hybrid architecture is a combination of both C/S and P2P architecture. This architecture type aims to bring the security of C/S and the scalability and flexibility of P2P architecture together. In practice, this means that some workload will be performed on peers, but most, if not all, of the critical calculation and data will be performed on the server. (Baughman et al., 2007.)

There are many ways to implement the hybrid architecture. One of those ways is to have the server act as a peer with more authority over data's validity and consistency, and all of the peers will connect to each other and form a mesh. The server is there to verify the received data and distribute it further. This implementation will lessen the server's strain compared to C/S architecture and will have improved control over the game compared to P2P architecture. This implementation can be further improved using the beforementioned superpeer technique used in P2P architecture. In this scenario, the server will always act as the superpeer instead of having one of the peers take on the role. The server's strain will be further lessened, and the same security level will be kept. (Bethea et al., 2011; Buyukkaya et al., 2009; Cassar et al., 2014.)

However, accomplishing the things mentioned above is a difficult thing to do. The difficulty arises from how complex the data distribution among the peers is and the complexity of how much is given to each peer without compromising the integrity, consistency, or security. (Cassar et al., 2014.)

3. Cheats

Cheating in modern online games can be largely categorized into two big categories: software-based cheating and non-software-based cheating. In practice, software-based cheats mean that cheaters have developed or obtained cheat software to be used on an online game to gain an unfair advantage over normal players. This cheat software could assist cheaters by showing any hidden information. The hidden information could be as small as seeing another player hiding behind the nearby wall or as large as showing the exact location of all the other players in the game world. For example, the cheater may react to specific things more quickly, like dodging attacks or initiating a counterattack by seeing the hidden information. (Cano, 2016; Yan & Randell, 2005; Web & Soh, 2007.) Non-software-based cheating means gaining an advantage without relying on a software or exploiting a game's bugs, such as colluding and disconnecting before match round ends, thus avoiding loss. (Yan & Randell, 2005.)

3.1 Software-based cheats

There are many types of software-based cheats, and there are many ways to categorize them. Some ways to categorize these cheats are by their nature, such as automation, visualization, modification, and exploitation; by their capabilities such as reading and modifying a game's data or memory; and by how they gain access to a game's data memory/functions, external memory reading, and internal code injection/hooking. (Cano, 2016; Yan & Randell, 2005.) While how cheat software gains access to the game's data/memory is out of the scope of this paper, we will explore cheat software's nature and capabilities because it will be important in the context of prevention and detection of cheat software.

Automation cheats are those that automatically do something for the cheater. This type of cheat could be something as simple as macros that repeatedly left-click a specific position on the screen to activate an activity in the game while the cheater is not present in front of the gaming device, gaining in-game items while not playing the game. It could also be as complicated as mimicking the human's mouse movement, such as non-linear and non-smooth acceleration/speed of mouse movement to assist in aiming at the enemy in FPS type of online games. In the context of cheating in online games, automation has no obvious limitation of what can be achieved. For example, a complex automation software suite (bot) can be developed to play the entire game for you from the beginning until the end. (Cano, 2016; Yan & Randell, 2005.)

Visualization cheats reveal hidden information to the cheater. This hidden information could be stealth enemies that could not be seen normally; details of the enemies such as health, weapons, and/or ammunitions; and/or game world's hidden information such as next event's details and enemies' spawn timings. Visualization cheats have a strict limitation of what it can achieve because what information the game client does not have cannot be revealed. (Cano, 2016; Moffatt et al., 2011; The & Khanh, 2010; Yan & Randell, 2005.)

Modification cheats modify how the game works. In some cases, this means modifying a cheater's game character to have better attributes or prevent a specific attribute from falling below a specific level. These modifications could be freezing or giving infinite health points, giving more movement speed than normally possible, preventing ammunition from decreasing, etc. (Cano, 2016; The & Khanh, 2010; Yan & Randell, 2005.)

