CSCE 665 Project Proposal
The Detection and Countermeasures of Cheat Engine

Lijiu Liang
Texas A&M University
UIN:928000822
liang.lijiu@tamu.edu

Zhenlei Song
Texas A&M University
UIN:862009388
songzl8_950606@tamu.edu

Abstract

Cheat Engine (CE) is an open-source memory scanner for Windows operating system, which is mostly used for cheating in PC-games. Cheat Engine can view the disassembled memory of a specific process and allow the alteration of game data to break the normal game routine. To keep their productions away from illegal alterations or cheating, PC-game designers take some measures to detect Cheat Engine and prevent it create a thread in its own process. Meanwhile, the existing methods of detection and countermeasures are deficient. This project does an investigation of current popular detecting method and implements a new method to detect and counter Cheat Engine by using Windows API.

1. Introduction

Cheat Engine [3, 10] is a free and open-source memory scanner created by Eric Heijnen ("Dark Byte"). It is developed on Lazarus[8] IDE for 32 and 64-bit versions of Windows. Most parts of Cheat Engine are written in Object Pascal, while the kernel modules are written in C. The most famous feature of Cheat Engine is viewing the disassembled memory of a PC-game process based on scanning changes in specific value. After knowing the address information of certain data, Cheat Engine allow the alternation of game states to give the user advantages such as infinite health, changing the number of items, or even clearing stages without playing. Cheat Engine also has some Direct3D manipulation tools that support vision through walls or creating amibots. Another important feature of Cheat Engine is, it allow users to share their addresses locations or Lua scripts via Cheat Table, a file with extension *'.CT'* that can be opened by Cheat Engine.

For the purpose of altering memory value, Cheat Engine scans the virtual memory section of the target process and finds the accurate address of certain value by detecting the value's change. However, game processes are executed on Ring 3 [4]of the privilege ring for the operating system. Ring 3 has no privilege to access physical memory and change the value. Cheat Engine can access Ring 0 by using Windows API. The Windows API (WinAPI)[6, 13] is Microsoft's core set of application programming interfaces available in the Microsoft Windows operating systems. WinAPI is the most direct method for almost all Windows programs to interact with Windows operating system. As shown in Figure 2[5], Cheat Engine calls *kernel32.dll* first and finally accesses *ntoskrnl.exe*, which offers various system services such as hardware abstraction, process and memory management[11].

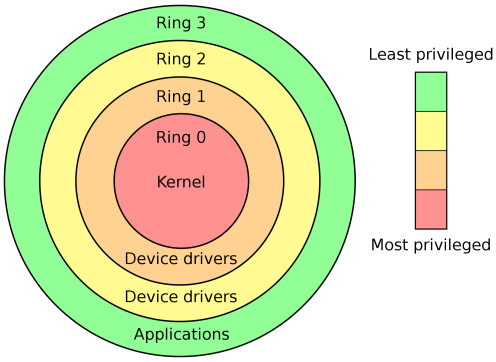

Figure 1. Privilege rings for x86 Windows [12]

Meanwhile, Windows API calls can be forbidden by using hooking [1]. In that case, Cheat Engine also provides APIs by itself. These APIs are offered by *dbk32.sys* and are basically similar to Windows APIs. Hence, Cheat Engine can bypass hooks implemented by game designers and access the memory of game processes all the time.

Game designers always take the detection of Cheat Engine into consideration when they design a PC-game. Although Cheat Engine is mostly applied in single-player Console games, which means it does no harm to other players directly, it is still unfair for those players who consume

