# A Survey on Aimbot

Ya-Hsin Chang, Yu-Cheng Liao, Yu-Shian Lin, Wei-Lun Shih
National Taiwan University
Taipei, Taiwan
r09922066,r09922172,r09944005,r09944048@ntu.edu.tw

## Abstract

In this work we investigate aimbot, a widely-seen cheating in FPS games. We will introduce popular methods to implement an aimbot, analyze the techniques they use, discuss the present defenses, and investigate on developing detection methods.

## 1 Introduction

Online game has grown into a huge industry. With the population and money involved, cheating in games is also being rapidly developed. However, since the defensive side is not willing to reveal their detecting methods and strategies, people can hardly understand what is happening and researchers have little information to help with the problems.

In this work, we will show an example of how a present cheating program is built, what vulnerabilities are utilized, and discuss the current and future defenses. Although there are some works showing the techniques, most did not demonstrate how those are combined in one application. Because the real-world situation is often more complicated than separate techniques, we want to thoroughly examine an example and analyze the content.

We choose aimbot as the example. Every player knows it. It is such an infamous cheating in first-player shooting (FPS) games that locates, targets, and shoots the opponents automatically. Dealing with aimbot is a security problem since the adversary accesses information that is not supposed to be leaked, and also utilizes machine-level reaction that violates the rule of FPS games.

The rest of this report is organized as followed: In section 2, we will briefly go through the common techniques used in aimbots and dig deeper for each of them in section 3, 4, and 5, along with current defenses and our implementation. A new type of defense using machine learning methods will be introduced in section 6. Related works are listed in section 7, and we will discuss and conclude our work in section 8.

## 2 Problem Definition

To tackle the annoying cheat, first we have to know our enemy. Aimbots often consist of two component, extra-sensory perception (ESP) and triggerbot. Since the game process is run on the client's machine, the server needs to pass the information of the position of every player so that the program can render them. ESP steals information from the game process and locates the opponents even when the player should not see them. Triggerbot, on the other hand, use that information to automatically direct the mouse to aim and shoot the target the moment he is revealed.

### 2.1 Threat Model and Assumption

There are various kinds of implementations which exploit different vulnerabilities in the game system. For ESP, it can take advantage of the memory content owned by the game process by matching values over the address space or exploiting debugger tools. It can also alter the game display automatically using DLL injections or by modification of devices or drivers. For Triggerbot, the cheating program can be started within the game process or run separately, making it even harder to detect.

Sadly we are not able to cover all of them in this project. We only be focusing on the vulnerability in the client program. That is, we assume the communication between client and server, and between client process and other lower layer software or hardware, are secure; the server and other soft/hardwares themselves are secure.

For the attackers, we assume (1) they know reverse engineering, (2) they are not able to attack other parts of the game's system, (3) they have total control (root privilege) over the machine that runs the game. All programs are run under Windows 10 OS.

### 2.2 Attack Methods

Three ESP attack methods meet our conditions: memory scanning, debugger exploitation, and DLL injection. As they can steal any information from a program when work in combination, they are popular across different kinds of cheat (and beyond game cheating). Thus the analysis of the attacks and defenses is even more important.

Triggerbot can be easily implemented within our assumptions, as long as it gets the information needed from ESP, so our main focus will be put on how it confuses anti-cheat tools.

Our restriction can be relieved when we introduce the machine-learning based detection, since it only depends on in-game behaviour and dedicates to catching the cheaters instead of eliminating the vulnerabilities to cheat designers.

## 3 Finding Game Variables

### 3.1 Memory Scanning

Windows platforms allow programs to access the virtual memory of another process as long as they are own by

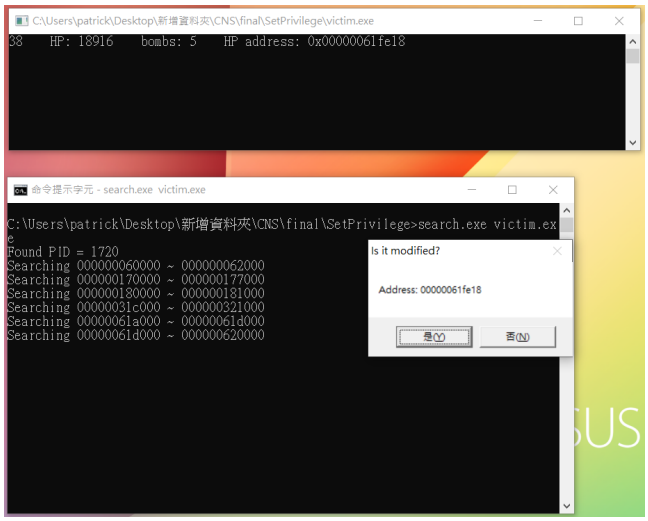


**Figure 1.** The program below searches over the virtual memory of the victim program above. The popped out window asks whether the HP value is modified successfully.

