Server-side Bot Detection in Massive Multiplayer Online Games

Stefan Mitterhofer[*], Christian Platzer[*], Christopher Kruegel[§], Engin Kirda[¶]
[*] Secure Systems Lab
Technical University Vienna
{sm;cplatzer}@seclab.tuwien.ac.at

[§] University of California, Santa Barbara
chris@cs.ucsb.edu

[¶] Eurecom, Sophia Antipolis, France
kirda@eurecom.fr

Abstract

One of the greatest threats that Massive Multiplayer Online Games (MMOG) face nowadays is botting. Botting is a form of cheating where a player uses a script to automate actions in a game without actually playing herself. This has a severe adverse effect on honest players and impacts their motivation to continue the game, threatening the subscription-based business model of online game providers. However, if game companies make an effort at all to automatically detect bots, it is done through signature checking for suspicious programs on the client-side, essentially relying on information from an untrusted source outside their control. To address the botting problem, we propose an automated approach to detect bots on the server-side, solely based on the activity of a character. Our approach is completely transparent to the end-user and takes advantage of the fact that bots follow a script that guides them through the virtual world. More precisely, by analyzing the movement data of a character, we extract waypoints and detect paths that are taken repeatedly. This allows us to find movement patterns that repeat frequently, indicating that a character is controlled by a script and not a human player.

Keywords: gaming, security, online games, cheating, gaming bots, bot detection, world of warcraft, mmorpg security, waypoint extraction

1 Introduction

Massive Multiplayer Online Games have soared in popularity in the past few years, with more than 16 million active subscribers in 2008 [1]. The market leader alone, Blizzard Entertainment's *World of Warcraft* (WoW), surpassed 10 million subscribers in January 2008, raking in an estimated $150 million in subscription fees per month. With such amounts of money at stake, it does not come as a surprise that game companies are interested in keeping their paying customers satisfied and threats to their revenue base at bay. One of these threats is *botting*, a form of cheating [2] in which a player uses a program that can play the game with a minimum of human interaction or even completely unattended. To avoid confusion in the subsequent discussion, we quickly introduce some terminology that will be used throughout the paper:

- A *character* is a game avatar that is controlled in the virtual world.
- A *bot* is a computer program that controls the character and plays the game largely or completely unattended.
- A *player* is a human that plays the game.

- A *route* is the course of movement that a character performs in the game world.
- A *path* is a sequence of locations that the character visits in the game world; a route can lead several times over the same path.

The reason for using bots is that parts of the game are inherently repetitive (and possibly boring). In particular, players often need to kill large numbers of enemies to gain experience points and to earn gold (a process called *farming* or *grinding* in the gaming community). This is required to improve the character and progress further in the game. It did not take long until the first programs were created that could free the player from these chores and automate the necessary steps. They were aptly named *bots*. Running a farming bot means that the character gets experience points and gold without having the player invest any time in the game, as the bot can reap those rewards very efficiently 24 hours a day, without fatigue or boredom.

Interestingly, bots are not only used by players to improve their own characters. The reason is the booming market for items, gold, and complete characters that are sold on the Internet to players that do not want to go through the hassle of obtaining them through hours of playtime. Thus, bots can be used to acquire in-game goods that can later be sold for real money. According to estimates, up to 100,000 people in China are employed as gold farmers for MMOGs to gather gold and other in-game items and build up characters for the sole purpose of selling them. Even cheaper than these *china farmers* are bots, making them the ideal tool for acquiring for-sale items.

1.1 Impact on the game

The impact of botting on a game can be substantial. In particular, the presence of bots can lead to significant levels of inflation in the game economy, as they create unusual amounts of gold by slaying monsters without end. The high amount of circulating gold increases prices for items, until honest players cannot afford them anymore. Moreover, bots often occupy good farming areas and kill other players. This has a severe adverse effect on the day-to-day gaming experience for honest players and impacts their motivation up to the point of making them quit, which has a very real impact on the game company's revenue. For example, the "Lineage" game series is suffering from these effects [3].

