Anti-Cheat Tool for Detecting Unauthorized User Interference in the Unity Engine Using Blockchain

Michal Kedziora, Aleksander Gorka, Aleksander Marianski and Ireneusz Jozwiak

Abstract The purpose of this paper was to analyse problem of cheating in online games and to design a comprehensive tool for the Unity engine that detects and protects the applications against unauthorized interference by the user. The basic functionality of the tool is detecting and blocking unauthorized interference in the device's memory, detecting the modification of the speed of time flow in the game and detecting time changes in the operating system. The research also examined the current potential of blockchain technology as a secure database for the game. Proposed algorithms where implemented and tested for usability and speed of operation.

1 Introduction

Along with the popularization of smartphones, a number of mobile games with multiplayer elements such as rankings, tournaments and competitions, in which other users participate is rising rapidly. In games of this type, especially those created by small developer studios, ability of users cheating is simplified, because most of the operations are performed on the device on the client side, and only results are sent to the server. This resulted in the creation of many tools enabling cheating in a very simple way that can be used by any person. Most of the mobile games even those one-person earns by means of micropayments, so any fraud in such a game causes financial losses for developers. Instead of buying a given item or improvement, the player will get it using cheat. Losses in multiplayer games will be even greater because the player who in an illegal way it will be in a high place with a ranking with a disproportionately high score may discourage honestly playing players for further play, and thus to spend money in it.

The best example of the size of the computer games industry is the fact that the game World of Warcraft, which premiered in 2004 to date has earned $ 9.23 billion,

M. Kedziora (✉) · A. Gorka · A. Marianski · I. Jozwiak
Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wroclaw, Poland
e-mail: michal.kedziora@pwr.edu.pl

A. Poniszewska-Marańda et al. (eds.), *Data-Centric Business and Applications*, Lecture Notes on Data Engineering and Communications Technologies 40,
https://doi.org/10.1007/978-3-030-34706-2_10

and the game GTA 5, which premiered in 2015, has earned over $ 6 billion and has been distributed in 90 million copies. None of the films ever made even reached half that revenue. A significant portion of GTA 5's revenues was generated through micropayments in multiplayer. However, many players used cheats to facilitate their gameplay and giving an advantage over honest players. Any cheat in a multiplayer game generates a loss. One of the creators of cheats for GTA 5 was convicted by a court in the USA because the software he created generated losses for the company Take-Two Interactive at half a million dollars.

An example of a universal cheating tool, which additionally has versions for Android and Windows is Cheat Engine. Its main functionalities are searching and editing variables in the device's memory as well as accelerating the flow of time in the game. It is able to search for variables of all basic types (int, float, string, double, bytes, etc.). Cheat Engine is a free program released under the free software license [1]. The next programs that have the same functions as Cheat Engine are SB Game Hacker APK, GameGuardian and Game Killer. The main function of all these programs is the manipulation of the device's memory, specifically the search for appropriate variables from the game and their editing to the values given by the user. Apart from interfering with the memory of the device, these programs are also able to change the speed of time in the game [2]. Another type of cheating tools are programs that allow users to bypass the payment requirements in the application. Additionally with the help of interfering tools in the payment system you can make unlimited purchases. Programs offering these functionalities are Creehac and Freedom APK [3]. The basic function of all these programs is to search and change the data stored in the device's memory.

2 Classification of Cheating Methods

Cheating in a computer game, we define as breaking the rules set by the authors of the game. The purpose of cheats is to quickly win the prize and gain an advantage over the opponent, whether it is an artificial intelligence controlled by a computer or the other player [4]. User can cheat in many different ways. One of them is using codes implemented by the authors to accelerate and facilitate testing during the production of the game. This technique is used in single-player games, therefore it does not generate financial losses for game developers and does not adversely affect its perception, and in some cases can add variety to the game. Another way of cheating is to use external software that affects the elements of game mechanics. This method is mostly used and mainly occurs in multiplayer games. It is negatively treated by the community and can significantly affect the developers finances. In addition, the player using such "help" can be banned, which will result in blocking his account and preventing the game. The ban can be temporary or permanent, which will involve the loss of the game, and therefore with the loss of money that has been invested in it. The last way to cheat in computer games that has not been included in this research is to use the bugs of the game itself. This method works well in both

single and multiplayer games. However, it is much less frequently used than external software because finding an error, the use of which can bring any benefits in the game is not easy. Moreover, when such an error is found and made available to a wider group of people, the creators usually quickly release patch to fix software.

2.1 Exploiting Misplaced Trust

An example of this is to give the player too much data that can be used to gain advantage over the opponent. If the player has access to the games logic code or its configuration files, he can easily change them to his advantage. Examples of cheats using this technique are: maphacks (show places where other players or items on the map are located) and wallhacking (allow players to see through the walls, and in some cases even going through walls). Both involve the interception of data that is sent from the server to the client, their appropriate modification and sending them back to the server [5]. These types of cheats also include various types of related frauds with interference in the devices memory. They most often modify the attributes of the player, items in the game world or even global parameters of the game world such as the speed of time flow or the size of the gravitational force. They are even used in games where there are no opponents. Then the most common purpose of the cheater is to speed up the overall pace of the game, which leads to faster gaining appropriate benefits in the game [6].