Exploitation cheats are a cheating activity that involves exploiting the flaws or bugs within a game design to achieve an unfair advantage. This exploit can be software-assisted, but it is not necessary. For example, there could be an exploit to duplicate a game's item. This exploit can be made more effective for a cheater by making a macro that automatically duplicates the item as fast as possible. (Yan & Randell, 2005.)

Before any cheat software can do automation, modification, or visualization, it will need to somehow access the game's information. This is, in most cases, done with reading the device's memory section allocated for the game. By reading the game's memory, a cheat software could display any hidden information within the game's code yet not visually displayed. This results in visualization cheats. The cheating software could also read the location of a game's objects which can be used to automate things by simulating the mouse movements and keyboard keypresses. This results in automation cheats. (Cano, 2016; The & Khanh, 2010; Yan & Randell, 2005.)

Reading a game's memory might not be enough to achieve some of the things that cheaters want to achieve, thus making memory modification the next step in software-based cheats. Adding memory modification capability to a cheat software will not only make modification cheats possible, but it will also make a greater level of automation possible. This greater automation level comes from the ability to directly invoke the game's code to interact with the game's objects, such as attacking or casting spells. (Cano, 2016; The & Khanh, 2010.)

3.2 Non-software cheats

Non-software cheats are the more "classical" way of cheating. Some examples of non-software cheats include cheating by colluding, match-fixing, "boosting," and intentionally disconnecting during the game (Yan & Randell, 2005). While not as game-breaking as software-based cheats, non-software cheats still negatively impact normal players due to the unfairly gained advantage or status. (Blackburn et al., 2014.)

Collusions as a non-software cheat in any online game have been widely seen in recent years. Collusions happen most often in online games that have a limited number of players interacting with each other simultaneously, like in MOBA, FPS, or RTS. A common collusion is when two or more players conspire to have a secret alliance until they are the only ones left in Free-For-All (FFA) game mode. (Yan & Randell, 2005.)

Match-fixing is when two or more players/teams have decided beforehand which one of them will win the match. The teams will proceed to play the match in a way that will favor the player/team they chose to win without letting any spectators notice. Match-fixing usually has monetary motives involved. For example, Team A decides to pay Team B to make them intentionally lose the match, Team A, an underdog, and Team B, the tournament's favorite. Team A then proceeds to bet using third-party personnel that Team A will win in the gambling platform. With large odds against Team A winning, the money obtained by this match-fixing can be a lot more than what Team A would obtain through legitimate means in the tournament. (Yan & Randell, 2005.)

Boosting is where a player hires or lets a more skilled player play on their account to achieve a better ranking while pretending to achieve everything by themselves. (Blackburn et al., 2014.) If the game has a ranking system, the boosted player often ruins other players' experience because they have attained a rank that does not suit their skill level, thus lowering the match quality for other players. Boosting is considered cheating

due to breaking the game's term of services, most if not all of the online games prohibit account sharing in any form. (Blackburn et al., 2014; Yan & Randell, 2005.)

4. Prevention

Cheat prevention is a challenging task that has to be taken into consideration at all levels, from design to maintenance, to achieve the best possible result. The reason is that no matter how good cheating prevention measures are taken in the maintenance phase, it will always be inferior to having a good design that takes cheat prevention into consideration. Preventing cheats at a design level costs much less than preventing cheats after the game's release. (Baughman et al., 2007; Bethea et al., 2011; Chambers et al., 2005; Mönch et al., 2006.)

Even when cheat prevention is taken into consideration in every phase of the game's development, there might be things that could not be anticipated. There might be things that by taking cheat prevention measures would require unreasonable overhead from the game, such as the network bandwidth. The cost in either the development or maintenance phase might also be too high for such preventive measures. This cost creates the need to have an anti-cheat system that could be continuously developed even after the game is released and continues to work independently from the game. (Bauman & Lin, 2016; The & Khanh, 2010.) Many online games develop their own anti-cheat system, but most online game developers buy a license to use or integrate third-party anti-cheat system into their games. Currently, the most popular online games use third-party anti-cheat systems such as BattlEye (BE) and EasyAntiCheat (EAC) (SteamDB, 2013).

