Sequence-Based Bot Detection in Massive Multiplayer Online Games

Christian Platzer
Secure Systems Lab, Vienna University of Technology
Treitlstrasse 1-3 4/183-3, 1040 Vienna, Austria
Email: cplatzer@seclab.tuwien.ac.at

***Abstract*—With the growing popularity of massive multiplayer online games (MMOG), the demand to protect these games and especially the participating players is increasing likewise. In this paper, we discuss a sequence-based solution to protect online games from being exploited by fully automated bots. Our research is based on various commercial and non-commercial bot implementations for the renowned game *World of Warcraft*. Our evaluation suggests, that a sequence-based detection technique is applicable even for mildly skilled players and effectively protects online games from a negative impact on game economics that causes both, grief to casual gamers and destroys game balance. Our implementation is fit to be deployed on either client- or server side, posing the first player-usable bot detection tool for *World of Warcraft*.**

I. Introduction

Online games ran through an astonishing development in the past ten years. With November 2010, the renowned massive multiplayer online game (MMOG) *World of Warcraft* hit the 12 million subscribers mark and is still gaining new users every day [1]. Considering all major games, a total of over 21 million people around the globe participate in multiplayer online games [1]. At the same time, the demand to protect these games and especially the participating players is increasing likewise.

One major problem comes along with almost every MMOG that exists today: *Bots*.
Where online games are concerned, the term *bot* usually entitles a program or method that allows a user to play the game partially or completely unattended. At first thought this might not seem like much of a problem, but the impact of bots on the game world and its population can be very serious. Gold farmers, for instance, which produce in-game currency at a far higher rate than ordinary players ever could, alter the prices of traded goods such that casual players cannot afford them anymore. As a result, in-game currency and items are traded via ebay and payed for with real money. At the time of this writing, 5.000 pieces of *World of Warcraft* Gold were worth roughly 20 US Dollars. A sum that many consumers deem worthy to make up for the additional spare time. While automatic gathering of resources, also called farming, is certainly one application of game bots, this particular segment is far better covered by human players in low-wage countries [2]. Besides, participating players are only secondarily affected by gold farmers when they experience raising prices. Furthermore, the game company can introduce money sinks to stabilize these effects.

A far more annoying application of bots is their deployment as level-bots or active team members in battle groups. The restricted action set of these programs and their inability to properly react to its surroundings may work a dedicated player's last nerve. Currently, human players depend on the game company to take care of bots flooding their servers. To counter this issue, we propose a novel detection technique to distinguish genuine players from game bots by comparing their action sequences. Our contribution is threefold:

1) We discuss a detection technique which uses Levenshtein Distances between action sequences to detect in-game bots.
2) We implemented our detection tool on a server- and client-side instance and an in-game, client-sided add-on that is capable of monitoring its own event horizon.
3) We evaluated our approach based on a worst-case scenario. We utilized several commercial bots, a self-written bot and a group of human players to produce the necessary data.

II. Motivation

The motivation for using a bot is simple. It produces an unfair advantage for the player compared to other participants. Similar to other games, combat interaction yields the best results in *World of Warcraft*. Killing monsters is the easiest way to gain experience, gold and resources. Once the maximum level is reached, additional items can be earned by engaging in Player versus Player (PvP) combat. This is perilous terrain for bot programs simply because the program's actions (movement and action sequences) are visible to other players of the same team. And these players usually tend to be peeved when two players out of a ten player team are bots. It drastically lowers the chances of winning an encounter, thus causing grief most gamers are not willing to accept. Once a player files a complaint, game companies do not hesitate to suspend the account in question or ban it completely. Unfortunately, spotting a bot based on its visible actions is not trivial. PvP combat tends to be hectic, restricting the time human players can spend watching teammates and judging their actions. Even in the smallest battleground, a bot with a decent movement- and action pattern [3], [4] is very hard to distinguish from human players. As a result, every bot comes with some inherent weaknesses which can be used in a detection attempt.

