A Windows-based Software Architecture for Protecting Online Games against Hackers[1]

Luan Bui The, Khanh-Van Nguyen
VietNam Multimedia Comporation, 65 Lac Trungst, HaNoi, VietNam
Hanoi University of Technology, 1 Dai Co Viet st, Hanoi, Vietnam
vannk@soict.hust.edu.vn, luan.bui@vtc.vn

***Abstract* - We study the problem of protecting Windows-based online games against hackers. We consider an interaction scenario between 3 parties, the game producer, the game distributor and the game player (client), where we aim to support the distributor to fight against cheater clients with minimum cooperation from the game producer. For a protection solution, we propose a software architecture which introduces the specifically designed module*GameGuard*. Upon being played in the Client machine, the GameClient program simultaneously interacts with *GameGuard* so that if a serious hack is attempted, it will be detected and then both modules will terminate.**

We support the game distributors with a general protection framework which is not dependent on specific games and also only needs minimum cooperation from the distant game producers. Our approach is powerful enough that we would defeat most kinds of attacks which are based on hooking Windows APIs, including ones from memory, graphic or hardware attack categories.

Keywords- Online game, Hacking, Hooking, Windows APIs, Kernel-mode

I. INTRODUCTION

Today Game Online is rapidly becoming a hot entertainment industry, and thus an important source of studies. This form of online service allows a collection of clients who may sit separately at their home but effectively join in a virtual world, where they can perform "wonders", according to imaginary scripts, without suffering the physical hardness as in the real world.

Typically, a game system consists of game server(s) and multiple copies of game client, which interact together following the game scenarios and rules. The game servers collect playing behaviors from game clients, process this data using the game rules then update the game clients with new game states. Especially, the recent presence of Massively Multiplayer Online Games (MMOGs), where multiple thousands or even millions people (as in the case of game "World of Warcraft") take their parts in a great fictional event, produces certain social effects and brings in a lot of revenue to the game manufacturers and service providers.

This also draws a lot of attention and hence, cheating acts from hackers who want to play for free of even steal money from accounts of innocent gamers. These hackers share their cheating techniques with a vast number of users, which causes big losses to the providers. Thus, research in this field of computing is becoming hot, which focuses not only on improving the game qualities but also on fighting against cheating in online games.

Cheating in online game can be in various forms where the cheaters can develops simple to sophisticated techniques. Typically, the hackers use their usually expert knowledge to interfere with the game logics and take advantage over other innocent gamers, thus violate the rules and fairness of the game. A typical example is hacking into the memory segment which stores the data of the game client to twist the game status for the cheater's benefit. Game manufacturers have tried hard to prevent against cheating but their efforts have not gained sufficient success in defeating cheating attempts.

The cheating game problem appears even harder for developing countries such as Vietnam where the domestic game industry is still in an infant stage. Due to the enormous demanding of the market, typically, most producers/providers of MMOGs are foreign while the domestic companies only act as distributors to serve the domestic clients with game client software and connection to the game server abroad.

[1]A preliminary version of this paper appeared in a conference proceeding [16].

This sets up a typical, undesired situation of running game service in Vietnam: the foreign producers can not actively investigate the domestic environment for finding and fighting against local cheating while the domestic distributor companies who can play a more active role in this fighting cannot directly create solutions by modifying the game client software. That is, the domestic distributors although are more familiar with cheating forms as well as hacking tools developed by domestic hackers but cannot do much to change the game client software which is developed by the foreign producers.

In this paper, we look deeply into this situation and develop a general solution to support the domestic distributors in protecting against cheating in online game. We create a mechanism framework which adds new software elements to the typical software architecture that is usually implemented amongst the foreign producer/provider (**P**), the domestic distributor (**D**) and the domestic gamer (**G**). These additional software modules are to be developed by **D** with a little cooperation from **P**: **P** only has to add to the code of the game client a few specified API calls that will activate a DLL (dynamic link library) module which is produced by **D**.

More specifically, in our proposed architecture a module named GameGuard (**GG**), developed by **D**, is added to the client site to guard against possible cheating attempts from game players using techniques to be mentioned later. Module **GG** is supposed to run simultaneously with the normal game client (**GC**), developed by **P**, and secretly communicates with **GC** to help protect against malicious hackers. Thus, the main factor of our proposal is the introduction of module **GG** and the secret interaction channel between **GG** and **GC**.

Based on the introduction of **GG** and the **GG-GC** interaction, we create a protection system with two layers of protection. Once activated, **GG** will try to hide all the game-related modules so that normal hacking tools cannot see them, using certain system techniques to interfere with some operations of Windows systems (e.g. to hook into certain system calls). This is the first layer of protection.

As for the second protection layer, **GG** also places hooks at a deeper level, i.e. Windows kernel mode, to examine new processes entering and check if they are hacking attempts (using a blacklist mechanism). Whenever it senses the active presence of a malicious hacking process **GG** can choose to send alarm to **GC** and then both will terminate immediately. Thus, even a malicious hacker which overcomes the first protection layer by digging into Windows kernel mode can still be defeated by this second layer of protection.