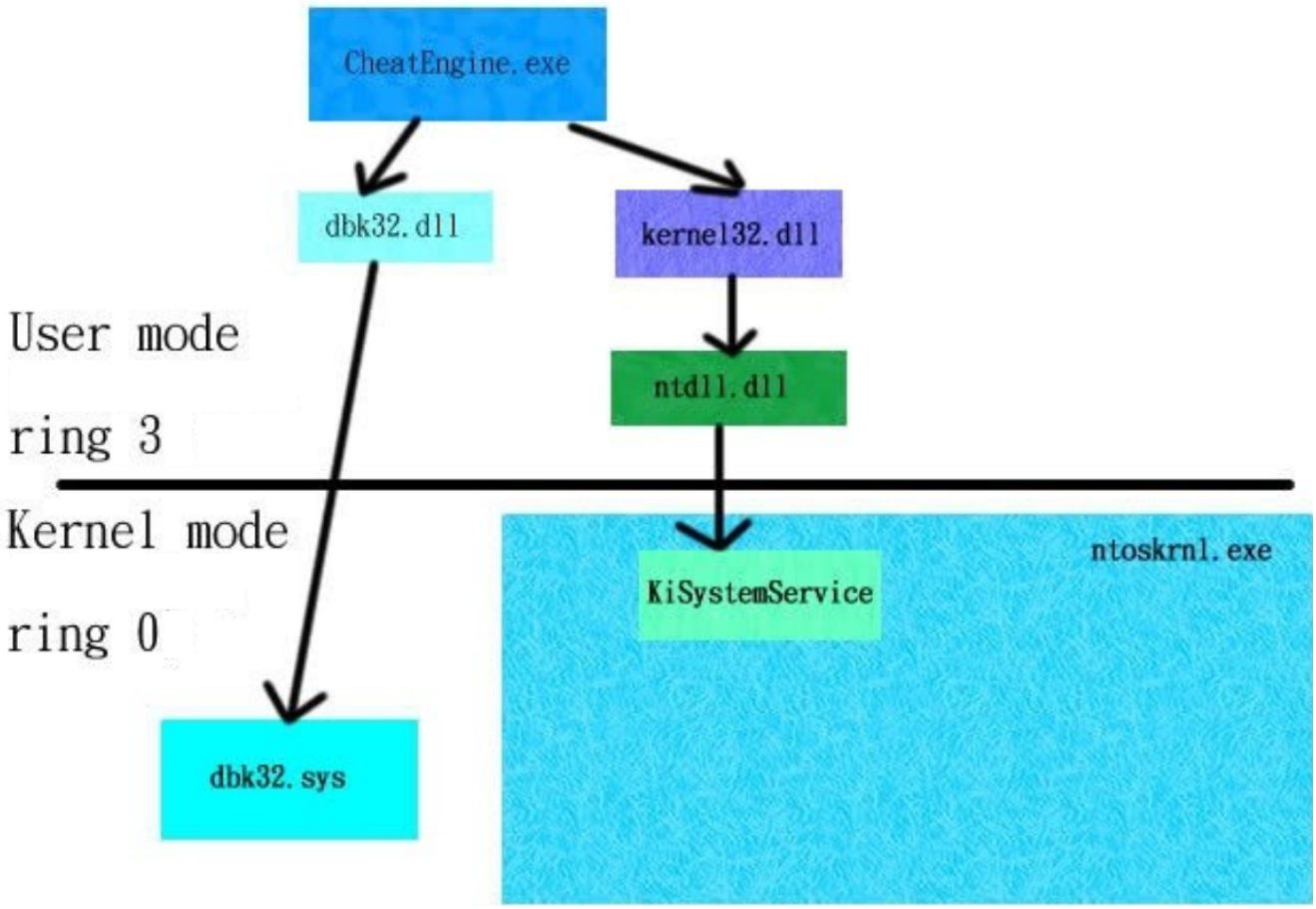

Figure 2. Methods for Cheat Engine to access Ring 0

a lot of time to achieve the same compliments. However, the existing method of Cheat Engine detection is simply scanning the name of the running executable files in background. Once Cheat Engine is found and the version number is matched, game process calls some Windows APIs to protect itself, such as quitting the game or even reboot the whole operating system. Users can easily bypass this detection by changing the name and version number of Cheat Engine in Lazarus[2]. On the other hand, if game designers implement complicated detection in their game, the detection itself will consume too much performance and the game experience will be decreased.