A. *Waypointing*

Compared to a human, a gaming bot is very limited in its capabilities to navigate the virtual environment. Modern MMOGs provide a rich, three dimensional game world where movement can occur in every direction. Additionally, certain obstacles and hindrances can exist, that prevent a player from getting there. Consequently, an implementation with random movement patterns is not an option, because it can easily result in a trapped character or an avatar that runs against a wall for hours. To prevent this, a waypoint system is mandatory. The desired path can either be recorded by the player or is shipped with the bot program. When starting the bot, the character traverses more or less the same path, with small variances depending on the environment and the bots sophistication level. This fact is actually a huge limitation because it renders a bot detectable if the traveled path is properly analyzed[5]. In smaller environments, like battlegrounds, such a detection technique is unfeasible due to the limited time the players spend in the instance and the relatively small size of the environment.

B. *Sequencing*

Just like the waypoint system, an element every bot has in common is the reaction to certain events. The main goal still is to kill enemies within the game. The decision which spells to launch or which skills to use to successfully reach this goal is again pre-defined in the bot's configuration. We entitle the sum of all abilities used from the point where an enemy is engaged until the enemy (or the own character) dies, as a *combat sequence*. A side-effect of the fixed configuration is, that these sequences are always very similar. Compared to a human player, the bot can make no, or very restricted choices based on the game environment. As a result, the abilities used to kill a single enemy in the game will always be very similar. This is even more blatant in PvP combat, where players use a very large variety of different skills in various combinations to thwart their enemy. Here we attach our detection algorithm and implement a behavior-based analysis of the player's actions.

III. Related work

The two previously mentioned weaknesses are the basis for most research conducted in this area. In [6] [7], for instance, a system is introduced which aims to detect bots by tracking their path information and trajectory respectively. Although desigend as a general-purpose solution for all kinds of games, it is ultimately restricted to a very narrow category of fast-paced action games, where turning the avatar with the keyboard is not an option.

In [8] [9], the authors propose the use of *captchas* that automated scripts cannot solve. The same principle is of course possible in MMOGs. The analogy would be a maze, where an avatar had to find the exit or overcome certain obstacles. Such an approach would certainly work but the number of players that decide to quit because they were unexpectedly trapped inside a maze while in the middle of PvP combat is one of many reasons why no game company would even consider such a method. In comparison, our approach is completely transparent to the end-user and has no influence at all on the gaming experience of an individual player and can be deployed at the server- or client side.

The most closely related paper is introduced by Kuan-Ta Chen et al. [10]. There, the authors utilize a traffic-analysis approach to identify bots for the game "Ragnarök Online". A major drawback of the proposed method lies in the fact that it is custom-made for that game. The authors try to distinguish traffic generated by the official game client from traffic generated by standalone bot programs through statistical analysis of packet transfer properties. Unfortunately, this approach does not work on modern games, as they mostly implement largely ping-independent command queueing on the client-side. This fact also renders the method described in [11] ineffective, where the timing of keystrokes is used to distinguish between humans and bots.

A more general view on security in online games is presented by Greg Hoglund and Gary McGraw [12]. In their book, they cover a wide section of game security topics, ranging from the legal issues over bug exploits and hacking game clients to writing bots. Although the book provides a good introduction into several gaming security areas, it concentrates mostly on the attacker's point of view and does not provide concrete solutions on how to detect or prevent botting.

Another piece of related work was done by Jeff Yan et al. [13], where the most important cheating methods were analyzed and categorized. This work can be seen as the theoretical foundation of our system. In the end, a bot is just another form of cheating. In [14] the concept was extended and a detection approach for aiming bots was introduced. Other than bots in MMOG's, aimbots are deployed in games where heightened reaction is of advantage, like in first person shooters for example. Time is not a factor for queue-based games, however. Even with a 200ms delay, those games are designed to work without any drawbacks.

Finally, path-based detection was discussed in [5], which represents the complementary research to this paper.