1.2 Current bot countermeasures

Looking at their negative impact, it does not come as a surprise that online game providers usually forbid the use of bots, trying hard to keep botting at a minimum in their games. It is, however, difficult to recognize whether a character is played by a bot. The reason is that a bot plays the game obeying the game rules. Currently, when game providers take any action at all to detect bots in their virtual worlds, it is mostly through human interaction. More precisely, these games often deploy a report system that allows players to report suspicious characters. When a character is reported, a game moderator approaches this character in-game and tries to start a conversation, thus checking if a human is at the keyboard. However, this method is inefficient and does not scale for worlds with tens or hundreds of thousands simultaneously online characters.

To the best of our knowledge, the only automated tool against bot programs is the *Warden* [4], a monitoring application that protects *World of Warcraft*. The Warden runs on the player's computer while she plays WoW and checks for suspicious programs such as debuggers or bots. It reports back to Blizzard, and violations result in temporary or permanent account bans. However, the Warden has several shortcomings: It can only perform signature checks for known programs, which means it will always be a step behind bot writers. Another important drawback is that it is running on the client's computer, which is completely out of Blizzard's control. This means that the results of checks ultimately cannot be trusted. Additionally, there are some simple work-arounds, such as starting the game in guest mode on an administrator account. In that case, the operating system prevents the Warden from accessing the processes at higher privilege levels. Not surprisingly, privacy issues have also been raised [5].

1.3 A novel approach: Behavior-based bot detection

In this paper, we propose a novel approach that relies solely on the server-side analysis of a character's behavior to expose bots, avoiding the drawbacks of client-side solutions. To this end, we take advantage of an intrinsic feature of bots, namely the fact that they are controlled by a script that automates a specific sequence of actions that are constantly repeated. In this paper, we focus on analyzing the game character's movement only. To achieve this, we extract waypoints that describe the traveled path and find repeating patterns in the route taken. We implemented and evaluated our approach, taking *World of Warcraft* as an example.

The remainder of the paper is structured as follows: First, we discuss in more detail how bots work. In Section 3, we introduce our new detection approach. Subsequently, we evaluate how effectively this approach distinguishes humans from bots in Section 4. In Section 5, we discuss related approaches and their shortcomings, before we conclude the paper and provide an outlook on our future work.

2 Inside a bot: How bots work

Bots are designed to carry out simple and repetitive portions of the gameplay, relieving the player from wasting her time with it. Usually, bots move around in the virtual world, killing non-player mobile entities (mobs). To achieve this, the player has to specify a certain script that determines which actions are chosen by the character, depending on the game situation. This includes the sequence of spells that are cast in combat, under which circumstances the character heals itself, and whether it picks up loot from enemy corpses. The player additionally sets a path that the bot should follow, and the enemies it should attack.

2.1 Interfacing with the game

To carry out a script, a bot first needs to collect information about its surroundings in the virtual world, and then react to it by sending commands to the game. While humans simply need to look at the computer screen to determine where trees, mountains, and mobs are, bots need to apply different methods to collect this information.

The most common way to obtain information about a character's surrounding is through memory reading. Similar to a debugger, the bot program hooks into the running game process and reads the list of mobs and their locations directly from the representation inside the memory of the game. While this works for mobs, it is not feasible to obtain the necessary terrain information in this way. The reason is that, based on the map files, the game engine decides on the fly if the current step can be taken or if there is an obstacle large enough to block the character. Hence, it would be necessary to re-implement parts of the game engine and a sophisticated route-finding algorithm to be able to calculate a sensible movement route in advance. To avoid this problem, the player usually needs to "teach" the bot a certain path by running it herself, with the bot recording waypoints that it will follow later to replicate the path.

Another way to get information about the game surroundings, which works in particular for *World of Warcraft*, is through writing an add-on for the game's scripting engine (which offers a limited API to query and manipulate game information). The add-on obtains the desired information in this way, but it is not possible to control the game through the API. Thus, the add-on outputs the information to the user interface by color-encoding certain pixels. These pixels are then read and decoded by a desktop automation scripting environment such as AutoIt, which decides about actions and communicates them to the game client.

2.2 Taking action

When enough information is collected, the bot program decides which action to take. The actions are usually communicated to the game through emulated key presses and mouse clicks, although there are also bots that achieve this by directly modifying memory of the game process.