The motivation of this project is to find out a new method to detect and counter Cheat Engine while not so consumptive. We will introduce detailed mechanisms of Cheat Engine and demonstrate the way of detection and countermeasures.

2. Survey

We investigated the popular way to detecting Cheat Engine applied by some famous enterprises. Steam® maintains a blacklist that is updating rapidly. Their famous anti-cheat system VAC[7] scans the environment of clients and matches the name appears in their blacklist. They also take snapshots for some specific flags in a game and upload to their database, then they analyse them manually.

Another famous anti-cheat program is TenProtect (TP)[9] by Tencent®. TP implements many hooks on Windows OS and hence it has the most significant effect to count any form of cheat, including Cheat Engine. However, this anti-cheat tool seriously relies on the current environment of client's computer. Crushes can be caused by any change on client's OS. TP is notorious by its frequent update and corrupting the performance of OS.

To sum up, it is difficult to find a balance point for an anti-cheat method. And for those PC-game developer, they prefer not to apply sophisticate methods to detecting Cheat Engine. Any form of detection may slow down the process of the game is the more critical fact for most of PC-game developer.

2.1. Countermeasure

There are mainly two classes of Cheat Engine countermeasures for games: external methods and internal methods. For external level, WinAPI offers a function *TerminateProcess* to kill any process by passing the handle of that process. This is the most direct way to counter Cheat Engine: as long as we detect it, we kill it instantly. However, this function requires administrator privilege. In other word, the PC game implemented this approach has to running under administrator privilege. This will lead serious security issues. The safer way is implementing an internal method that manipulates the game itself, like quitting the game itself or applying other method to prevent cheaters befitting from Cheat Engine. For instance, the "save" option may be disabled after Cheat Engine is detected. In the famous online game *Monster Hunter: World*, Capcom® disable the Internet connection service to prevent cheaters from infecting other players.

According to above reason, the countermeasures varies from different games. Game designers should propose an internal method based on the mechanisms of the game.

3. Detection

We implement two naive methods to detect Cheat Engine and using WinAPI call to kill the Cheat Engine's process as a countermeasure first.

3.1. Detection based on scanning all processes

In order to access the basic information of a process, function *CreateToolhelp32Snapshot* takes a snapshot of the specified processes. We can get the process ID and the name of the executable file for this process. And function *Process32First* and *Process32Next* are used to traverse the current processes in the background. Combine these functions, we can scanning all of the current processes and search for the name of executable file by matching with a blacklist.

As we mentioned before, this approach can be easily bypassed by modifying the source file of Cheat Engine in Lazarus. The name of executable file can be customized randomly and the blacklist will fail to match the name.

3.2. Detection based on opening window

Another approach to search whether the process of Cheat Engine is running is searching the current opening windows. Up to the latest released version of Cheat Engine, it has to work with a GUI window opened. So there must be a window for Cheat Engine when it is working.

This approach can evidently reduce the workload of detection, for the reason that the amount of opening windows is significantly less than the amount of current processes. Function *FindWindow* will find the window of Cheat Engine by exactly matching the name of the window. After searching, if there exists a target window, function *GetWindowThreadProcessId* accesses the pid of this process.

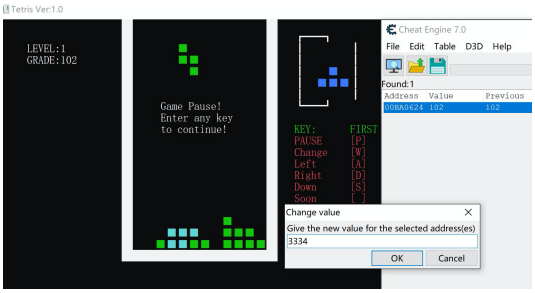

Figure 3. Tetris Data Modify [12]

However, this approach has a considerable limitation. It has to match the window by exact name. Although the exact name of windows can be accessed by function *GetForegroundWindow*, this approach again can be bypassed by modifying the source file of Cheat Engine in Lazarus. We are trying to extract another feature to identify the process of Cheat Engine.