2.2 Abusing the Game Procedure

This deception is described as a situation in which the player abuses the way the game works and how the player is punished and rewarded. The use of such abuses causes the benefits to be tilted, which should be random to the fraudster's side. An example here is the interruption of the connection from the game in the event of a loss. This will result in not being recognized in the player's ranking, which will make the ratio of winnings to losers false [5, 7].

2.3 Exploiting Machine Intelligence

This deception involves the use of artificial intelligence techniques to gain an advantage over the opponent. An example here can be the use of a game simulation chess with artificial intelligence, while playing with a real opponent. It is enough to copy the opponent's moves to the game board against the advanced artificial intelligence algorithm, and then mirror the movements of the "computer" on the board on which we play with the opponent. Another popular example of cheat using artificial intel-

ligence are aimbots (automatic targeting of the opponent, most often used in FPS games such as first person shooter) or triggerbots causing automatic shooting to the opponent when the player is aiming at him, provides worse results than aimbot, but detection is very difficult [5, 8].

2.4 Modifying Client Infrastructure

The way cheats work using the client's infrastructure is very similar to those described above based on poorly placed trust. However, it does not base actions on modifying the game client and does not use the data received from the server to work. In this case, the device or its drivers are attacked. An example here can be wallhack, which can be implemented by modifying the graphics card drivers, which will change the transparency of the walls [6].

2.5 Exploiting the Lack of Secrecy

This type of fraud can be done in a game where there is no system that cares about the secrecy and integrity of data sent to the server. In this case, the communication packets can be intercepted, changed and sent to a server which, due to the lack of an integrity checking system, will allow entering incorrect data into the database. In addition, if important data is stored in the device's memory in an unencrypted form, trust will also be incorrectly placed, which may lead to even easier interception and modification of data by a fraudster [5]. However, this type of fraud does not only apply to data processed by the game but also to its source code. There are suitable programs that are able to reverse engineer the software. After its completion, the fraudster has access to the source code of the game client, which he can freely change. Then he can run the modified game. This method gives unlimited possibilities and can be used to create virtually any cheat [6].

2.6 Exploiting Lack of Authentication

In a game where there is no proper authentication mechanism, users can collect information about other players' accounts by using a fake server to which they will log in. In addition, after connecting to such a server, the scammers can modify their data, and other players' data and in this way unblock things that they should have access to after making a micropayment. Another case is insufficient server security, which may not stop the cheater from interfering with its functions and gaining access to the database [7, 9].

2.7 Timing Cheating

This method involves delaying the sending of packets by the fraudster, but it does not affect the time of receiving packets. The cheater and other players on his device are not delayed, but other players on their own devices notice the cheater with a considerable delay. Thus, a cheat player has much more time to react, thanks to which he gains the advantage [5].

2.8 Human Factor and Other Methods

All the cheats described above belong to the group of technical frauds and they are characterized by the fact that whether a given fraudster manages to violate the system depends on the strength of security implemented by the authors of the software on both the game and server side. In contrast, frauds in this group are caused by a human factor and the security implemented by the developers have little impact on them. The first type of fraud is account theft, which is caused by carelessness or naivety of the players. Fraudsters use social engineering methods, e.g. phishing, password harvesting fishing, or software to get account data. However, it must be somehow delivered to the player's computer of the game, most often via files downloaded from the Internet for example: guides, add-ons or cheats created by cheaters. Such software includes implemented e.g. keylogger, which registers subsequent keys pressed by the player, and then sends them to the cheater's computer. Collusion fraud is another type. It occurs when a group of two or more dishonest players, e.g. from opposing teams, starts exchanging information that should not be known to the other party. The last type is fraud related to internal abuse. It can be committed by people responsible for the functioning of the game, e.g. administrators or game masters. It occurs when people with more rights than a regular player make it easier to play for selected units, e.g. their friends [7, 9].

2.9 Related Work

A number of papers [10–14] was introduced with aim to research issue of cheating in online games. Detailed systematic classification of cheating in online games was presented by Jeff Yan [9]. Also several tools are already implemented. The most popular tool created for the Unity engine to prevent cheating is the Anti-Cheat Toolkit. It offers functionalities such as protection of variables in memory, provides recognition of the basic types of wallhack, detects flow speed and time changes, protects and extends the built-in system to write to a file named PlayerPrefs [15]. Another tool available to the Unity engine to counteract cheating is PlayerPrefs Elite. The functionalities aimed at improving the security of game data offered by this program

are: detection of unauthorized modification of data stored in the PlayerPrefs system, provision of a secure key manager, and encryption and decryption of variables stored in PlayerPrefs. In addition, the tool extends the PlayerPrefs system provided by Unity libraries with the ability to store arrays and vectors. The next tool similar to previous programs is Anti-Cheat—Protected Variables. Its basic application is protection of fields and properties from memory manipulation, protection of data stored in the PlayerPrefs system and detection of speedhacks. Another existing solution that provides functionality different from the previously described tools is Obfuscator Pro. However, it does not give direct protection against cheaters by detecting them or blocking interference in the program. Its main functionality is to obfuscate the program code by adding a random code or changing the names of classes, variables, functions etc. to unreadable ones. Thanks to this, it provides protection against fraudsters' intrusion into the program code. In contrast, the code itself does not provide virtually no protection against basic frauds and it is necessary to extend it with tools that provide basic functions to protect variables or detect other frauds. In summary, no tool includes both anti-fraud and anti-malware functions. In addition, none of the tools found offer an additional form of data protection, e.g. by storing them on the server, and non of presented solutions uses blockchain technology for this purpose [16–18].