3 Analysis mechanism

3.1 Main challenge

Contrary to human players who roam freely in a virtual world, bots follow a prescribed or previously recorded list of waypoints when they move around. This suggests that while the movements of a human player, drawn on a 2-dimensional map (a movement graph), will look more or less random, such a graph produced by a bot will show that certain paths were taken again and again. Bots usually follow paths more or less exactly, but leave them sometimes to attack a monster or collect loot from a slain enemy. These small irregularities increase the difficulty in detecting bots through movement analysis.

From looking at a movement graph, it is often straightforward for a human to recognize repeated movement patterns. However, this question needs to be answered automatically to scale to large populations of players. We propose an approach that automatically detects if a bot is in control.

3.2 Collecting data

When a player moves her character in the virtual world, the game client regularly sends packets with the new coordinates to the server. Hence, the movement coordinates are readily available on the server-side. We log the character's movement and use this *game trace* as the basis of our detection approach.

3.3 Building the route

In the first step, we reconstruct the route that the character took in the game world simply by connecting the coordinate dots that arrive at the server. Then, we process the dots by using the Douglas-Peucker line simplification algorithm [6]. The output of the algorithm are simplified lines that resemble the original lines within certain tolerance levels, but consist of fewer vertices. In this way, we eliminate dot clusters that occur when multiple packets are sent in quick succession at the same location. This happens, for example, to update the rotation when the character is turned with the mouse. The simplification helps the waypoint extraction algorithm to concentrate on areas where the dots accumulate because the character passed there several times. The result of this step is a route represented as an ordered sequence of dots as displayed in step (1) of Figure 1.