For convenience, we name our proposed protection system by *GameGuard*. Our *GameGuard* system also provides *soundness* against hacking directly attempted at **GG** or the **GG-GC** interaction mechanism. Our only security assumption is on the integrity of the **GC** code, which is supposed to be highly guaranteed by the game producers, who should have invested enough in off-the-shell technologies to provide authenticity of **GC** (see e.g. [2,4,7]). That is, the integrity of the modules added by **D** is securely based on the integrity of **GC**: if **GC** can be trusted then the **GG** and other elements added by **D** can also be trusted. Note that our aim is to support the role of **D**, so other working aspects of the general game model such as the mentioned technique of making **GC** authentic is beyond the scope of this framework.

The paper is organized as follows. In section 2, we present our model of attacks and discuss related literature and existing real-world products. Section 3 presents the core of our system, introducing module **GG**, its main functions and the interaction with **GC** to protect against hacks. We discuss our implementation of the proposed framework and initial experiments at VTC in section 4. Finally, section 5 evaluates our framework and initial product and gives our concluding remarks.

II. RELATED WORK

We propose to study the problem of protecting online games against hackers in Windows environments. This problem belongs to the area of research in protecting against general cheating in online games which has recently been an active area with a number of extensive reviews [6,9,10,12]. There are many different forms of cheating in online games and there is no unique way to classify them. In [11,12], the authors presented a detailed list of 11 forms of cheating, using various different cheat scenarios. Although descriptive, this classification is considered unstructured and uneasy to apply [10]. A more structured classification is proposed in [3] and supported in [10] which has inspired our attack model above. This divided online game cheats into 4 classes depending on where the cheat targets at: (i) game-level cheats: targeting at errors on game logics or

implementation; (ii) application-level cheats: either modifying the game modules or data files, interfering with the memory of the running game; (iii) protocol-level cheat: interfering with the packets sent and received by the game (which may be inserted, destroyed, duplicated, or modified by an attacker); and (iv) infrastructure-level cheats: modifying or interfering with the software (e.g. graphic drivers) or hardware (*e.g.*, the network infrastructure) that the game is using. It is easy to see that the infrastructure cheat category contains both our *graphic and hardware attacks*, protocol cheat matches our *server attack*, application cheat contains our *memory attack*, while game-level cheat (specific to game logics) is not in the scope of our attack model.

Recently, a special game-level cheat using *automated bots*, has been considered as very dangerous attack that could be used to degrade and demolish MMOGs. Bots are programs that are designed to imitate game players to take part in MMOGs, but in a harmful way. Attackers can instruct these automated bots to do very specific tasks, harmful to honest players, e.g. by eating away resources, which would make disadvantages to all other legitimate players. They take away the fun and competitiveness of the games and hence, legitimate players may feel bad and leave the game, causing financial loss to game producers. Some promising approaches have been proposed to detect this dangerous attack. In [1], Bethea et al. proposed server-side verification of client behavior, where common structures on the client side can be exploited, through ways of using constraints on client-side states, to validate messages between clients and the server. In [14], user behaviors are statistically analyzed using game activity logs such that established threshold levels for these user activities help to identify game bots.

Our paper assumes the authenticity of (or assurance of the integrity of) the game client code, which the game producers should have applied necessary technologies, e.g. [2,4,7], to master.

In the real-world, there exist several known products which support protection against online game hackers: Nprotect, X-Trap, DMW Anticheat, GameGuard, PunkBuster, VAC, ShoxGuard, XRay or Warden [16]. Nprotect and X-Trap are the most popular products, either of which most popular online games have been integrated with. Unfortunately, they are not well-documented and we have found it hard to find a rich source of information about them. We compare our solution with Nprotect and X-trap in Section V.

Scalability is the most important aspect in designing MMOGs, which has a strong relation to security issues. MMOG-based virtual worlds can create incredibly enormous attraction e.g., World of Warcraft (WoW) has drawn 8 millions gamers in 2006 [5]. Most MMOGs still use the traditional Client/Server where clients are connected to a cluster of central servers. This architecture brings in important advantages in centralized management and security control to the producers/providers, but does not provide scalability to support large numbers of players. The common way to temporarily resolve this to use sharding [13], where players must be divided into multiple separate communities – shards (one must play within his/her own shard). This certainly goes against the socializing trend of online gamers that causes certain disappointment.

In recent research in MMOGs, peer-to-peer (P2P) architecture has been extensively studied to promote scalability, low cost and fast response time [13]. This architecture would distribute the game control and processing and communication to a vast number of independent nodes (peers) which can be the player's PC. Distributing a game among peers helps to solve bottleneck problems of network bandwidth or central processing, however, makes maintaining control over the game more complex. Peer-to-peer architectures also tend to be vulnerable to churn and cheating [13].

III. OUR ATTACK MODEL ON WINDOWS-BASED ONLINE GAMES

We present our general view of typical classes of hacks in a typical model of Windows-based game deployment.