4.1 Solutions

There are many solutions to prevent cheats, but they can be categorized into server-side and client-side solutions. Server-side solutions simply mean that those solutions are deployed in the server, such as server-side verification or minimizing information exposure to players. Client-side solutions are deployed with the game's client, those solutions usually involve anti-reverse engineering, anti-hooking, and/or anti-code injection. (Bauman & Lin, 2016; Bethea et al., 2011; Chambers et al., 2005; The & Khanh, 2010.)

4.1.1 Server-side

The server-side solution will not work in P2P architecture due to the nature of serverless architecture. Also, server-side solutions introduce an additional strain to the server, making scalability even worse than before. (Baughman et al., 2007; Bethea et al., 2011; Chambers et al., 2005; Moffatt et al., 2011.)

Server-side verification means that some or all of the game client's actions need to be verified by the server before the actions take place or become permanent. In practice, this often means that an update to critical data such as health points will be checked by the server whether the update is valid or not. This could also mean that the critical calculations such as movement speed are being checked/calculated from the server. For example, the server receives an update on player A's location, and in the next update, player A's location was drastically changed. The server will check whether this change could be made without cheating and then will take appropriate action. (Bethea et al., 2011) In theory, this could prevent anything that modifies the game client to achieve unfair actions/interactions. However, due to computational power, network speed, and bandwidth limitations, it is realistically impossible to verify everything.

While server-side verification can prevent many cheats, especially modification cheats, it will still not prevent visual cheats. For example, the "wallhack" allows a cheater to see another player through the wall. To prevent or minimize the effect of such cheats is to minimize the information exposure to game clients. The server should not send any information that the client does not need to them. Even though the idea itself is simple, the implementation is not. To know what information the client needs will depend on the genre of the game and how much overhead the server has. For example, an FPS game might have a valid reason to send information about the enemy behind the wall if the current gun can penetrate the wall. However, it could also be hidden, and if the player decides to shoot the wall, the server will calculate whether that shot will hit the enemy behind the wall or not. (Chambers et al., 2005; Moffatt et al., 2011.)

The server-side solution will increase the development cost because of added works and testing that needs to be done. This will also greatly increase the cost of operating/maintaining the game due to the need for increased computational power and network speed/bandwidth in the server.

4.1.2 Client-side

Every software-based cheat starts from reverse-engineering the game's code except certain cheat types that scan for specific pixels to gain information from the game. The reason is that the cheat software needs to be somehow aware of the game's world. The game's code needs to be protected to prevent or make it harder for cheat developers to create cheats for the game; this is known as anti-reverse engineering. For this reason, most of the games or third-party anti-cheat solutions use anti-reverse engineer techniques. These techniques may include control flow hiding, code encryption, code virtualization, and/or packing. Having an anti-reverse engineer system is generally a good idea, even for offline games, because it will help with the unauthorized distribution of the game. (Bauman & Lin, 2016.) However, just protecting the game code is not sufficient since not only is it not effective enough, but it is also hard to update/improve afterward.

To further protect the game from software-based cheats, the anti-cheat system needs to protect the memory segment that stores the game client's data. Protecting the game's memory segment is challenging because the anti-cheat system needs to essentially protect the memory segment in an untrusted environment while still giving access to the game client. This task is more difficult if the game is cross-platform due to different levels of exposure and access to a low-level application programming interface (API). For this reason, the main focus will be on Windows-based solutions because Windows is the most used operating system (OS) for online gaming purposes, that is, if you do not count mobile and console games. The currently proposed and implemented solutions to protect the game's memory segment are to prevent blacklisted (or only allow whitelisted) processes from accessing the memory segment; elevate the game process's level so that it would require at least kernel-level of access to read/modify the game's memory or hide the game's process from Windows process list. In Windows environments, this can be done by hooking into the lowest-level (ring0) API. (The & Khanh, 2010.) Over the years, protecting the software's memory has had improvement, with the latest one having the most promise for wide adoption was Intel's Software Guard Extensions (SGX). SGX provides a user-level code in an isolated region of memory protected from processes running at higher permission levels or with lower ring access. However, this feature falls short because it is only available on Intel processors, specifically on Intel Skylake or newer. (Bauman & Lin, 2016.)