the same user.[12] Despite the mention of **SeDebugPrivilege** on the manual, it turns out that even closing all privileges by the *AdjustTokenPrivileges()* function does not restrict read/write permissions. This is shown in our simple experiment.

Finding the interested game variables remains straightforward by enumerating all possible virtual memory address. One can speed up by examining information of a consecutive memory region. To be more specific, the target variable must lie in a region with state **MEM_COMMIT** and type **MEM_PRIVATE**.[13] By matching value byte-by-byte, one can then discover at which address the target variable is stored. A simple demonstration is shown in figure 1. It tries to modify the stored value at all matched address and let users tell whether it has changed successfully.

Strengths of this approach are simplicity and lack of prevention (due to OS design). However, when the exact value of the variable is not provided to the user, this method becomes unrealistic. Moreover, scanning over the whole memory space is also quite time-consuming; almost every game program adopts dynamic memory allocation, making the found address useless after each restart. This may intimidates naive attackers. Nonetheless, the pointer used to store the dynamically allocated memory region usually possesses a static address. A clever attacker may conduct a two-step attack: the first scan matches the target variable's value, and the second finds which address stores the exact pointer value identical to the found address in the first step.

### 3.2 Debugger and Disassembling

Debugger is the tool for developers to debug a process. This utility is achieved by the ability to pause another process and check the memory content owned by the process. The pauses, as known as "breakpoints", are usually set between two commands, and both read and write operation are allowed during the breakpoints. Common debuggers on Windows platform include WinDbg, Visual Studio Debugger, OllyDbg, etc.

Debuggers are often equipped with disassembling functions. Attackers can set breakpoints right before any game event and identify which assembly codes perform the change of interested variable. Instead of looping over the whole memory space, this approach proves efficient but rather technical for normal users.

This new method actually solves the problem mentioned at the previous subsection: while some player information are explicitly shown as exact values on the screen , most information remain hidden. For instance, the player's HP value is often shown on the screen while position remains unknown. As long as the hidden variables reside in the same class with a variable of known value, attackers can find the base address of the entire class by examining the assembly code accessing the known variable, since its address is represented as the sum of the base address and its offset. After knowing the base address, attackers can then investigate the change of following bytes and hence locate the hidden variable.

Luckily there are ready-to-use defense. Two Microsoft functions *CheckRemoteDebuggerPresent()* and *IsDebuggerPresent()* can examine whether a debugger is attaching the running program. In a game process it is very likely that an attaching process is a malicious cheating software. However, most games do not have these functions in their source code and rely on anti-cheat tools to setup the defense.

### 3.3 DLL Injection

After knowing where all interested variables are stored, attackers can take actions based on instant game states. This can be done in an automatic fashion, using DLL injections.

DLLs are pre-compiled libraries which is intended to be load dynamically by a main program. At first, attackers define cheating logic, such as when and at whom to shoot, in a source code and compiles it into a DLL file. Since Windows platforms assure thread creation from other processes using *CreateRemoteThread()*[10], along with *LoadLibraryA()*[11], attackers can inject the predefined code into any running process owned by himself. The newly created thread is a dedicated background thread and will not become invalid until the victim process has stopped. Hence, it frees attackers from manual check and modification on the important game variables.

DLL injection can be used in aimbot to continuously read out positions and other information, or to perform shooting directly. The technique itself has quite a lot of different application, and it is quite hard to defense. We refer to this page[1] to give a brief introduction on this topic as we are not able to cover all the content.

# 4 Implementation

## 4.1 Our Implementation

In our presentation video[1] we demo our implementation to lock down the health bar, using a memory-scanning mechanism combined with DLL injection. The victim game is an off-line single player FPS game written by ourselves. Codes and game executable can be found in https://github.com/zfrank7777/Aimbot-Survey-and-Demo.git, along with the memory scanning example shown above.