The typical architecture of game software components deployed at the client machine is depicted in Fig. 1. The elements which involves in the game playing process can be placed in 4 layers as follows. The game (console) application lies on the top layer which use services of the game engine in the below layer. The third layer consists of the DLLs (Dynamic-link libraries) with API calls and respective drivers to support those API calls, including the graphic driver (i.e. DirectX in Windows systems). These DLLs and drivers provide services for controlling hardware (bottom layer) to the game application (or Game Client) as well as the game engine. This deployment

architecture reveals 4 general categories of possible game hacks as we mention below.

Memory attack. Here, the hacker first tries to find the memory area which stores the game states and date. Once having found it, the hacker would twist certain game state variables or info to his advantage and hence, cheating on the other innocent gamers and violating the fairness of the game. For example, the hacker would increase the values on the strength or the command degree of his game character, or would enhance the power of his weapon in a game of battle.

Graphic (DirectX) attack. Here, the hacker hooks the graphic API so he can interfere with the graphic system. Basically, his hooked API calls would call his own programmed graphic codes instead of DirectX ones. He can use these unauthorized codes to obtain unfair advantages in his playing by twisting certain features of graphics objects in the game. For example, he can make a solid wall become transparent so he can see through it.

Server attack. Hacker targets at possible security vulnerabilities in the server or the communication link between client and server so that he can modify the game status to his own advantage. Usually, the hacker would modify the game states and data that the client sends to the server.

Hardware attack. Hackers attack the hardware drivers so that he would earn unfair advantage over innocent gamers in the physical aspects, e.g. playing with higher speeds of keyboard or mouse, or with higher rate of CPU processing. Similar to graphic attack, these hacks would typically hook the DLLs which support the API calls that control hardware actions.

Our approach does not aim at separate kinds of attack but rather, we look at the root of most problems: there is a common component in the Windows system that most attacks are able to interfere with. That is the DLL modules that support several APIs where the most common technique and hack mechanism that the hackers rely on by hooking these APIs. Thus, we aim at preventing and detecting hooking attempts from the game hackers. Our approach is general and flexible enough that we would defeat most kinds of attacks which are based on hooking Windows APIs, including ones from memory, graphic or hardware attack categories. Also, we aim to support the game distributors with a general, flexible protection framework which is not dependent of specific games and that distributors only need minimum cooperation from the game producers.

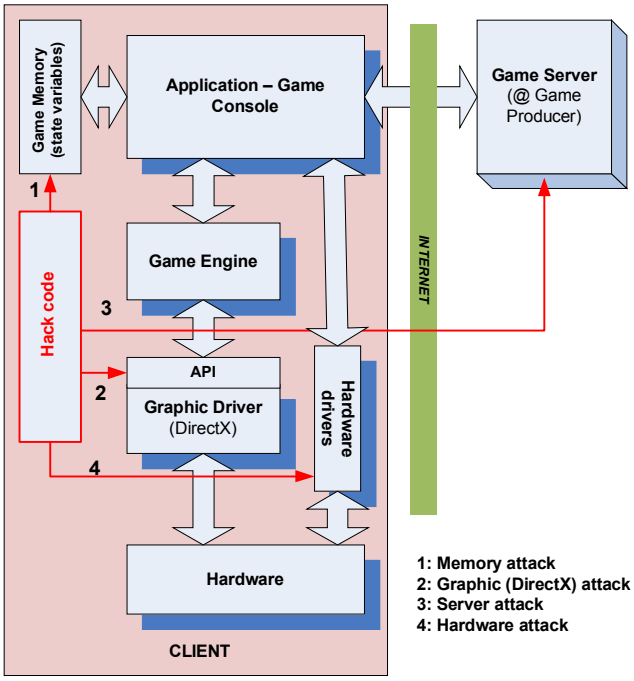

Figure 1. Game deployment architecture and possible attacks

(The read arrowed lines illustrate the 4 attack scenarios)

IV. OUR PROPOSED SOLUTION FOR GAME DISTRIBUTORS

We propose software architecture and technical framework to be implemented by the domestic distributor **D** (with cooperation from the foreign producer **P**). At the center of the proposed protection system is module GameGuard, or **GG** in short (thus we call ours the GameGuard system). Bellow we will first describe **GG** and other elements of our proposed architecture then present our technical framework in implementing these software modules to protect against cheating.

A. Introducing GG and GC-GG interaction mechanism

We describe our proposed architecture (depicted in Fig. 2). Module **GG** is the cornerstone of our proposed system. **GG** is an independent process which runs simultaneously with **GC** (GameClient) and securely synchronizes and communicates with **GC** to fight against cheating. Typically, if **GG** finds out that a blacklisted program is being installed into the client

computer, it will alarm **GC** and both will terminate immediately.

It is important for our framework to be flexible enough: it should be applicable to most Windows-based game clients using only minimum cooperation from their respective **P**. In order to achieve this flexibility, we introduce another new element to the software architecture at the client side, **GC-agent**, which will represent **GC** to communicate with **GG**. In fact, **GC-agent** is a DLL (dynamic-link library) component of **GC**, which is activated by **GC** upon **GC**'s initialization. The interface between **GC** and **GC-agent** is just a few API calls; hence, all **D** has to do is to ask the respective **P** to implement such a simple interface into **GC** and let **GC** to activate **GC-agent** right on **GC**'s start-up.