Developing a complex and extensive anti-cheat system is quite costly; therefore, most game developers or publishers tend to buy a license or contract an anti-cheat company to deploy their anti-cheat solution. For this reason, anti-cheat solutions are made in a way that requires little action from the game developers. However, this does not stop cheat developers from finding a way to bypass, exploit vulnerabilities and develop a cheat application, therefore creating an arms race between the cheat developer and the anti-cheat developer. (Robles et al., 2008; The & Khanh, 2010.)

5. Detection

Sometimes, it is hard to differentiate between cheat prevention and detection. In this paper, detection will be defined as detecting a successful cheating action and preventing further cheating with the same cheat software.

Cheat detection is often underestimated in an anti-cheat discussion even when studies show that with limited resources, the best strategy is to invest more into detection than prevention. This is mostly because game experiences and/or reputations have been ruined before cheating is successfully detected. Detection is more of damage control than damage prevention due to the nature of current cheat detection techniques. Nonetheless, cheat detection is still important because it is realistically impossible to prevent all cheatings perfectly. (Barreto et al., 2017)

5.1 Solutions

Cheat detection can be divided into detecting cheat software and statistical/behavioral analysis to detect cheaters. Cheat detection is not as straightforward as cheat prevention due to the nature of the ever-evolving cat-and-mouse game that cheat developers and anti-cheat developers have. Efforts have been made to stop this cat-and-mouse game by instead of analyzing the statistic and behavior of the players to detect cheaters, but those have been proven to be difficult to implement and prone to false positives (Alkhalifa, 2016; Chapel et al., 2010; The & Khanh, 2010.)

5.1.1 Detecting cheat software

Once again, this section will focus only on Windows-based solutions for the reasons mentioned in the "Prevention - Client-side" section. The current proposed and implemented solution is to have a hook in the ring0 API that gets called every time a new process is created or tries to get the handle for the game's process. The process or the file will then be analyzed. If the process is found to be suspicious, the details and possibly the file that created the process will be sent to an anti-cheat expert for further analysis. Once the expert has determined the process or file to be a cheat software, the code signature or file hash will be added to the blacklist to prevent any further cheating with the same cheat software. (The & Khanh, 2010.) Anti-cheat developers could also acquire the cheat software like cheaters do. For example, an anti-cheat developer could download cheat software from public sources or buy it from private cheat developers. After that, they can analyze it and then blacklist it.

This solution is expensive and, most of the time, ineffective. This is because of the need to manually analyze the process or file and sometimes manually acquire the cheat software to be analyzed. Also, it is easy to bypass this type of detection due to how a cheat software gets blacklisted by code signature or file hash. Thus, it depends on how strict the scans are; it could be easily bypassed by modifying the code of cheat software. On the other hand, if the scans are too loose, it will produce false-positive results.

There have been proposed solutions that are hardware-based, but those have the same limitation as Intel SGX's solution in cheat prevention, thus resulting in poor adoption (Feng et al., 2008). However, hardware-based solutions might be a good solution for esports tournaments where a small number of players are competing, and the hardware used to play the game is owned by a single company. This is, however, out of this paper's scope.

5.1.2 Detecting cheaters by statistic

Currently proposed and implemented statistic-based detection solution analyzes either the player's overall statistics or the game's specific events. Win rate, headshot percentage, and online time are usually included when analyzing the player's overall statistics. Mouse movement before/after kills and reaction time are usually looked at when it comes to the specifics of the game. (Chapel et al., 2010; Liu et al., 2017; Yu et al., 2012.)