IV. Detection Approach

Our approach is based on the combat sequence each avatar produces when engaging an enemy. For each fight, a list of actions is extracted. The difference between subsequent combat sequences is measured in terms of their levenshtein distance [15]. Killing each monster with the same combination of skills would therefore result in a distance of 0. We calculate the similarity value v_i for a particular combat sequence c_i following the formula

$$v_i = \frac{\sum_{j=(i-k)}^{i-1} d_{levenshtein}(c_i, c_j)}{k},$$

with d being the levenshtein edit distance. This method takes an interval of size k before the combat sequence in question and averages the levenshtein distances to the current one (i).

Our final goal was, to detect all tested game bots and successfully distinguish them from humans playing the game. Our implementation is designed to operate on both, the game client itself, or the server where the data is actually processed. The major difference when operating on the client-side lies in the event horizon. The visible surroundings to a specific player are always limited to a globe encircling the avatar. This limitation is necessary to restrict the amount of data that has to be transferred from the server to the game client. In the special case of battlegrounds, the event horizon includes the whole instance making them an ideal target for client-side bot detection.

A. Implementation

We decided to use the logging facility *World of Warcraft* provides to gather the data. This logging facility is primarily designed to enable backtracking through fight sequences. It is heavily used by large player communities to decipher who messed up a fight in dungeons where 40 people are playing in a single raid group. A nice side-effect of this logging facility is, that it keeps track of *other* player's actions as well. Therefore, it is also possible to simply follow a character and record the produced combat sequence to find out if a bot is playing. For the server-side, it was simply a matter of logging the actions caused by a single character and, therefore, not a challenging task.

We implemented this tool in C# using Visual Studio 2008. It parses combat logs produced by *World of Warcraft* and generates a list of combat sequences for oneself and each surrounding player. Furthermore, it is important to add the right elements to a combat sequence, since it forms the necessary alphabet to operate on. It is also the main reason, why generic approaches for anomaly- or cheat-detection cannot cope with the speed and quality of today's game bots. It requires a deeper understanding of the game dynamics, which is usually only attained by either game developers or long-term players.

B. Evaluation

Our baseline consists of a a set of combat sequences produced by bots and human players. In total, the recorded data comprises 485.761 combat actions, whereof 266.318 (55%) were produced by bots. The human traces were collected by a total of 41 individuals over a period of several weeks. The bot traces were produced by our self-written program (AltNav), MMOMimic[3], FairPlay[4] and ZoloFighter[16]. During the evaluation phase, we used the bot trace with the highest variance and the human trace with the lowest variance in combat routines with the only restriction that they have to be of the same class (e.g. Fighter, Mage, etc..). Character classes exhibited different combat schemes, depending on the variety of skills available to them. Therefore, comparing them to each other is not feasible.

To get comparable results, we executed the following steps:

- We modified each bot to include conditional actions based on player health, pet health, encountered mob etc. The goal was to make the bot behave as much as a human player as possible. It should be noted here, that the majority of bots do not use such a high level of randomization by default. They mostly incorporate a single fight sequence which is simply repeated until either the target or the player dies. For the evaluation we always used the **most diverse** bot result.
- After recording the bot's actions, we switched over to a human player with the directive to kill the same monsters and traverse the same route.
- The levenshtein distance was calculated for each combat sequence.
- The result graph was generated to visualize the distances and decide which detection metric to apply, respectively, which evaluation strategy is the most promising.