On the other hand, our framework lets both **GG** and **GC-agent** to be implemented by **D** so it is straightforward to establish a secure communication channel between these two modules, using standard process intercommunication such as pipelining: we choose to use pipelining equipped with encryption/decryption operations in our implementation in Section IV.

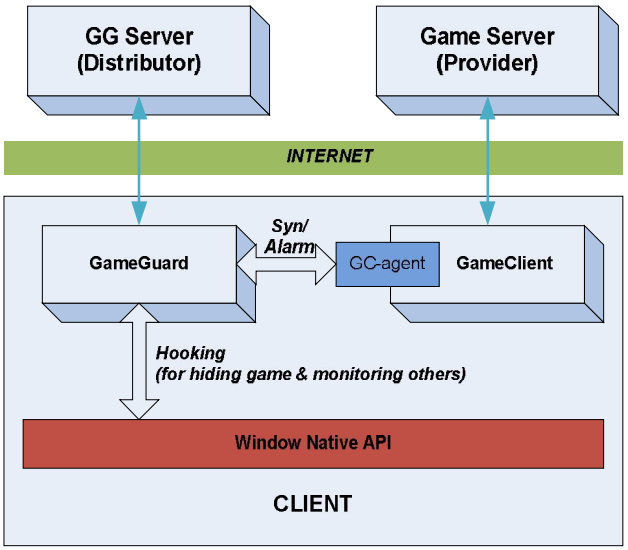

Figure 2. Proposed software architecture for protecting against cheating

Through **GC-agent**, **GG** and **GC** can frequently synchronize with each other so each can learn about the presence or absence of the other immediately. This helps to set up an *almost-invariant logic* in our system: ***GG** and **GC** will always (almost) run or terminate together*. Even if a process of the two dies suddenly for some certain reason the other will terminate just after when sensing lost signal from the former.

B. Basic functionality of GG

As mentioned in Section I, we deploy 2 layers of protection where one is for hiding game elements (processes and data) from the view of other processes and another one is for detecting the presence of hacking attempts (then quitting before being attacked). So, the first one is more passive while the second is more active. This active detection layer is deployed in 2 phases:

+ Firstly, *Full Scan*: Initially when **GC** and **GG** start up, **GG** scan all the running processes, existing system services and certain DLL modules (with potential of being hooked) to collect their *signatures* (consists of ID values and hashed codes, i.e. checksums). It then compares these signatures with the blacklisted ones and upon certain pre-alarm for certain hack signature found in the blacklist, **GG** can terminate the game.

+ Secondly, *Monitor*: After this initial full scan, **GG** creates a software trigger (by hooking to some Windows low-level API) that will have **GG** to be aware of and scan each new process or service coming in the system. Thus, this is only a *scan* for just the new, coming in executives.

In Windows environments, to achieve these functionalities one can use certain hooking techniques which interfere with certain system operations such as listing the running processes or the services which are registered in the Windows Registry. In Windows OS family, the *Native API* is the publicly incompletely documented API which is lowest level API, used internally by the OS. Most of the Native API calls are implemented in kernel-mode module *ntoskrnl.exe* and are exposed to user mode by *ntdll.dll* – the shield of kernel model (however, some Native API calls are implemented in user mode directly within this *ntdll.dll*).

To achieve the hiding purpose, our framework proposes to hook into this Native API by injecting new code to *ntdll.dll*. The higher-level API calls which would ask these hooked Native API codes for information about running processes will obtain nothing about **GC**, **GG** and other game modules.

To achieve the detection purpose, we propose to also hook into Native API so that **GG** can monitor the existing processes of services and even the new

processes or services coming in. For example, we hook API call *NtCreateProcess* so that when a new process is initiated, the system will call on *NtCreateProcess*, so **GG** can monitor this event of new coming party. Even if hackers have attacked and hooked in the Native API to try hiding themselves beforehand, we can still detect this happening by investigating the respective DLLs using checksum method. Note that, when digging into such low level of Windows systems, no code can completely hide its presence. However, **GG** can always decide to terminate the game to avoid being further attacked. We will mention more about these hooking techniques in the implementation section.

Blacklisting new hack code, general or specifically targeting at certain games, is usually performed by human experts who can be employed by **D**. The blacklisted signatures are sent to **GG** by an auto-update mechanism. We propose to have a specific server at **D** site to maintain the updated blacklist. In our framework, upon its start-up **GG** opens a secure connection with this blacklist server to ask for updates periodically.

C. System soundness: authenticity of the added modules

Our **GG**-based protection system which relies on an authentic **GG-GC** communication can be defeated by experienced hackers if we allow them to successfully create a fake **GG** (i.e. without being detected by **GC**): that fake **GG** can nullify our detection-and-alarming mechanism. Thus, we must deploy a mechanism to assure the integrity of **D**'s added modules, i.e. **GC-agent** and **GG**. Remind that authenticity of **GC** is pre-assumed in this paper; technologies that **P** should apply to assure the integrity of its product is beyond the scope of this paper. Our framework achieves the following assurance for our proposed GameGuard system.