Statistic-based detection depends on the modeling of the game and cheaters to differentiate between cheaters and legitimate players. This means that before the detection can be implemented, a set of data has to be collected, and models have to be made. It is a costly operation, and the result will be tied to that specific game the models are made for, thus making statistic-based detections not well adopted in the gaming industry. Artificial intelligence (A.I.) based solutions have been proposed and used in recent years, such as machine learning. These A.I.-based solutions have the potential to be more accurate than traditional human-made models and produce fewer false positives. These solutions will not fix the fundamental problem with statistic-based detection. If the specifics of statistic models are exposed or leaked, the cheaters can effectively avoid detection by software or gameplay style means. (Alkhalifa, 2016; Chapel et al., 2010; Liu et al., 2017.)

Statistic-based detection is prone to false positives because all of it is probabilities. It is up to the implementer to decide at what percentage it becomes beyond reasonable doubt that the player is cheating. Therefore, most of the proposed and used solution includes manual reviews by humans at some point before the decision is made whether the player is cheating or not. This will introduce human error to the system, but it will mitigate if not eliminate the systematic error by statistic-based detection completely. (Barreto et al., 2017; Chapel et al., 2010; Liu et al., 2017; Yu et al., 2012.) Having a manual review at any point of the time in a cheat detection system will increase the cost of operating the system and greatly decrease the scalability of the anti-cheat system because of labor costs. Some solutions use statistic-based detection to flag potential cheaters and have other players do a majority vote whether the player cheated or not. (Lahti, 2018; Valve, 2016.)

6. Discussion

Software-based cheats cause the most damage in online games when it comes to cheating because it is often a game-breaking level of abuse, and it can be done on a large scale. But no matter how carefully a game is designed, how much cheat prevention and detection are taken into consideration; there will always be cheat developers motivated enough to find the way to cheat (Barreto et al., 2017). Game design that doesn't unnecessarily make the game more prone to cheating is the cheapest and most effective way to prevent cheatings. If a lot of effort is put into a game design that is cheat resistant, it will lower the required investment to other kinds of cheat prevention and detection afterward. For example, having unencrypted network communication and unminimized information exposure to the client will enable visualization cheat and enable cheat based on a Man-In-The-Middle (MITM) attack. Essentially, cheaters can have a software to parse the network packets from servers and get the exact location of other players, which results in a virtually undetectable cheat because the parser could be running on another computer completely isolated from the game and anti-cheat software. At the minimum, the game design should at least minimize information exposure, encrypt network communication, and implement server-side verification if C/S architecture is being used. This will, at minimum, minimize the advantage gained from visualization cheats and make modification cheats unusable.

Game code and memory segment protection will further make it harder for cheat developers to develop cheats. Developing a robust and reliable anti-cheat system is challenging and costly. Thus, buying a license to use an anti-cheat system or contract an anti-cheat company to develop an anti-cheat system is reasonable depending on the cost of such license or contract. This, however, can have a backlash effect from the players within games due to the nature of the techniques used by the anti-cheat system (Rawda, 2020). Some of the techniques used by an anti-cheat company are controversial, such as hooking into Windows' ring0 API "NtCreateFile" and sending a sample of the suspicious process back. NtCreateFile API gets called whenever any process tries to create or open a file (The & Khanh, 2010). Furthermore, the aggressive nature of these techniques has caused problems to the players' system, such as Blue Screen of Death (BSOD) (Gonzalez, 2018).