Scams based on artificial intelligence can be detected in three ways. In the first of these, you need other players who, if they observe someone suspicious behavior, can report it through the appropriate system. When a suspicious player is reported by a sufficient number of people, he is checked by the game administration or other most experienced players. People who check potential cheaters have access to their statistics and to video recordings of the game. After looking at fragments of games and statistics, such people decide whether or not a person has illegal support software. The difficulty of detecting cheats based on artificial intelligence depends more on the cheater himself. If he sets the aimbot to maximum effectiveness. So he will automatically aim at unnaturally fast pace to every player who is within the range of the shot, he will be unmasked and banned very quickly. However, a cheater who will use a triggerbot or even aimbot, but in a thoughtful way. So he will set the program so that automatic aiming is activated only when another player is very close to the crosshair, even a person watching the video from the game may not notice that the cheater is using illegal software. The described system can be found e.g. in the game Counter-Strike: Global Offensive where it operates under the name Overwatch.

The effectiveness of the system described above is estimated at 15–30%. To achieve much greater efficiency at the level of 80– 95%, it is necessary to use methods of artificial intelligence, deep learning or dynamic Bayesian networks. These methods, having the appropriate test sets, are able to learn "normal" gameplay, and thus will able to recognize deviations from it, which may be caused, among others, by the use of cheats or the use of game errors. However, a large amount of data is needed for the machine learning system to function properly. The Valve company responsible for Counter-Strike: Global Offensive collected all data from the previously described Overwatch system. Thanks to this, she was able to implement the VACnet system (Valve AntiCheat), which uses deep learning methods to recognize

cheaters. However, for the correct operation of the system based on artificial intelligence or machine learning methods for a large number of players in real time, you need a lot of computing power. For example, Valve AntiCheat needs up to 1,700 processors to work properly [8, 19].

The first and basic barrier that should be used to prevent a fraudster from reading data should be to encrypt it. The data stored in the device's memory should be encrypted and transferred between the client and server. In addition to the data, the scammer may have access to the source code. However, obfuscation transformation will make reading the game client code much more difficult. This technique involves transforming the program to preserve its semantics and at the same time significantly impede understanding. Code obfuscation can be done in many different ways, here are a few: formatting, changing variable and function names, deleting comments, splitting, joining, changing the order of arrays, cloning methods, changing inheritance relationships. The problem with the method of obfuscating the code is that, although it will make it harder to understand the code, the determined attacker will eventually find out how it works. The last and most effective way to secure important game data is to perform all significant operations on that data on the server. Even if the fraudster gains access and changes the data in the device's memory, the server will still perform the operation on its own data and then update it with the fraudster. The main disadvantage of this solution is that the game must have a server with which each client has a permanent connection during the game [6].

The game design itself has a large impact on the occurrence of fraud based on the abuse of procedures and the use of errors. You can prevent such cheats by implementing functionalities into the game that will make these cheats unprofitable for the player. An example of such functionality may be the introduction of a penalty system for players leaving the game during its duration. However, with these types of scams, it is more important and more difficult to detect them, because before developers can fix something, they need to know what is working incorrectly. Detection of this type of error occurs by collecting various types of data about players, and then their appropriate association by analysis carried out by people or programs using methods of artificial intelligence. In this way, for example, you can detect the connection between players who have a higher win/lose ratio than players who leave the game more often [6].

3 Proposed Anti-Cheat Solution

The solution has been designed for games containing multiplayer elements such as rankings, tournaments and trade between players, created mainly for mobile devices. At the moment there is no generally available aimbot or triggerbot operating on a mobile device. For this reason, the project has omitted protection against fraud by using artificial intelligence algorithms for its operation. The most common feature provided by almost all cheating tools available on the Android system is to search for cached data and freely change them. Therefore, the main functionality of the

implemented software will be to detect attempts to interfere with this data and to prevent this interference. This protection will be implemented in two versions. An online version using blockchain and an offline version that uses the cryptographic hash function to check the integrity and authenticity of the data. Another feature commonly available in various types of cheats is speedhack. It allows the player to increase or decrease the speed of time in the game, which can significantly affect the speed of the player's progress. For this reason, one of the additional functionalities of the designed anticheat algorithm will be the detection of such interference in the speed of time flow. Often, games designed for mobile systems have a system of awarding players daily login rewards or depend on many factors since system time for example, the amount of raw materials the player receives or the time it takes to expand buildings. Therefore, very frequent cheating in games is interference in the system time of the device. It does not require any additional software and is very easy to carry out, and in unprotected games can bring great benefits to cheating players. In the designed tool, the appropriate function will be implemented that communicates with the server from which time will be collected.