Authenticity Guarantee from Distributor: *Assuming that **GC** code is authentic, the added modules **GC-agent** and **GG** (and its follower, the hack-blacklist) have their integrity to be aware and guarded by the system: attempts to modify or fake them are to be defeated with a very strong possibility.*

In order to make that claim, we propose to check the integrity of the added modules right before activating them. To do that, we line them up to create a sequence of activation (starting with **GC**): each module A will validate then if OK activate B, the next one in line. On validating B's integrity, specifically, A will compute the checksum of B's code stored in the physical storage (using a secure cryptographic hash function) and compare with the true value pre-computed and injected to A's code by **D** (with cooperation of **P**in case of A being **GC**). Particularly, **GC** will validate-then-activate **GC-agent**, which will do the same with **GG**(which may do the same with other modules in a more refined model). If any step of this validation sequence falls, **GC** will be alarmed and terminate immediately.

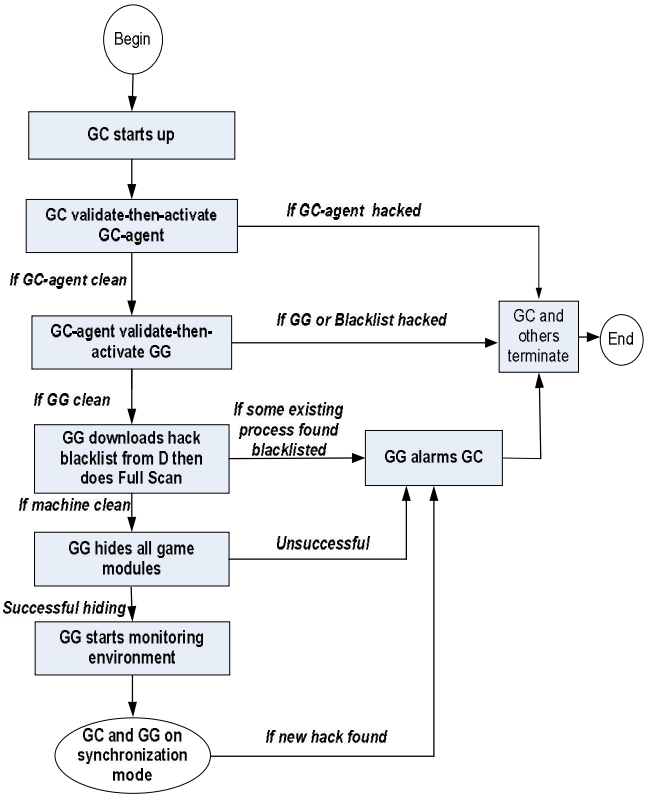

Figure 3. System working flowchart: GC, GC-agent and GG cooperate to fight against cheating

Once true **GG-agent** and **GG** are executed, our GameGuard system can actively monitor the system environment and hence, can foresee possible harms that may happen and can avoid being hacked (by terminating the game and related modules).

D. Protecting against hack codes: putting the actions together

Below we will consider our proposed GameGuard system in full action. We depict all the major, previously mentioned actions in the flowchart diagram in Fig. 3. Note that we omit the full details in **GG**'s

communicating with **D** to download/update the hack blacklist. Certainly, **GG** has to assure the authenticity of the **D**'s server and hence, download the true blacklist.

To illustrate how the system functions, consider a typical cheating example when a cheater tries to twist the graphic aspects of an online game, e.g. to make a solid wall become see-through. Typically, the hacker would hack certain DLL modules which provides API calls to Windows graphic driver, i.e. the DirectX driver. By a proper configuration, we can have our system pre-compute true checksums so that **GG** will invalidate those hacked DLL modules during the Full Scan phase, which leads to the game's termination before the hacker can actually cheat on.

V. IMPLEMENTATION AND EXPERIMENTS

We have implemented our framework to construct a GameGuard system for being used at VTC, the Vietnam Multimedia Corporation. The software product is being under experimentation where initial evaluation is very encouraging so that the product will soon be used by VTC and its large base of secondary distributors. Below we describe the system structure (depicted in Fig. 4) and explain some technical details.