Such capabilities raise users' privacy protection questions such as what information is being specifically collected and how the user's information is protected. This topic has been one of the most debated topics in gaming communities recently, and for a good reason. Some anti-cheat system companies have stated that they will not collect information unrelated to cheating and will not sell players' information (Battleye, 2016, The Riot Security Team, 2020). However, there have been many scandals related to privacy and internet neutrality in recent years, that this statement holds little to no meaning. Even if this statement holds true, it still does not change the fact that there is potentially a major vulnerability in a player's computer. If an anti-cheat system's server(s) is compromised either by an external attacker or a rogue employee, the privacy of its users will also be compromised. In the worst-case scenario, controls of the anti-cheat system get compromised, and then attackers could potentially steal any kind of data from the user's system.

Investing heavily in the game design, such as minimizing information exposure; underlying architecture, such as mirrored C/S architecture; and adding an appropriate third-party anti-cheat system seems to be the most cost-effective and optimal way to

protect online games from cheaters. The question remains, what does an appropriate third-party anti-cheat system mean? If everything is measured by gain and loss in financial, having a good anti-cheat system might not be optimal. For example, the current game with the highest all-time peak of concurrent players online in Steam gaming platform is PlayerUnknown's Battleground (PUBG). This game uses BE as an anti-cheat system. Battleye, which is currently is one of the most used anti-cheat systems. For example, out of the top five non-free-to-play online games in SteamDB's (2013) charts of most played games; two games use BE, the rest either use their proprietary anti-cheat system or do not use any known anti-cheat system. BE has generated controversy on the internet due to the techniques used (Douggem, 2014). It has the capability to block access to a game's process (thus requiring cheat software to also be operating in ring0); send back files to its server; and remotely execute shellcodes in the player's system.

While BE has a lot of capabilities to prevent and detect cheats, it is however not perfect. There have been reports in 2017 by PUBG and BE that state that they have a daily ban rate of 6000 - 13000 (Duwe, 2017). This means thousands of cheaters have, one way or another, ruined a normal player's experience before getting banned, and since it is a daily ban rate, the normal players' experiences continue to be ruined the next day. But if taken into consideration that cheaters also have to, one way or another, get a new copy of the game before they can cheat again due to bans on their previous accounts. In that case, it is a reasonable assumption that the cheaters themselves generate quite a lot of income for the game developers/publishers. However, without the game's internal statistics of bans and revenues, it is impossible to say for sure whether it is profitable for game developers or game publishers to have a "healthy" number of cheaters in a popular online game or not.

7. Conclusion

What is the difference between cheating prevention and detection in any online game, which solves the cheating problem more effectively, and which one provides a more cost-effective solution? Prevention is done at the game or architecture design level, and it can only truly prevent few types of cheats, but it will be really effective in preventing those. Detection is done after the game has been published, it can be continuously developed, unlike prevention which will be hard to add after the game has been published. Every type of cheats can be detected but depending on the detection method and cheat type, it will be more effective to prevent than detect. Implementing server-side verification, minimizing information exposure to the game's client, and deploying a third-party anti-cheat system is the optimal way from the cost and effectiveness perspective. If there are not enough resources to invest in both prevention and detection, investing more into detection is more effective due to the nature of detection methods and the various types of cheats it can detect.

Many different questions, cheat types, and anti-cheat solutions were left unexplored in this paper due to a lack of resources and scope of this paper. It could be interesting to explore how much cheaters negatively affect normal players and which types of cheat brings the most harm to the game.

References

Alkhalifa, S. (2016). *Machine learning and anti-cheating in FPS games* [Master's thesis, University College London]. ResearchGate https://dx.doi.org/10.13140/RG.2.2.21957.86242

Barreto, C., Cardenas, A. A., & Bensoussan, A. (2017). Optimal security investments in a prevention and detection game. *Proceedings of the Hot Topics in Science of Security: Symposium and Bootcamp*, Hanover, MD, USA. 24-34. https://doi.org/10.1145/3055305.3055314

Battleye (2016, April 18). *Privacy policy*. Battleye. Retrieved September 19, 2021, from https://www.battleye.com/privacy-policy/