3.1 Detection of Interference in the Device Memory

Tampering with the device's memory can be hindered and detected in several different ways. Effectiveness of method depends mainly on whether the game processes data on the client or server side. In games that do not have any multi-player elements the implementation of additional systems could transfer only negative effects, e.g. in the form of a decrease in performance. Another situation occurs when the game has multiplayer elements, but the whole logic is performed on the client's device, and only players results are sent to the server. In this case, implementation of solutions detecting cheating is necessary. However, due to the fact that data operations are performed on the client's device, the implemented methods will never give 100% resistance. However, they may make it more difficult to intervene in the data so that it will be unprofitable, and for most people who do not have adequate knowledge and skills will make it impossible. Locally, data can be secured in several different ways. One of them is to duplicate data and store it in several versions, and then compare whether these versions differ from each other. Different versions mean that one of them has been changed unauthorized. Another way is to encrypt data stored in memory and decrypt them when read and changed. For encryption we can use simple algorithms, but some cheat programs are able to find and change even such encrypted data. The last type of security can only be used in multiplayer games that require a constant connection to the Internet. It consists in storing all data and performing all operations on the server, while the game client only provides a graphical interface. In the our proposed solution, instead of a standard server, a blockchain was used to ensure greater security and greater data integrity.

The first step is the installation and configuration of the Loom environment. Launched locally, DAppChain acts as the side chain of the Ethereum Blockchain

network. The Loom SDK is a basic component of the Loom network. The Loom network is a platform for building scalable side chains in Ethereum, with particular emphasis on social games and applications. The Loom SDK helps developers create their own blockchain without having to understand the details of its infrastructure. The Loom SDK generates DAppChain, which uses Ethernum as the base layer. Launching DApp as an Ethernum side chain (sidechain) enables the use of an alternative set of rules such as DPoS to achieve high scalability while providing security guarantees at the Ethernet level. Delegate Proof of Stake (DPoS) is a consensus algorithm that relies on a reputation system and continuous voting through which a team of trusted entities (delegates) is selected. Delegates can create and add blocks to the chain and include specific transactions in the blocks. All their activities are publicly available, and real-time digital democracy will promptly dismiss an inefficient or malicious delegate. All services offered by the Loom network use LOOM tokens. Tokens can be used as currency in Loom SDK-based applications and games. In addition, they are needed by developers whose application uses the Loom network to pay for licenses and side chain usage bandwidth measured in transactions per second. Loom does not use player devices to process transactions. Therefore, each transaction is indirectly processed by the Ethernum network which is connected with the generation of costs charged to the developers whose game uses the side chains of this network. Next, the Loom SDK package was imported to the Unity environment and the appropriate functions connecting and communicating with DAppChain were implemented.

```
async void Start()
{
 if (privateKey == null)
  {
   privateKey = CryptoUtils.GeneratePrivateKey();
   publicKey = CryptoUtils.PublicKeyFromPrivateKey(privateKey);
  }
   var contractEvm = await GetContractEVM(privateKey, publicKey);
   await StaticCallContractEVM(contractEvm);
   contractEvm.EventReceived += ContractEventReceived;
  await CallContractEVM(contractEvm);
}
```

The Start() function is executed when the application is started. Its task is to generate the private and public key needed for communication with blockchain. These keys must be unique to each user from the application. Next, an intelligent contract instance is created based on the keys. When you connect to DAppChain via contracts, you can send and save data in it. The status of a smart contract can be changed by signing a transaction and sending it to blockchain, which verifies its correctness. Most of these functions are performed using the Contract.CallAsync() method provided by the Loom SDK. In the body above the method, the function of an intelligent contract is defined. During the implementation, the GetMsg() and SetMsg() functions were used. Their task is to collect the value stored under a given key in sequence and change the value assigned to a given key in DAppChain.