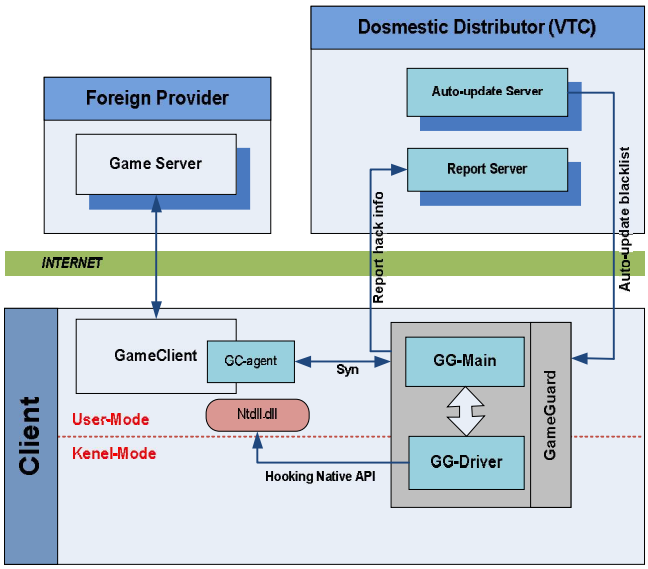

Figure 4. Product implementation at VTC: network model and the system structure at client

As seen in Fig. 4, the product's GameGuard has two major components: **GG-Main** and **GG-Driver**. **GG-Main** performs the main functions of **GG** which are: (i) communicating with **GC-agent**; (ii) hiding game modules and detecting hacks; and (iii) communicating with distributor servers. In **GG**'s communication with **GC-agent**, we use the standard pipelining and the AES encryption scheme to create a secure connection between these two modules. Synchronization between them is accomplished by exchanging specific SYN packets every fixed short period of time.

To perform hiding game codes and info and detecting hacker presence, we interfere with the Windows kernel by implementing specific hooks at the Windows Native API. Here we have to implement a separate component of **GG** in a form of Windows driver to perform these kernel-mode interferences. Executing Windows drivers are allowed to be at Windows kernel-mode, hence, our **GG-Driver** can hook necessary Native API calls (which are coded in the *ntdll.dll*). For hiding our game processes and other related information, we hook the Native API calls which return the process info (*ZwQuerySystemInformation*with parameter *SystemInformationClass*as*SystemProcesses and-ThreadsInformation*) and the calls which return the registry info (*ZwEnumerateKey* with parameter *KeyInformationClass* as *KeyBasicInformation*). For detecting new hack attempts, we hook the necessary Native API calls that help monitoring the OS environment, including *NtCreateFile*, *NtCreateKey*, *NtCreate-Process*. Note that *NtCreateProcess* is called by the OS to create a new process, *NtCreateKey* to create a new key for registering a new driver with the Windows Registry, and *NtCreateFile* to create a file in a hard disk.

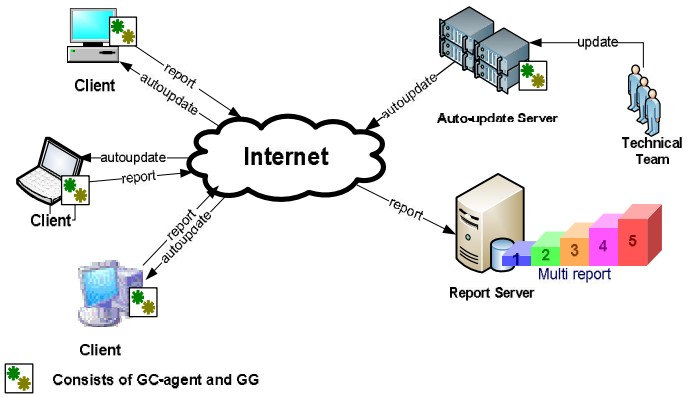

Figure 5. Information flow between GG and the distributor's servers

The communication of **GG** with **D** (i.e. with certain servers of **D**) consists of reporting hack-related

information to Report server and periodically checking and updating with auto-update server. The reports to distributor may include the signatures and samples of processes that are doubtful of hacking, which are definitely useful for the technical team at **D** in studying new hacks. The auto-update mechanism is not only for getting the updated blacklist (from **D**) but also for getting the new **GG** version if exists (with necessary validation and proper installation into the client). We illustrate this two-fold accomplishment in Fig. 5.

Extension for multiple GCs. In practice, there exists a quite popular trend that many game players want to play several different games simultaneously in a single client machine or they want to play in several accounts of a single game to rapidly increase the player's level of strength and command in this game. We extend the framework so that a single **GG** module can support multiple **GC**s (Fig. 6). Inside **GG**, we deploy a separate running thread **GG-Listener** to communicate with the **GC-agents**. The **GC** which is firstly executed will launch **GG** while the later coming **GC**(s) only need to open new connection to **GG**. If **GG** detects a serious hack attempt it will terminate immediately that leads to termination of all **GC**s as well.

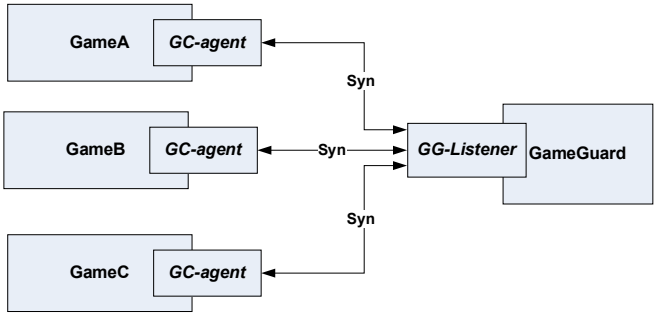

Figure 6. GameGuard serves multiple Game Scheme of a Single Clients.