Baughman, N. E., Liberatore, M., & Levine, B. N. (2007). Cheat-proof playout for centralized and peer-to-peer gaming. *IEEE/ACM Transactions on Networking*, 15(1), 1-13. https://doi.org/10.1109/TNET.2006.886289

Bauman, E., & Lin, Z. (2016). A case for protecting computer games with SGX. *Proceedings of the 1st Workshop on System Software for Trusted Execution*, Trento, Italy. 4:6. https://doi.org/10.1145/3007788.3007792

Bethea, D., Cochran, R. A., & Reiter, M. K. (2008). Server-side verification of client behavior in online games. *ACM Transactions on Privacy and Security*, 14(4), 1-27. https://doi.org/10.1145/2043628.2043633

Blackburn, J., Kourtellis, N., Skvoretz, J., Ripeanu, M., & Iamnitchi, A. (2014). Cheating in online games: A social network perspective. *ACM Transactions on Internet Technology*, 13(3), 1-25. http://dx.doi.org/10.1145/2602570

Buyukkaya, E., Abdallah, M., & Cavagna, R. (2009). *VoroGame: A hybrid P2P architecture for massively multiplayer games* [Conference presentation]. Consumer Communications and Networking Conference, Last Vegas, NV, USA. https://doi.org/10.1109/CCNC.2009.4784788

Cano, N. (2016). *Game Hacking: Developing Autonomous Bots for Online Games*.

Cassar, S., Montebello, M., & Zammit, S. (2014). *Hybrid peer to peer and server client system for limited users multiplayer first person style games* [Conference presentation]. Conference on Games and Virtual Worlds for Serious Applications, Valletta, Malta. https://doi.org/10.1109/VS-Games.2014.7012161

Chalk, A. (2021, April 23). Warzone director says cheaters are 'ruining some of the best work that I've done in my life'. *PC Gamer*. https://www.pcgamer.com/warzone-director-says-cheaters-are-ruining-some-of-the-best-work-that-ive-done-in-my-life/

Chambers, C., Feng, W., Feng, W., & Saha, D. (2005). Mitigating information exposure to cheaters in real-time strategy games. *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Stevenson, Washington, USA, 7-12. https://doi.org/10.1145/1065983.1065986

Chapel, L., Botvich, D., & Malone, D. (2010). Probabilistic approaches to cheating detection in online games. *Proceedings of the IEEE Symposium on Computational Intelligence and Games, Copenhagen, Denmark, 195-201*. https://doi.org/10.1109/ITW.2010.5593353

Douggem (2014, June 2). Battleye is sending files from your hard drive to its master server [Online forum post]. Reddit. https://www.reddit.com/r/arma/comments/2750n0/battleye_is_sending_files_from_your_hard_drive_to/

Duwe, S. (2017, October 13). BattlEye has banned over 320,000 cheaters from PUBG. *Dot Esports*. https://dotesports.com/pubg/news/pubg-battleye-bans-320k-news-18018

EAC (2018, May 18). *Our Games*. EAC. Retrieved September 21, 2021, from https://www.easy.ac/en-us/partners/

Farlow, S., & Trahan, J. L. (2014). *Client-server assignment in massively multiplayer online games* [Conference presentation]. International Conference on Computer Games, Louisville, KY, USA. https://doi.org/10.1109/CGames.2014.6934147

Feng, W., Kaiser, E., & Schluessler, T. (2008). Stealth measurements for cheat detection in online games. *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, USA,15-20. https://doi.org/10.1145/1517494.1517497

Gonzalez, O. (2018, July 19). 'Fortnite' BSOD: How to Fix the Blue Screen of Death Bug. *Inverse*, https://www.inverse.com/article/47196-how-to-fix-the-fortnite-blue-screen-of-death-bug

Lahti, E. (2018, March 26). Valve has 1,700 CPUs working non-stop to bust CS:GO cheaters. *PC Gamer*. https://www.pcgamer.com/vacnet-csgo/