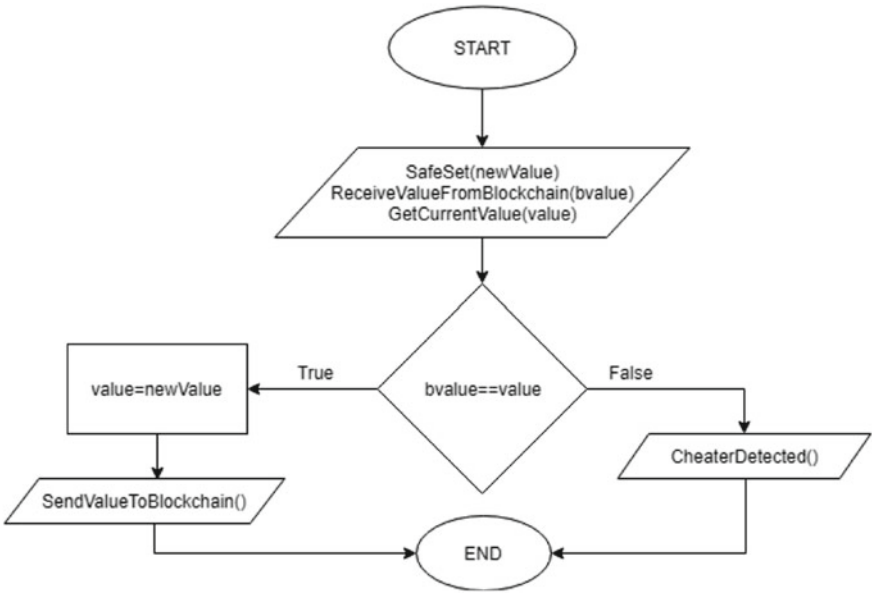

Fig. 1 Algorithm for detecting device memory interference using blockchain

```
private async Task CallGetContract(Contract contract)
{
  var result = await contract.CallAsync<MapEntry>(``GetMsg'', new MapEntry
   { Key = this.key });
}
private async Task CallSetContract(Contract contract)
{
 await contract.CallAsync(``SetMsg'', new MapEntry
  {
   Key = this.key,
   Value = this.value.ToString();
  });
}
public async void SafeSet(int value)
{
  await CallGetContract(contract);
   if (this.blockchainValue != this.value)
    {
     Debug.LogError(``CHEATER DETECTED!'');
     return;
    }
   this.value = value;
   await CallSetContract(this.contract);
}
```

To protect against fraudsters interfering with the device's memory, a special algorithm has been implemented, the scheme of which is visible in Fig. 1. Every time the program intervenes in a variable, this algorithm is executed. The algorithm implementation function retrieves values from the blockchain and compares it with the

current value stored in the memory. If they are the same, the variable is changed and the contract executed, which updates the value in the blockchain. The problem in this case is that variables stored in DAppChain can not be updated too often, because this will cause too much load on the blockchain.

3.2 Data Integrity Protection

Earlier we described two methods of data protection against unauthorized user interference. The first was to keep a copy of the data, but if the copy of the data is kept in the same form as the original, with the help of the appropriate software user can easily find and change it with the original. The fraud will not be detected then. The second method was data encryption. But the use of simple encryption algorithms can be broken by a suitably advanced cheat, and the use of advanced encryption methods can affect performance. For these reasons, a combination of these two solutions has been implemented as in Fig. 2. Individual variables will be copied and a hash function will be called for them, and then, when performing each operation on a particular variable, it will be compared with the original. If a difference is detected, it means that the data has been changed in a different way than with the authorized operations.

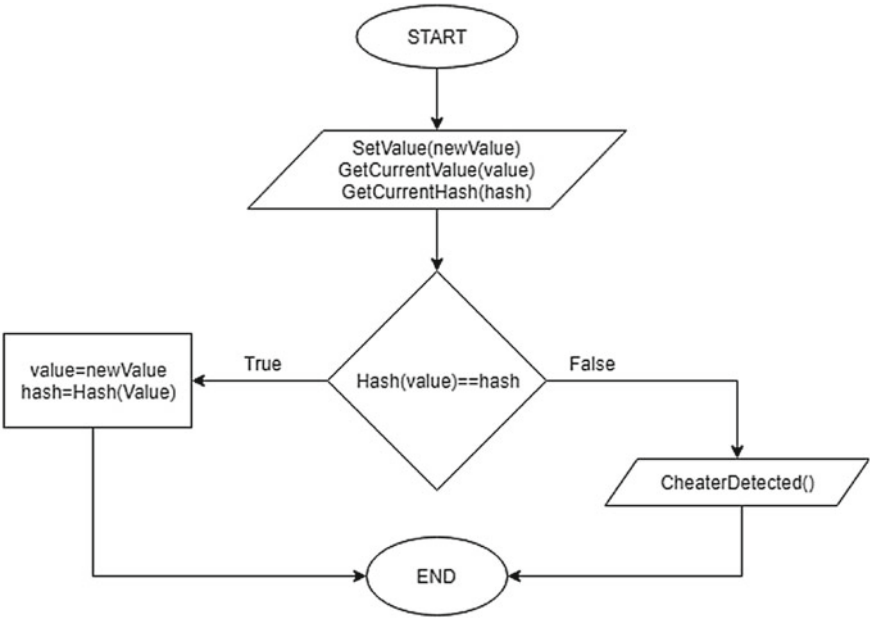

Fig. 2 Algorithm for detecting memory tempering using cryptographic hash functions

For each of the basic types, the overriding class has been implemented e.g., the int has been replaced by the HashInt class.

```
public class HashInt : IEquatable<HashInt>
{
    public int Value;
    private byte[] hash;

    private HashInt()
    {
        Hash(this);
    }

    private HashInt(int value)
    {
        this.Value = value;
        Hash(this);
    }

    private static void Hash(HashInt hashInt)
    {
        var sha = new SHA256Managed();
        hashInt.hash = sha.ComputeHash(Encoding.UTF8.GetBytes(hashInt.Value.ToString()));
    }

    private static bool IsCheated(HashInt hashInt)
    {
        if (!Compare(hashInt))
        {
            Debug.LogError(``Cheater detected!'');
            hashInt.Value = 0;
            return true;
        }

        return false;

        }
    private static bool Compare(HashInt hashInt)
    {
        var sha = new SHA256Managed();
        return hashInt.hash.SequenceEqual(sha.ComputeHash(Encoding.UTF8.GetBytes(hashInt.
     Value.ToString())));
    }

    #region operators, overrides, interface implementations

    public static implicit operator HashInt(int value)
    {
        var hashInt = new HashInt(value);
        IsCheated(hashInt);
        Hash(hashInt);
        return hashInt;
    }

    public static implicit operator int(HashInt hashInt)
    {
        IsCheated(hashInt);
        Hash(hashInt);
        return hashInt.Value;
    }

    public bool Equals(HashInt other)
    {
        return other.Value.Equals(Value);
    }
```

```
        public override string ToString()
        {
            return Value.ToString();
        }

        #endregion
    }
```