Existing cheat programs, including "Cheat-Engine" introduced in the next subsection, require multiple steps, each after a change, to find one variable's address. In contrast, our approach tries to modify the value stored at each suspicious address actively and let users identify whether the value has changed as expectation, resulting in a one-step-only search. In many gaming situations, users may not observe an immediate game variable change and may not have time to perform variable search after the change. For example, in an FPS game, a player must be shot by an enemy to observe a decrease in HP value, but it is often too late to search out the address during a battle. Note that the verification process does not necessarily require feed backs from a human user. It can be done automatically given the advance of digit recognition.

Despite these advantages, our implementation only works on off-line games with hard-coded write to process memory. In a server-client game system, the server should have all information and only allow client to send legal interaction. The server is able to identify abnormal change in communicated information if we simply modify the game variables locally. Upon detection, the server can either ban the player or just correct the value. However, it is still possible to achieve ESP cheating only using the read-only functions. Since values passed from the server is still buffered in the local process for game display, attackers should be able to identify those addresses although they are restricted to the conventional multiple-step approach.

Another important thing is the reason why we can find the health information with ease: (1) it is a discrete value, (2) it has an clear UI to perceive its change, (3) there are only a few interaction with this information is the source code (it is only checked after the player get hit). We also tried to find the position of the player's character, but the information of the position does not satisfy all the factors above. Moreover, it is not saved in the same class with the health information. Therefore, there is no straightforward approach to find the position information even if we have access to the class base address of the HP value.

[1] https://drive.google.com/file/d/1SoQnh_WSwfBmTwn-1mDhCWgUA3HqrWCs/view?usp=sharing

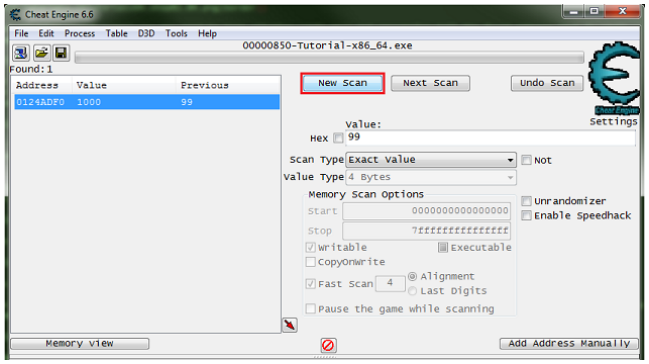


**Figure 2.** Cheat Engine UI. Search query and options are on the right side, and the result is shown on the left.

## 4.2 Cheat Engine

Cheat Engine[2] is a popular game-cheating software that can access game variables. It has a well-organized development with wiki and github pages. Although it does not directly implement ESP, there are examples[4] utilizing Cheat Engine to build an aimbot. From the tutorial of Cheat Engine we learn its capabilities, including:

- Access the memory space of the game process in real time
- Classify the data type of variables in memory
- Search through the memory content with user-friendly options
- Dissemble the code to find code sections that access to certain address
- Modify dissembled code to create unexpected behavior
- Inject new code between operations

Sadly we fail to analyze the full code of Cheat Engine. We only realize that it uses all the techniques we introduce above and beyond (such as driver manipulation and hooking), and it implements different ways to achieve a goal to break through different level of defenses.

With the memory address and values derived from it, ESP, along with many other cheats, is accomplished. Also, if one can elaborately use the code injection or modification, we think it is possible to implement a complete aimbot within Cheat Engine.

Figure 2 shows the user interface of Cheat Engine. The address searching function is very user-friendly, as there are many different ways to search through the memory, such as deciding data type, filtering (un)changed values, searching values in certain range, and so on. As the vulnerabilities are not likely to be resolved, it is expected that Cheat Engine will stay prevalent on unprotected games with such popularity and easy usage.

# 5 Triggerbot

Triggerbot is a program that triggers the shot automatically. With triggerbot enabled, the user's shooting accuracy increases significantly, gaining huge advantages during battle. The following describes the implementation of triggerbot[14], and some strategies of both attacker and anti-cheat developer.