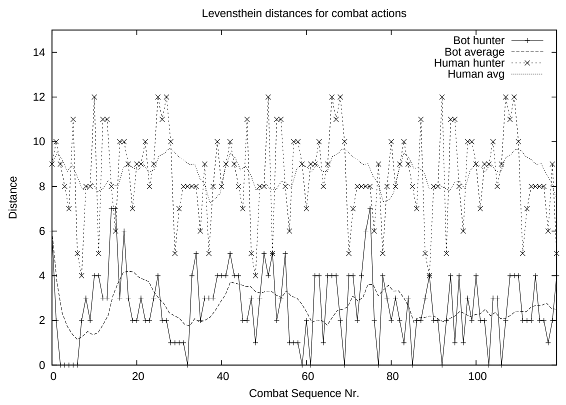

Fig. 1. Concrete levenshtein, k=1, average=10

Figure 1 shows a result plot for a hunter played by both, a human and a bot. The trace shows 90 minutes of actual (human) gameplay which resulted in more than 120 distinct combat sequences. Therefore, the time interval between each combat sequence is 45 seconds. The recording shows non-PvP action which happens at a far lower pace than battlegrounds. Here, k was set to 1, thus, only the directly preceding combat sequence was taken into account. For the average curve, 10 concrete distances were considered. This figure clearly shows the different styles of a bot compared to the human player. For this type of interaction, the human player acts within a levensthein distance interval of $[4, 12]$ with an average of slightly above 8, while the bot acts in an interval of $[0, 7]$ with an average of around 3. Although feasible, a drawback of this method is that once a bot knows how it is calculated, the metric can be dodged by alternating between two completely different combat sequences, which in turn results in very high values. As a countermeasure, we took the *minimum* levenshtein distance for a certain interval k. The exact values v_i are processed as

$$v_i = min(d_{levenshtein}(c_i, c_j)) \quad \forall \{j | k \leq j < i; j \epsilon N\},$$

with N being the overall amount of combat sequences that have to be evaluated, and k the range again. Figure 2 shows the same sample as before, with the values calculated by this new method. Whenever a combat sequence is repeated within the interval k, it causes the value to be zero. With this formula, lower discrete and averaged values can be expected for humans and bots. Here though, the bot causes a certain amount of zero-passes, a human player does not, which is a good beginning for a detection metric.

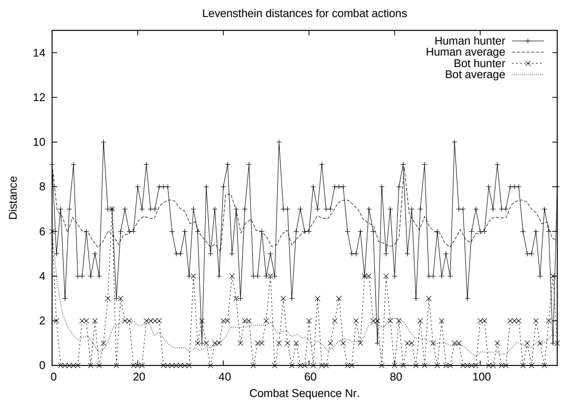

Fig. 2. Minimum levenshtein, k=5, average=10

Detection metric

In our first approach, we used the number of subsequent zero-passes caused by the player as an evaluation metric. Whenever more than four subsequent values were at zero, an alert was raised, and the player was identified as a bot. With this method, we were able to detect all bots and all players from our test set reliably. During the evaluation, however, a human, playing a Mage, caused one or sometimes even two zero-passes. A more passive player would certainly be able to cause several zero-passes in a row and, therefore, cause a false positive. A solution to this problem is, to extend the above formula to:

$$v_i = min(d_{levenshtein}(c_i, c_j)) \quad \forall \{j | j \neq i; j \epsilon N_k\}.$$