To overcome the above limitations, we implement a improving method using hash

3.3. Detection based on data integrity

The main purpose of cheat engine is to modify essential data in games. The idea of checking the data integrity come up then. Normally essential is stored in memory itself. If we calculate its hash value each time when writing, and check if the 'data-hash' pair match when reading. We can find out if there exists illegal writing operation. Here we present an example in a classic game 'Tetris' shown in Figure 3.

We pretend to be a cheater using cheat engine to modify the score from 102 to 3334. Then at the end of current round, this action is found. Then force to quit game process at once.

3.4. Detection based on DLL integrity

Besides data aspect, the other purpose using cheat engine is to manipulate the functions in games. Like in shooting games, cheaters can replace API of clicking mouse to auto-aiming shooting function. By modifying Windows system DLL, or modifying DLLs provided by games.

In this aspect, we came up with that either to change Windows API or replace game API, the ultimate result is to modify DLL. Thus, our detection measure is to detect the integrity of all DLLs and EXE files repeatedly. Calculate Md5 value of all lib files, if any of them is changed, it must be illegal. Then a cheat action has been found. The MD5 format is shown in Figure 4.

Figure 4. Tetris Data Modify [12]

4. Conclusion

This project starts form the facts that video games cheats in recent years and cheat engine ones are hard to be detected. Plus current protection methods are easy to be escaped. Like in Steam and TP, are basically based on users' reports. This method is not sufficiently effective and takes time to respond as well. Furthermore, they almost have no actions toward cheat engine at all.(We took experiments on steam games, using cheat engine to change scores, and there was nothing happening)

Besides, the other methods publicly used towards cheat engine are to check processes names. Which is even easier to be escaped by cheaters.

In contrast, our main approach against cheat engine is according to methods of cheat engine itself. Data integrity and DLL integrity. The main idea of both approaches are similar to each other, which is to real-time detect the illegal modification. And in out tests, both approaches works well. All attacks from cheat engine were found and processes terminated.

References

[1] Jonathan D. Hooking explained: detouring library calls and vtable patching in Windows/Linux/MAC-OSX. http://ntvalk.blogspot.com/2013/11/hooking-explained-detouring-library.html. Accessed February 13, 2020.

[2] Cheat Engine Forum. Bypass detection. https://forum.cheatengine.org/viewtopic.php?t=582912&sid=4b1a146e269204a5558f5c1db330389f. Accessed February 14, 2020.

[3] Eric Heijnen. Official website of Cheat Engine. https://www.cheatengine.org/. Accessed February 13, 2020.

[4] P. A. Karger and A. J. Herbert. An augmented capability architecture to support lattice security and traceability of access. *1984 IEEE Symposium on Security and Privacy*, 2(2), 1984.

[5] Yuren Lin. How does cheat engine work. http://bbs.52miji.com/thread-93007-1-1.html. Accessed February 13, 2020.

[6] Microsoft. Windows API Index of MSDN. https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list. Accessed April 2, 2020.

[7] Steam®. Valve Anti-Cheat System (VAC). https://support.steampowered.com/kb/7849-RADZ-6869/. Accessed April 2, 2020.

[8] Lazarus Team. Official website of Lazarus IDE. https://www.lazarus-ide.org/. Accessed February 13, 2020.

[9] Tencent®. Security Service Centre of Tencent Game. https://gamesafe.qq.com/safety.shtml. Accessed April 2, 2020.

[10] Wikipedia. Cheat Engine. https://en.wikipedia.org/wiki/Cheat_Engine. Accessed February 13, 2020.

[11] Wikipedia. Ntoskrnl.exe. https://en.wikipedia.org/wiki/Ntoskrnl.exe. Accessed February 13, 2020.

[12] Wikipedia. Protection ring. https://en.wikipedia.org/wiki/Protection_ring. Accessed February 13, 2020.

[13] Wikipedia. Windows API. https://en.wikipedia.org/wiki/Windows_API. Accessed April 2, 2020.