## 5.1 Triggerbot Implementation

To implement the triggerbot, first we need to know the address of the game states such as teamID, health, and crosshair. Triggerbot relies on the metadata mentioned above to develop functions, e.g. recognizing live or dead body. The address can be found by trial-and-error attempt to locate certain variable using cheat engines (e.g. Artmoney[18]). Some people even spread the know variable address among the Internet.

Next, we have to assure that memory read/write from game application is successful. To retrieve modify the memory of the target application, one can write a program that is similar to file I/O. For instance, Microsoft provides ReadProcessMemory Windows API[9] for programmers to inspect the process memory. For Linux the memory maps are stored in /proc/$pid/mem. Libraries such as ProcMem[5] also exist to reduce hackers' burden reading/writing memory.

When the metadata are ready, the remaining works are to design desired functions. Hackers can customize their shooting pattern or enhance shooting accuracy. Advanced feature like anti-recoil is useful to mitigate the reaction force of the shot.

## 5.2 Attack and Defense Strategies

In this subsection, we are going to talk about the possible strategies for attacker to avoid anti-cheat detection, as well as the common methods for anti-cheat system to find cheaters.

**5.2.1 Attacker.** Since executing the triggerbot requires memory reading/writing, attacker could be caught cheating due to disallowed memory modification. One way to reduce the risk is avoid intensive or abnormal memory reading/writing behavior, especially writing. For instance attacker can change the camera angle by modifying the value of the corresponding variable. The anti-cheat system can detect this abrupt shift easily and thus ban the player. In general, memory writing should be as less as possible.

Another way to mitigate the chance of being detected is to make triggerbot more human. What "more human" means is to adjust the bot's behavior so that anti-cheat service can't tell the difference between a real player and a bot. Overly high accuracy or short reaction that surpasses professional players are suspicious and easily be reported by other players. In contrast, if the performance of the triggerbot is better than average player and under the level of the professionals, it would attract less focus from peer players and anti-cheat system. Although attacker sacrifices triggerbot performance, the benefit of cheating still make the cheater more competitive.

**5.2.2 Anti-cheat Developer.** There are many methods to mitigate cheating behavior. Here we introduce to common measures, memory integrity and behavioral analysis.

Memory Integrity aims to check whether the value in an user's process memory is tampered or not. Game Developer can design a workflow such that client process send the data back to server, and let server simulate the replay. If some variables in simulated result changes unreasonably, it may imply the player is dishonest.

Behavioral analysis is a powerful method to detect cheating behavior. This method detects cheats by analyzing the performance of each player in the game. For example, the action such as pushing the button at non-human speed, and extremely short reaction from enemy showing in the screen to triggering the shot, are things that human can't achieve. Recently some researchers incorporate deep learning into behavioral analysis and obtained competitive performance. Cheating events should be harder to avoid anti-cheat detection.

# 6 Behavior-based Defense

Currently, the most successful defense approaches relies heavily on the players' collaboration. In order to block and detect known hacks, game sellers have to release patches constantly while the cheat methods evolves over time. Moreover, the anti-cheating software themselves are also vulnerable to hacks.

Hacks can be performed and implemented in different ways, but these form of hacks result in similar game playing patterns. We can find a solution to automatically detect whether a player is cheating or not.

In this section, we will discuss 3 types of behavior-based defense and its application in current online-game cheating detection.

## 6.1 Bayesian Approach

**6.1.1 Bayesian Network.** Bayesian network is an acyclic graph that represents a set of variables and conditional dependencies in a in a probabilistic manner. Each node in the Bayesian network represent a variable, and each edge represent the dependency from the parent node to its children node.

[17] showed that Bayesian network can help us to estimate the probability of the player being a potential cheater. Let us consider a simple example, a Bayesian network models the causal relation between three variables: *Cheating*($\tilde{C}$), *Distance*($\tilde{D}$) and aiming *Accuracy*($\tilde{A}$). Here the variable *Cheating* and *Distance* are the parent node of *Accuracy*. Therefore, the probability of the player is cheating or not and the distance

from the aiming target can affect the result of aiming accuracy. The value of *Accuracy* has a dependency upon *Cheating* and *Distance*, while *Cheating* and *Distance* are independent of any random variable. The joint probability distribution of this Bayesian network is given by:

$$
\begin{aligned}
&P(\widetilde{C},\ \widetilde{D},\ \widetilde{A}) \\
&= P(\widetilde{A}|parents(\widetilde{A})) \cdot P(\widetilde{C}|parents(\widetilde{C})) \cdot P(\widetilde{D}|parents(\widetilde{D})) \\
&= P(\widetilde{A}|\widetilde{C},\ \widetilde{D}) \cdot P(\widetilde{C}) \cdot P(\widetilde{D})
\end{aligned}
$$