We have experimented our GameGuard system with several popular games in Vietnam such as CF, FIFA ONLINE2, AU. We run our system in Windows XP sp 2 and Windows 7 ultimate (32bit) under the presence of several popular hack tools. We summarize our initial experiments in Table 1: the success of these experiments shows that our GameGuard system would be successful and accepted in the Game industry, at least in Vietnam.

VI. EVALUATION AND CONCLUDING REMARKS

A. Evaluation

Below we compare our GameGuard system with existing game protection software and evaluate the strength and weakness of our solution. We have closely looked at *X-Trap* and *Nprotect* which are two popular game protection products. These are designed with the bootstrap of protection module is embedded into **GC**; thus, any running game requires a separate X-Trap (or Nprotect) process fully devoted to protecting that single game. If there are two different games running in the same client machine (or a single game running in two accounts), the required two separate X-Trap processes (which both hook to the OS core) would very likely conflict with each other and/or degrade the system performance. Such a bad situation can be easily avoided by our GameGuard system which has the flexibility of having just one **GG** module simultaneously supporting multiple **GC**s.

Game|hack / OS		WinXP SP2 Pro	Win7 Ultimate
Game CrossFire (CF)	Sniper Rifle 1.0	✓	✓
	Ghost Mode Hack	✓	✓
	AssassinWall CF 1039	✓	✓
Game FiFa Online2 (FF2)	Hack FifaOnline2	✓	✓
Audition (AU)	Ice Modz 6041 RC1	✓	✓
	AU Skill helper	✓	✓

Table 1. Successful experiments with popular online games and hack tools

Other game protection products are usually more specific, i.e. they mostly only target certain kind of hack or cheat or only target at protecting a certain given game (or a group of). *Our GameGuard system with two layers of protection is a more general and complete approach* which can potentially defeat most popular kinds of hacks, either memory, graphics or hardware hacks. For example, hacking to the game's memory space to twist game states and data would be

generally defeated by our hiding mechanism. Moreover, if experienced hackers discover this hiding mechanism and want to hook Windows kernel-mode (through the Native API) to unhide our game modules and data, their aim will likely be defeated again by our detection mechanism, which has been up to monitor all potential hack attempts. For example, in order to hook the Windows Native API the hack code must be at kernel level: that is these hackers have to write their hack code as a Windows driver and register that driver with Windows Registry. However, **GG** has already hooked *NtCreateKey* which is called by the OS to create new key for the Registry; thus,**GG** would detect this malicious hack attempt. Hacking to the system graphics (DirectX) or hardware also requires hooking certain Native API calls which can be detected in a similar manner.

We believe that our framework of combining these two protection layers would have defeated more than 90% hack attempts: the expert hackers can only win if they can find and exploit certain undocumented backdoor that would be at a system level deeper than the Native API (which is already little documented and designed for OS internal use only) – that is certainly of very rare chance!

The deployment of a separate module GameGuard with a 2-layer protection mechanism provides us a general, flexible approach. Moreover, the *strength of our GameGuard system is further enhanced by the additional mechanism of communicating with the distributor's servers for auto-update and reporting doubts of new hacks*. A popular downfall amongst many existing hack protection products is that they are too single-minded and fall to recognize variants of old hacks with just a little modification (only in physical aspects). Our GameGuard system has learning capability and is very flexible to update and get enhanced just like most popular anti-virus products: new versions of GameGuard modules and new hack patterns are regularly auto-updated to clients. These functionalities are very rare amongst existing products.

Our GameGuard system still has certain aspects to be improved, i.e. in stability (lower than some existing products but only when in single game mode). Our project is still not complete and we still need to improve the overall compatibility to all Windows systems. In table 2 below we summarize the comparison with the two mentioned existing hack protection X-Trap/N-protect.

Table 2. Comparison with X-trap/N-protect

	Supp. for multi-games/ accts	Stabi.	Ability to work readily with any game	Compat. with all Windows systems	Strength (in protect.against hacks)
Game-Guard	Yes	Higher for multi-games	Flexible	Still being improved	Higher, more powerful (self-eval.)
X-Trap Nprotect	No	Higher for single game	Not flexible	Better	Lower

B. Concluding Remarks

We propose a software architecture and an implementation framework for protecting Windows-based online games against hackers. Our network model is between 3 parties, the game producer, the game distributor and the game player (client), where we aim to support the distributor to fight against cheater clients with minimum cooperation from the game producer. Our protection system is based on module GameGuard, which accomplishes the main protection duties while simultaneously, interacts with the GameClient program so that if a serious hack attempt is detected, both will terminate. Our two layers of protection, in hiding game modules and detecting hacks, integrated with the mechanism of auto-updating/reporting with the distributor server provide a general, powerful and flexible approach in building hack protection products. Our GameGuard system can be used to protect most Windows games, and would defeat most popular kind of hacks from 3 different categories – memory hacks, graphics hacks and hardware hacks. Our GameGuard system is also sound against attacks directed at the protection modules: we achieve authenticity of our protection modules, assuming the authenticity of the GameClient from the producer.