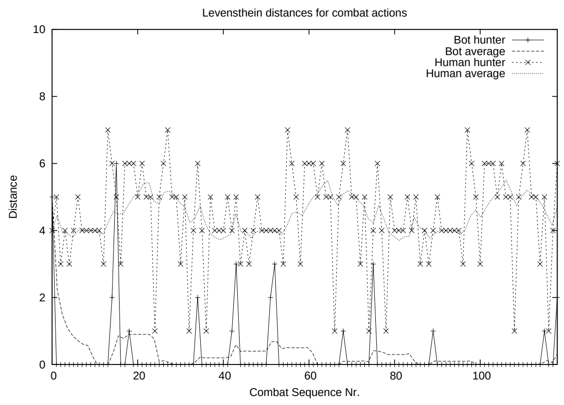

Fig. 3. Minimum levenshtein, N_k=40, average=10

N_k has to be chosen such, that it covers the desired time-frame. In Figure 3, it was set to 40, resulting in effectively 30 minutes of played time. The result of the formula is, that a zero value is produced whenever the same combat sequence is repeated within this time-frame. Other than before, we now count the zero-passes of the *averaged* curve. This method has a much higher accuracy rating, because the variance a human introduces, causes at least one unique combat sequence out of 10. The threshold for this method was set to four subsequent zero passes for the average curve. For a bot to be detected as such, it has to produce 14 non-unique combat sequences within the given time frame. This threshold is admittedly low, but during the evaluation process, it showed that especially players new to the game, use a limited amount of their abilities at the beginning. With these settings, we were still able to detect all available bot traces reliably. In a PvP environment, we had far less problems to identify bots. Even with the first introduced detection approach, the intervals the characters usually operated in, were between $[6, 20]$ with an average of over 14, while the bot acted in an interval of $[0, 4]$. PvP gameplay has the property to be highly interactive. The resulting fight sequences have a high variance because human enemies have a broader reaction spectrum than computer-controlled ones, resulting in a heightened demand of proper reactions. The bot, on the other hand, executes its predefined sequence every time, regardless of its surroundings.

V. Data collection

To produce a versatile amount of human traces and reduce the amount of bias our sample introduces, we decided to start our own *World of Warcraft* server. We used the *ArcEmu* server emulation [17] to create a free *World of Warcraft* shard that can be played with the most recent client without modification. The test also provided the necessary results for a server-sided appliance. Originally planned to produce the main evaluation data, the experiment did not exactly yield the desired results. After conducting the test, the reasons for its poor performance are obvious:

- Players with a working game account want the game to be perfect. A requirement that can never be met with a reverse-engineered game server, where certain functionalities are implemented based on a guess how they might work.
- New players hardly see the reason why they should play the game since it offers almost no interaction with other players. Production servers hosted by Blizzard are usually populated by 5000+ people, a number that can not even be hoped to be achieved here. Therefore, the game experience for new players is just like a badly maintained single-player game.
- Hosting a free-shard for a game that must usually be paid for, raises some legal issues. Game companies explicitly prohibit this form of service. Therefore, the test had to occur in a closed environment, in this case the university campus. This, in turn, limited the number of participating players.
- A short-term server like the one at hand, offers only certain motivations for players. The only reason that they

actually do it is, because they get the chance to see very expensive or hard to obtain equipment or a dungeon which is normally unaccessible to them. In any case, the motivation for those players to advance their characters is very limited.

The test was conducted during a 4-week period. Before, and during the experiment, 70 accounts were issued, from which 35 logged in to the server at least once. Player recruitment was done via a public university forum and mailing lists with undergraduate students. The setup was such, that the players started with a level 84 character which they were supposed to advance to level 85, to be able to wear the new equipment and enter the desired dungeons. The data gathered during the advancement period was the main point of interest. After one month of server uptime, only 10 people really advanced their character to level 85. Nevertheless, none of the recorded game traces produced a false positive and therefore support our detection approach. Setting up the starting characters, the game environment and a stable host proved to be a time-consuming task. During the test, many players issued requests about missing items or functionality within the game that had to be looked upon. All things together, it does not pay of to gather player data by setting up a home-brew game server. The ideal case would be having direct access to the log files of a production server hosted by the game company to further advance such an approach.

We actually contacted Blizzard Online Entertainment and offered a cooperation in this regard. Unfortunately, after overbearing the initial obstacles of not being taken serious, it became clear that the company has no intention whatsoever, to share any relevant information on where the actual frontier between cheaters and game developers lies. Furthermore, our request to share log data, or test our tool on game traces was rejected as expected due to security and privacy concerns.

VI. Conclusion

In this paper, we presented a sequence-based approach to protect current massive multiplayer online games from being exploited by bots. The implementation is designed to be to usable on the server side, to raise alerts for possibly cheating players, as well as on the client-side to provide a tool for honest players and therefore detect misbehaving fellow participants. With our proof-of-concept implementation for *World of Warcraft* , we showed that it is possible to reliably categorize over 485.000 action sequences and tell the difference between automated and human players.