Liu, D., Gao, X., Zhang, M., Wang, H., & Stavrou, A. (2017). *Detecting passive cheats in online games via performance-skillfulness inconsistency* [Conference presentation]. 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Denver, CO, USA. https://doi.org/10.1109/DSN.2017.20

Maiberg, E. (2014, April 30). Hacks! An investigation into the million-dollar business of video game cheating. *PC Gamer*. https://www.pcgamer.com/hacks-an-investigation-into-aimbot-dealers-wallhack-users-and-the-million-dollar-business-of-video-game-cheating/

Mönch, C., Grimen, G., & Midtstraum, R. (2006). Protecting online games against cheating. *Proceedings of the 5th ACM SIGCOMM Workshop on Network and System Support for Games*, Singapore, 20-es. https://doi.org/10.1145/1230040.1230087

Moffatt, S., Dua, A., & Feng, W. c. (2011). *SpotCheck: An efficient defense against information exposure cheats* [Conference presentation]. 10th Annual Workshop on Network and Systems Support for Games, Ottawa, ON, Canada. https://doi.org/10.1109/NetGames.2011.6080984

Prather, J., Nix, R., & Jessup, R. (2017). *Trust management for cheating detection in distributed massively multiplayer online games* [Conference presentation]. 15th Annual Workshop on Network and Systems Support for Games, Taipei, Taiwan. https://doi.org/10.1109/NetGames.2017.7991547

Rawda, O. (2020, May 6). Understanding Valorant's Anti-Cheat Controversy. *CBR*. https://www.cbr.com/understanding-valorant-anti-cheat-controversy/

Robles, R. J., Yeo, S. S., Moon, Y. D., Park, G., & Kim, S. (2008). *Online games and security issues. Second International Conference on Future Generation Communication and Networking*, Hainan, China. https://doi.org/10.1109/FGCN.2008.199

Schiele, G., Suselbeck, R., Wacker, A., Hähner, J., Becker, C., Weis, T. (2007). *Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming* [Conference presentation]. Seventh IEEE International Symposium on Cluster Computing and the Grid, Rio de Janeiro, Brazil. https://doi.org/10.1109/CCGRID.2007.97

SteamDB (2013, December 8). *Steam Charts*. SteamDB. Retrieved September 19, 2021, from https://steamdb.info/graph/

The, L. B., & Khanh, V. N. (2010). GameGuard: A windows-based software architecture for protecting online games against hackers. *Proceedings of the 2010 Symposium on Information and Communication Technology*, Hanoi, Vietnam. 171-178. https://doi.org/10.1145/1852611.1852643

The Riot Security Team (2020, April 17). A Message About Vanguard From Our Security & Privacy Teams. *Riot Games*. https://www.riotgames.com/en/news/a-message-about-vanguard-from-our-security-privacy-teams

Valve (2016, July 3). *CSGO – Overwatch System*. Steam. Retrieved September 19, 2021, from https://help.steampowered.com/en/faqs/view/65DA-BD12-0DE9-9853

Warren, T. (2020, May 6). The world's biggest PC games are fighting a new surge of cheaters and hackers. *The Verge*. https://www.theverge.com/2020/5/6/21246229/pc-gaming-cheating-aimbots-wallhacks-hacking-tools-developer-response-problem

Webb, S. D., & Soh, S. (2007). Cheating in networked computer games: A review. *Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts*, Perth, Australia. 105-112. https://doi.org/10.1145/1306813.1306839

Yan, J., & Randell, B. (2005). A systematic classification of cheating in online games. *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, USA. 1-9. https://doi.org/10.1145/1103599.1103606

Yu, S. Y., Hammerla, N., Yan, J., & Andras, P. (2012). *A statistical aimbot detection method for online FPS games* [Conference presentation]. 2012 International Joint

Conference on Neural Networks, Brisbane, QLD, Australia. https://doi.org/10.1109/IJCNN.2012.6252489