Based on our framework, we have implemented a hack protection system which is being extensively experimented at VTC, the Vietnam Multimedia Corporation, with very encouraging results. We use extensive hooking techniques to interfere with Windows systems (particularly with the Native API) so that our GameGuard module can hide all game modules from the view of average cheaters as well as successfully detect most hack attempt. Our initial evaluation shows that our product outperforms certain

existing hack protection products in important aspects such as suitability for multi-game/multi-account mode, power and robustness to protect against many popular categories of hacks, as well as flexibility to learn fighting against the new ones.

Our future work focuses on two aspects of this product. We will further study of methods for better recognizing hack patterns, so that our system can detect variants of a core hack pattern. This requires using certain methods in AI areas but we want to avoid heavy computation that would critically slow down our detection process. Finally, we propose to study for better assurance methods on the authenticity of our GameGuard modules where we don't need to assume the authenticity of anything from the game producers.

VII. REFERENCES

[1] Bethea, D., Cochran, RA., and Reiter, MK. 2011. Server-Side Verification of Client Behavior in Online Games. Server-side verification of client behavior in online games. ACM Trans. Inf. Syst. Secur. 14(4): 32.

[2] Chang, H., Atalah, M., Protecting Software Code by Guards, ACM CCS-8 Workshop on Security and Privacy in Digital Rights Management, pp. 160-175.

[3] GautheierDickey, C., Zappala, D., Lo, V., and Marr, J. 2004. Low- Latency and Cheat-proof Event Ordering for Distributed Games. In Proc. NOSSDAV '04, pp. 134-139.

[4] Grimen, G., M¨onch, C. and Midtstraum, R. 2006. Tamper protection of online clients through random checksum algorithms. In Proc. of ISTA 2006, pp. 67–79, May 2006.

[5] Hoglund, G., McGraw, G. 2007. Exploiting Online Games: Cheating Massively Distributed Systems, Addison-Wesley Professional.

[6] Kabus, P., Terpstra, W. W., Cilia, M., and Buchmann, A. P. 2005. Addressing cheating in distributed MMOGs. in Proc. of *NetGames '05,* pp. 1-6.

[7] Mönch, C., Grimen, G, and Midtstraum R, Protecting online games against cheating, ACM Netgames'06, pp. 1-11.

[8] Neumann, C., Prigent, N., Varvello, M., & Suh, K. Challenges in peer-to-peer gaming. ACM SIGCOMM Computer Communication Review. 37, 1 (2007), pp. 79-82.

[9] Pritchard, M. 2000. How to Hurt the Hackers, in Game Developer Magazine, Jun. 2000. pp. 28-30.

[10] Webb,J. and Soh,S 2007. Cheating in networked computer games – A review. In Proc. of the 2nd international conference on Digital interactive media in entertainment and arts - 2007. ACM International Conference Proceeding Series, Vol. 274.

[11] Yan, J. & Randell, B 2005. A systematic classification of cheating in online games. In Proc. ACM NetGames '05, pp. 1-9.

[12] Yan,J., and Choi, HJ. 2002. Security Issues in Online Games. The Electronic Library, Vol. 20, No.2, 2002.

[13] Yahyavi, A., & Kemme, B. 2013. Peer-to-peer architectures for massively multiplayer online games: A Survey. ACM Computing Surveys (CSUR), v.46 n.1, p.1-51

[14] Kang, AR., Woo, J., Park, J., & Kim, HK. 2013. Online game bot detection based on party-play log analysis. Computers & Mathematics with Applications 65(9): 1384-1395.

[15] Bui, TL. and Nguyen,VK. 2010.GameGuard: A Windows-based Software Architecture for Protecting Online Games. In Proc. of Symposium on Information and Communication Technology, SoICT' 2010, Hanoi, Vietnam, August 27-28.

[16] Real-world products:
X-Trap, Nprotect: http://global.nprotect.com

ShoxGuard: http://shock-guard.com

XRay: http://en.wikipedia.org/wiki/X-ray

Warden: http://en.wikipedia.org/wiki/Warden

ShoxGuard: http://shock-guard.com

XRay: http://en.wikipedia.org/wiki/X-ray

Warden: http://en.wikipedia.org/wiki/Warden

AUTHORS' BIOGRAPHIES

Luan Bui The is an R&D engineer at Vietnam Multimedia Corporation (VTC).He received his Bachelor (2002) in Faculty of Mathematics and Informatics, and Master degree(2010) in School of Information and Communication Technologies (SoICT) of Hanoi University of Science and Technology (HUST). His research interests include security issues in game online and software development.

Khanh-Van Nguyen is a Senior Lecturer at the School of Information and Communication Technologies (SoICT) of Hanoi University of Science and Technology (HUST), where he leads the Software Engineering and Distributed Computing Lab (SEDIC Lab). He received his Ph.D. degree in Computer Science from the University of California, Davis (2006) and also spent one post-doctoral year (2007-08) as a CNRS researcher at University Paris 7 & 11.

His main research domain is algorithms and theoretical models for computer networks and distributed computing.