The approach is not limited to *World of Warcraft* but is feasible to every queue-based MMOG provided that the necessary domain knowledge is available. While several solutions to toughen a game against bots are thinkable, the effort to actually implement them is rarely appointed. One reason might be insufficient resources. After all, not every game is as successful as *World of Warcraft*. Competing game developers have a hard enough life with keeping their games free of bugs. For the major players, however, the conclusion is a different one. Having the resources to protect games against bots, but not using them, either means the effects are inside their self-set tolerance zone or *wanted* in some respect. In the end, a *boting* player still is a *paying* player. With our tool, the gamers themselves are given the power to detect cheaters and report them to the company. And user complaints are hard to ignore for game companies.

Acknowledgments

The author would like to thank Stefan Mitterhofer for his hard work setting up the ArcEmu environment and providing the free-shard to gather movement data. Additional thanks go to Peter Tummeltshammer for his very first implementation of AltNav and to the reviewers for their valuable input for the final version. The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n. 257007 (SysSec) and the Austrian Research Promotion Agency (FFG) under grant 820854 (TRUDIE).

References

[1] http://www.mmodata.net/, “Mmodata charts version 3.3,” 6 2011, last accessed: 14.06.2011.

[2] J. Dibbell, “The life of the chinese gold farmer,” *Life*, 2007.

[3] *MMOMimic*, last accessed: 15. June 2011. [Online]. Available: http://www.mmomimic.com

[4] *WoW Bot:FairPlay-Bot*, last accessed: 22. July 2011. [Online]. Available: http://www.fairplay-bot.de/

[5] S. Mitterhofer, C. Platzer, C. Kruegel, and E. Kirda, “Server-side bot detection in massively multiplayer online games,” *IEEE Security & Privacy*, pp. 29–36, 2009.

[6] K.-T. Chen, A. Liao, H.-K. K. Pao, and H.-H. Chu, *Game bot detection based on avatar trajectory*. Entertainment Computing-ICEC, 2009.

[7] K.-T. Chen, H.-K. K. Pao, and H. Chang, “Game bot identification based on manifold learning,” in *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*. ACM, 2008, pp. 21–26.

[8] R. V. Yampolskiy and V. Govindaraju, “Embedded noninteractive continuous bot detection,” *Comput. Entertain.*, vol. 5, no. 4, pp. 1–11, 2007.

[9] P. Golle and N. Ducheneaut, “Preventing bots from playing online games,” *Comput. Entertain.*, vol. 3, no. 3, pp. 3–3, 2005.

[10] K.-T. Chen, J.-W. Jiang, P. Huang, H.-H. Chu, C.-L. Lei, and W.-C. Chen, “Identifying mmorpg bots: a traffic analysis approach,” in *ACE ’06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*. New York, NY, USA: ACM, 2006, p. 4.

[11] H. Kim, S. Hong, and J. Kim, *Detection of Auto Programs for MMORPGs*. Springer Berlin, 2005, vol. 3809.

[12] G. Hoglund and G. McGraw, *Exploiting Online Games: Cheating Massively Distributed Systems (Addison-Wesley Software Security Series)*. Addison-Wesley Professional, 2007.

[13] J. Yan and B. Randell, “A systematic classification of cheating in online games,” in *NetGames ’05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. New York, NY, USA: ACM, 2005, pp. 1–9.

[14] S. Yeung, J. C. Lui, J. Liu, and J. Yan, “Detecting cheaters for multiplayer games: theory, design and implementation[1],” *Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE*, vol. 2, pp. 1178–1182, Jan. 2006.

[15] W. I. Levenshtein, “Binary Codes Capable of Correcting Deletions, Insertions and Reversals,” *Soviet Physics Doklady*, vol. 10, pp. 707–+, Feb. 1966.

[16] *WoW ZoloFighter Bot, discontinued 15.04.2009*, last accessed: 15. June 2011. [Online]. Available: http://www.zolohouse.com/wow/wowFighter/

[17] *ARCEmu World of Warcraft Server Emulator*, last accessed: 15. june 2011. [Online]. Available: http://arcemu.org/