Each class has two fields, one with the value in the normal form, and the other with the value in the form of a hash. As a hash algorithm, the 256-bit SHA-2 version was used. The key role in the classes is in the functions overwriting all basic operators, because before each of them is called, and therefore when each attempt to edit the variable is checked the compatibility of the value field with hash. If the fields do not match, then the variable has been edited unauthorized and it is reset. In the event that the game connects to the server in this place, a function can be placed that sends information that the player has tried to commit fraud. This could result, for example, in placing the player on the black list and not taking it into account in the rankings.

3.3 Timing Cheating

The speed of playing time can be changed in two ways. The first is to find and change the field stored in memory, responsible for the time scale. Second way to change the speed of the game is more complex. To understand it, you should know that each program communicates with the operating system kernel using predefined functions called system calls. Each operating system has a different set of calls, but often performs similar tasks such as allocating memory, reading and writing files or handling processes. One of the key information processed by the system is knowledge of time. If a computer game is to render 60 frames per second, the rendering function is called average every 16.6 ms. In Windows, the GetTickCount() function returns the number of milliseconds that have passed since the appearance of the system. Thus, the second way to Speed Hacking works by injecting the code into a running process and taking control over the timing function. The purpose of this is to return accelerated or slowed ticks so that you modify the speed of the program. However, the use of this method causes a "crossover" between the time from the start of the game measured by the game environment and the one measured by the operating system. This knowledge has been used to implement a function that detects cheating rate swings [1]. At the moment of starting the program, the start time taken from the system and the game environment is saved. Data is saved and also compared in the form of a number of ticks, in order to achieve the highest possible accuracy. The IsSpeedhackDetected() function is called in the Update() function, which is performed in every game frame (Fig. 3).

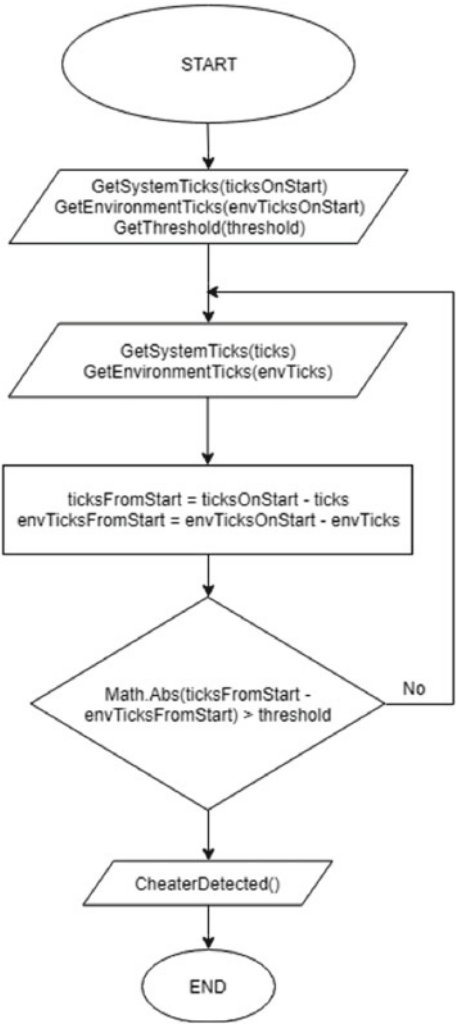

Fig. 3 Algorithm for detecting Timing cheating

```
private bool IsSpeedhackDetected()
{
    var ticks = System.DateTime.UtcNow.Ticks;

    if (ticks - previousTicks < INTERVAL_TICKS) {return false;}
    previousTicks = ticks;
    var ticksFromStart = ticks - ticksOnStart;

var vulnerableTicks = System.Environment.TickCount *
System.TimeSpan.TicksPerMillisecond;

var ticksCheated = Mathf.Abs((vulnerableTicks
- vulnerableTicksOnStart) - (ticksFromStart)) > THRESHOLD;
```

```
        if (ticksCheated)
        {
            ResetTicks();
            return true;
    }
        return false;
```

The Update() method is the basic function in the Unity engine. Thus, the program in each frame checks if the system time taken using the System.DateTime.UtcNow. Ticks property differs from the time of the currently used environment that is accessed by using the System.Environment.TickCount property. If the difference between these values rises above the set threshold, the change in the speed of time will be detected.

3.4 Modifying Client Time Settings

Time and date change detection refers to games that reward a player for coming into play at a given time. Thus, the method described earlier will no longer work because the player accrues bonuses when he is not present in the game and he is rewarded when entering the game. Thus, in this case, the game very often uses the system date and time, which can easily be freely changed in the system options. This can be prevented by downloading the time from the server and comparing it with the system time of the device. The TimeCheatingDetector class has been implemented, which provides a static Start Detection function. This function, at some fixed default for one minute, takes time from the server and checks if it differs from the system time by the amount defined as the threshold and set by default to 5 min (Fig. 4).