**6.1.2 Dynamic Bayesian Network.** The Bayesian Network approach can help us inferring the probability of a player using cheats or not, however, this approach can only model the static behavior of a player. Since aiming is a fine-tuning process, the accuracy can mostly based on the one of last moment. The outcome of random variables can be time-dependent. Therefore, we need a temporal model to perform cheat detection.

Dynamic Bayesian Network(DBN) is the temporal model that uses Bayesian Network relating variables to each other over adjacent time slices. This enhancement adds a dependency between variables *Cheating*($\tilde{C}$) in each two adjacent time steps. The value of *Cheating*($\tilde{C}_t$) at time $t$ can be calculated from the variable $\tilde{C}_{t-1}$ at time $(t-1)$.

In a DBN, the aiming accuracy can depends on whether the player is cheating, whether the player or the target is moving, the aiming direction, the distance between the player and the target, and other random variables.

### 6.2 Trajectory Analysis

In [15], the authors compared the avatar trajectory of a human player and multiple types of cheat bots. They found that human players are tend to explore all areas on the map and avoid open spaces, in order to reduce the chances of being attacked. Even though they spend most of their time in narrow areas and confined spaces, there were still large variations in their trajectory.

By contrast, bot trajectory patterns are more regular and predictable. This observation can be used as the basis for a preliminary discriminant analysis.

We can compute the entropy by

$$H_{si}(x) = -P(x)\ log\ P(x) - (1 - P(x))\ log(1 - P(x))$$

if the binary random variable $x$ represents the event that a trace touches a location $i$. The entropy value of human traces should be higher than those of bots.

### 6.3 Skill-performance Inconsistency

Aimbots usually interact with humans intensively, making traditional bot detectors ineffective. [7] proposed a system uses a cascaded classifier that detects the inconsistency between performance and skillfulness. By extracting features precisely, they are able to filter out high level players with bad skills, which have a great chance of using aimbots.

## 7 Related Work

There are tons of game cheating in the world. [20] and [21] showed a rough classification to give an image of different exploitation. [16] updated the information and included the defenses of different hacks. All of them only did general analysis without presenting any real-world example. On the other hand, [3] delivered questionnaires and focus on the most popular cheating and counter-measures.

As the defending strategy is more of catching the cheaters than the cheat developers, the method used in the detection is often not shown in public. Thus there are only a few works introducing a small proportion of all kinds, such as [8] surveyed cheat in MMORPGs and [19] use bridge (the poker game) as an example of game with incomplete information.

On the contrary, researchers are sharing their methods of the behavior-based detection, as they are more difficult to break without clear rules. [6, 22] along with other works introduced above are of this type.

## 8 Conclusion and Future Work

In this work, we divided the well-known game cheat "aimbot" into two important components, ESP and triggerbot, and went through their implementation in depth.

ESP tells the attacker all hidden information. We demonstrated how to find where the interested game variables are stored, either by the "memory scanning" method or the "debugger" method. We also succeeded in injecting a cheating plug-in on our real FPS game. We concluded that on Windows10 platforms, there are no effective way to prohibit memory read/write from other processes. Therefore, instead of detecting whether a remote debugger is attached, it may be more effective to check memory integrity — an additional hidden variable dependant on the protected variable is added; their consistency indicates whether the latter is modified irregularly.

The implementation of triggerbot is rather simple; as long as it has all information from the ESP component, all actions could be hard-coded into a rule-based manner. The most important part of this component is to circumvent cheating detection; triggerbots must resemble human players to a certain extent.

Since the rapid evolution of cheating bots creates deployment difficulty of anti-cheat patches, game sellers have to develop a scalable and efficient method to find out whether a player is cheating or not. The biggest difficulty of this method is to filter and modelize players' gameplay data into a measurable form.