4 Practical Tests

Testing of the implemented functionalities have been carried out with the help of Cheat Engine software. For the function of hashing the variable copy, an attempt was made to find the appropriate variable in memory and to edit it later. Undoubtedly, the results obtained were very interesting. While trying to find a variable, it was not found by the program. The reason for this was the function overwriting the operators, because after using any of them a new object was created that overwrote the old one. Thus, each change in value was also associated with the change of the address of the variable. This prevented the Cheat Engine program from narrowing the number of addresses, which resulted in the lack of finding the right one. To test the effectiveness, a special test function was implemented that changed the value without creating a new object. After this change, the program found the value after three scans, but unauthorized change was detected by the implemented classes each time. The results of finding a given field by the Cheat Engine for the fixed and

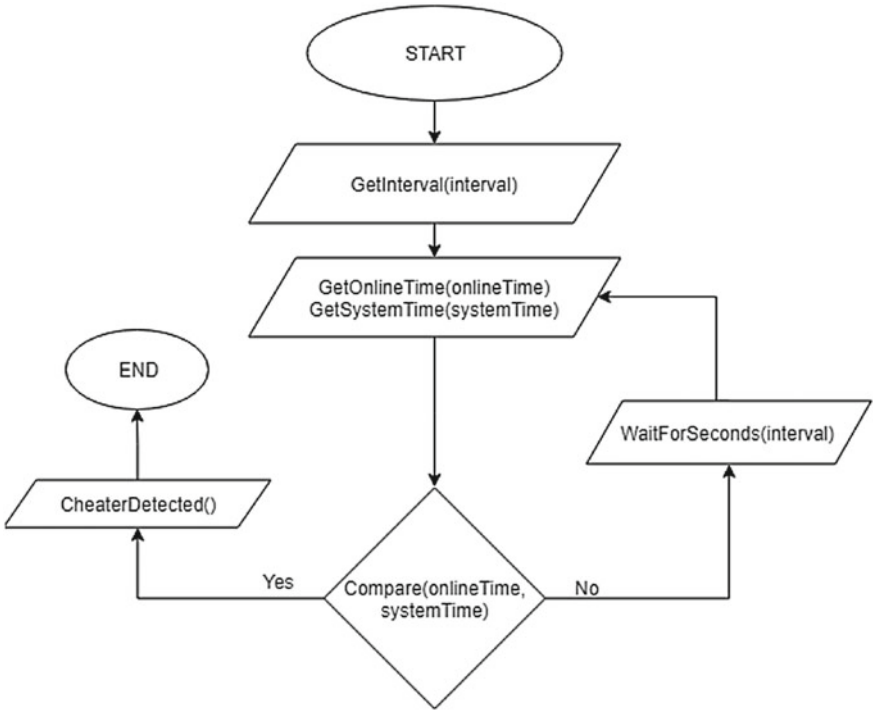

Fig. 4 The algorithm for detecting time change fraud using an external server

variable address of the variable are presented in Table 1. The tests have shown that the solution works in the intended manner, and moreover, it has not one but two barriers to interfering with memory. The first one is dynamically changing value indicators, making it difficult for programs to find the right address, and the other hash copies of variables that ensure the integrity and accuracy of the data. Next the test was carried out for the function detecting the change of the game time speed. The Cheat Engine program speedhack was used, and then the speed has been set several times to both values less than 1 and bigger. At the detection threshold of 500 ms, the detection time was practically instantaneous. However, the change in detection threshold for 5 s detection time is so long that it creates the theoretical possibility of cheating. The player or program can jump from the acceleration value 2–0.5 every 2.5 s and in this way after every 5 s the difference between the time measured by the environment and the time returned by the system will be 0. Implementation of such a system for cheating would require the necessity of knowledge with a specific detection threshold size. Therefore, if the detection threshold is drawn from between a small range of 500 to 1000 ms each time it is checked, the meaning will make it difficult to implement a program that jumps between values and does not "travel" in time. What's more, this program could theoretically be useful only in arcade games, in which the player can take a break between successive short sequences. In this case,

Table 1 The number of variables found after the Cheat Engine performed subsequent scans for a statically and dynamically changed address

No change of variable address				Dynamic change of variable address			
Scan 1	Scan 2	Scan 3	Scan 4	Scan 1	Scan 2	Scan 3	Scan 4
91	1	1	14231	3	0	0	12144
2	1	1	13665	6	0	0	13713
12	1	1	13726	2	0	0	13923

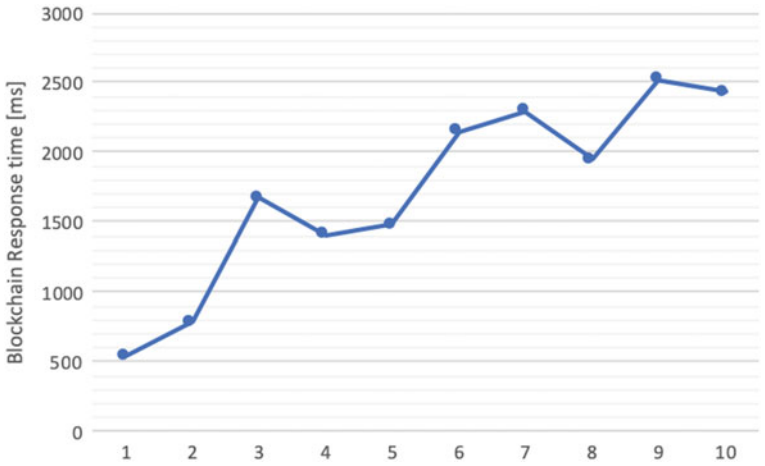

Fig. 5 Blockchain response times for subsequent attempts to download and change data

the player could perform the arcade sequences when the time is slowed down and wait a while when the time is accelerating, but for values below 1 s, it will not be in practice, useful (Fig. 5).