Our ESP program is able to perform variable search in only one-step while similar cheating programs all require variable change for filtering. However, it still needs feed backs from users to know whether the found address is correct. Regarding the advance of digit recognition tools, an extension for near future is to let users specify where the variable

is shown on the screen; the ESP program then recognizes whether the shown value has changed as expected. It can then work automatically.

## References


[1] Ryan Becwar Anastasios Pingios, Christiaan Beek. 2017. Process Injection. https://attack.mitre.org/techniques/T1055/

[2] Dark Byte. [n.d.]. Cheat Engine. Retrieved November 1, 2020 from https://www.cheatengine.org

[3] Hangbae Chang, Jong Hyuk Park, and Hongsuk Kang. 2008. The Security System Design in Online Game for u Entertainment. In *22nd International Conference on Advanced Information Networking and Applications-Workshops (aina workshops 2008)*. IEEE, 1529–1533.

[4] Guided Hacking. 2012. C# How to make an AIMBOT tutorial. https://www.youtube.com/watch?v=NUeifQK7ukM

[5] HikaN. [n.d.]. *csgo-simple-cheat*. https://github.com/HikaN/csgo-simple-cheat/blob/master/ProcMem.cpp

[6] Peter Laurens, Richard F Paige, Phillip J Brooke, and Howard Chivers. 2007. A novel approach to the detection of cheating in multiplayer online games. In *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*. IEEE, 97–106.

[7] Daiping Liu, Xing Gao, Mingwei Zhang, Haining Wang, and Angelos Stavrou. 2017. Detecting Passive Cheats in Online Games via Performance-Skillfulness Inconsistency. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 615–626. https://doi.org/10.1109/DSN.2017.20

[8] Gary McGraw and Greg Hoglund. 2007. Online games and security. *IEEE Security & Privacy* 5, 5 (2007), 76–79.

[9] Microsoft. [n.d.]. *ReadProcessMemory function*. https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-readprocessmemory

[10] Microsoft. 2018. CreateRemoteThread function (processthreadsapi.h). *Microsoft Docs* (2018). https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createremotethread

[11] Microsoft. 2018. LoadLibraryA function (libloaderapi.h). *Microsoft Docs* (2018). https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-loadlibrarya

[12] Microsoft. 2018. OpenProcess function (processthreadsapi.h). *Microsoft Docs* (2018). https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-openprocess

[13] Microsoft. 2018. VirtualQueryEx function (memoryapi.h). *Microsoft Docs* (2018). https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualqueryex

[14] Motherflufferr. 2014. Making hacks with C++. https://www.unknowncheats.me/forum/counterstrike-global-offensive/129168-making-hacks-c.html

[15] Hsing-Kuo Pao, Kuan-Ta Chen, and Hong-Chung Chang. 2010. Game Bot Detection via Avatar Trajectory Analysis. *IEEE Transactions on Computational Intelligence and AI in Games* 2, 3 (2010), 162–175. https://doi.org/10.1109/TCIAIG.2010.2072506

[16] Reza M Parizi, Ali Dehghantanha, Kim-Kwang Raymond Choo, Mohammad Hammoudeh, and Gregory Epiphaniou. 2019. Security in online games: Current implementations and challenges. In *Handbook of Big Data and IoT Security*. Springer, 367–384.

[17] John C. S. Lui S. F. Yeung. 2008. Dynamic Bayesian approach for detecting cheats in multi-player online games. In *Multimedia Systems*. IEEE, 211–236. https://doi.org/10.1007/s00530-008-0113-5

[18] System SoftLab. [n.d.]. *Artmoney: a Memory Editor*. https://www.artmoney.ru

[19] Jeff Yan. 2003. Security design in online games. In *19th Annual Computer Security Applications Conference, 2003. Proceedings*. IEEE, 286–295.

[20] Jeff Yan and Brian Randell. 2005. A systematic classification of cheating in online games. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. 1–9.

[21] Jeff Yan and Brian Randell. 2009. An investigation of cheating in online games. *IEEE Security & Privacy* 7, 3 (2009), 37–44.

[22] Siu Fung Yeung, John CS Lui, Jiangchuan Liu, and Jeff Yan. 2006. Detecting cheaters for multiplayer games: theory, design and implementation [1]. In *CCNC 2006. 2006 3rd IEEE Consumer Communications and Networking Conference, 2006.*, Vol. 2. Citeseer, 1178–1182.