The blockchain has been tested for contract processing speed and for detecting interference in its memory. The tests were carried out in a local environment, and therefore the real results that would be carried out on a normally operating blockchain could be significantly different, most likely the time received would be greater. In addition, blockchain was launched on Windows at the Windows Subsystem for Linux layer. The response time was measured in the game and it is the time from the start of the value retrieval transaction to receipt of confirmation of its change from the block chain. The times obtained are shown in Table 2. They ranged from about 0.5

Table 2 Blockchain response times for subsequent attempts to download and change data

	1	2	3	4	5	6	7	8	9	10
Response time [ms]	543	2145	1675	1402	1476	786	2297	1945	2523	2428

to even 2.5 s. A performance test similar to those described in previous paragraph was carried out using Cheat Engine detected all attempts to interfere with memory and prevented them by overwriting the changed value by the one taken from the blockchain.

5 Conclusion and Future Work

The purpose of this research was to design and implement a solution for the Unity engine, detecting the interference of external software in the device's memory, and in the way the device works. This goal has been achieved, and the designed and implemented algorithms has been tested for functionality and effectiveness. To conduct an attempt to interfere in the data and operation of an application built using the Unity engine using a written tool, one was used from the most popular cheat programs—Cheat Engine. The application detected any attempt to interfere with external software. In addition, after each operation, the tool dynamically changed the addresses of the fields on which the operation was performed practically made it impossible to find the appropriate variables using the Cheat Engine program. At the current stage, the software provides protection against three popular cheat methods. However, in the future it can be extended with the function of code obfuscation and securely save and read data from both the device and the server. In addition, a blockchain set as a side chain of the Ethernum network was used as a secure database for games and applications created using the Unity engine. However, tests have shown that at present due to its limitations in the form of the number of transactions processed in each second and the amount of time needed to execute transactions, using it only for the purposes of data security of the game does not make sense. The exception may be a situation in which micropayments in the game are implemented using cryptocurrencies, because the number of operations is then many times smaller and on each of them the application developers earn, which reimburses the costs they have to bear for the maintenance of the blockchain.

References

1. Feng WC, Kaiser E, Schluessler T (2008). Stealth measurements for cheat detection in on-line games. In: Proceedings of the 7th ACM SIGCOMM workshop on network and system support for games. ACM, pp 15–20
2. Heo GI, Heo CI, Kim HK (2015) A study on mobile game security threats by analyzing malicious behavior of auto program of clash of clans. J Korea Inst Inf Secur Cryptol 25(6):1361–1376
3. Bremer J (2013) Automated analysis and deobfuscation of android apps & malware. Freelance Security Researcher
4. Consalvo M (2007) Cheating: gaining advantage in videogames. Massachusetts Institute of Technology, pp 5–8

5. McGraw G, Hoglund G (2007) Online games and security. In: IEEE Security & Privacy, vol 5, no 5, pp 76–79
6. Joshi R Cheating and virtual crimes in massively multiplayer online games. Royal University of London
7. Tolbaru SA (2011) Cheating in online video games, University of Copenhagen, Datalogisk Institut, 15 Aug 2011
8. Yeung SF, Lui JCS, Liu J, Yan J (2006) Detecting cheaters for multiplayer games: theory, design and implementation 1178–1182
9. Yan J, Randell B (2005) A systematic classification of cheating in online games. In: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games. ACM
10. Park JK, Han ML, Kim HK (2015) A study of cheater detection in FPS game by using user log analysis. J Korea Game Soc 15(3):177–188
11. Ahn D, Yoo B (2017). A study of cheating identification and measurement of the effect in online game
12. Raymond E (1999) The case of the quake cheats, Unpublished manuscript
13. Baughman N, Levine B (2001) Cheat-proof playout for centralized and distributed online games. In: Proceedings of the twentieth IEEE INFOCOM conference
14. Li K, Ding S, McCreary D (2004) Analysis of state exposure control to prevent cheating in online games. In: The 14th ACM international workshop on network and operating systems support for digital audio and video (NOSSDAV)
15. Cano N (2016) Game hacking: developing autonomous bots for online games. No Starch Press
16. Olleros FX, Zhegu M (2016) Research handbook on digital transformations, Edward Elgar Publishing
17. Swan M (2015) Blockchain: Blueprint for a new economy, O'Reilly Media
18. Zyskind G, Nathan O (2015) Decentralizing privacy: using blockchain to protect personal data. In: Security and privacy workshops (SPW), 2015. IEEE, pp 180–184
19. McDonald J (Valve), Using deep learning to combat cheating in CSGO