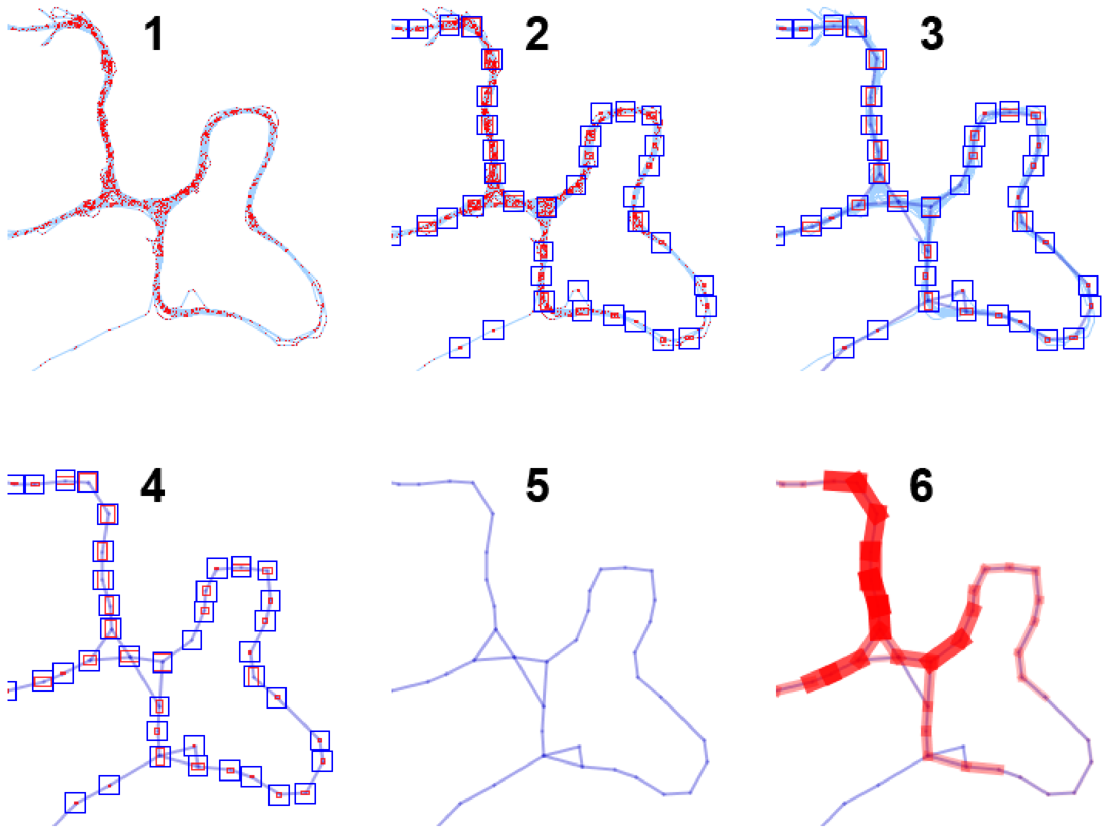

Figure 1: The movement data processing steps: (1) Route and dots, drawn after dot burst removal. (2) Waypoints extracted from dot accumulations. The smaller red rectangles mark the extracted clusters. (3) Simplified path, created by connecting waypoints with straight lines. (4) Original route removed. (5) Simplified path, waypoints removed. (6) Line segments show the number of times they were traveled.

3.4 Extracting waypoints

When the same path is taken several times, dots of different runs are accumulating to clusters of high dot density. In our next step, we extract these clusters and create a waypoint around every cluster. The waypoint is an area of a fixed diameter centered on the base of the cluster. We choose the diameter as small as possible but large enough that, if a waypoint location on a path is traveled multiple times, the area of the waypoint is counted as passed in every run.

We utilize a custom clustering algorithm based on the common k-means algorithm [7], where the cluster size is limited to the waypoint size. As our proximity measure, we use the Euclidean distance because it is the most accurate measure for this purpose. By constraining cluster growth to the waypoint size, we make sure that a waypoint always contains all the dots of the respective cluster and its area is passed every time the character moves through one of the dots. Finally, we remove overlapping waypoints by keeping the one with the larger number of dots.

As shown in step (2) of Figure 1, we end up with a set of waypoints distributed over the entire length of the path that the character took, a little sparser in sections of the path that were traveled less often, and denser along a ”beaten track”, as the dots of each run accumulate there.

3.5 Abstracting the route representation

We then use the set of waypoints to create an abstracted and simplified representation of a route. This is done by iterating over the initial sequence of dots, and, for each dot, recording the corresponding waypoint that is passed along the way. When there is no waypoint for a dot, we simply move on. As each waypoints is uniquely identified, a route can be described by the sequence of waypoints as depicted in (3) of Figure 1. This description is called a *movement sequence*.

3.6 Finding repetitions in the route

For tackling the challenge of finding repetitions in the movement sequence, we leverage previous work in the area of bioinformatics. In this field, much effort has been dedicated to analyzing long sequences of DNA or proteins for interesting subsequences. We use an extended suffix array [8] to find repetitions in the movement (waypoint) sequence. A suffix array is a data structure for large texts, which does some precalculations at creation time to facilitate and speed up subsequent search operations. At its heart lies an array of all suffixes of the text in alphabetical order. For the word "banana", for example, the sorted array consists of the suffixes {"a", "ana", "anana", "banana", "na", "nana"}.

We extend the suffix array by also calculating the *longest common prefix* (LCP) table. The LCP table contains the longest common prefix that a suffix shares with the previous one in the array. In other words, the LCP table tells how many characters two subsequent entries have in common before they differ, starting at the beginning of the suffixes. An LCP of 5 means that the suffixes are equal in their first five characters and differ in the sixth one. As all suffixes are substrings starting at different positions in the original text, this means that the substring of length 5 starting at these positions is a repetition. In the example suffix array for the word banana, the LCP table for the entry "anana" would be 3.

In the case of our waypoint sequence, the LCP table allows us to quickly look up repeated subsequences. In particular, a high LCP means that a long part of the route was repeated. In a perfect world, a bot would always pass the same waypoints in the same order, making it easy to find the complete bot path from these repetitions. However, a bot may choose not to run the full path in one go, or it may not follow the path exactly enough to always pass all waypoints on its way. This case is actually the common case, with "perfect" runs being an exception. The processing of LCP values allows us to handle these inaccuracies and create a viable measure for subsequence repetitions.

3.7 Bot detection metrics

We propose two metrics for identifying bots: One takes into account *how often* a character travels through locations it visits on its route, by looking at the average path segment passes. The other one takes a look at the *amount and length of repeating subsequences* in the route by calculating the average LCP values of the waypoint sequence suffix array.

3.7.1 Decision based on average path segment passes

A *path segment* is a pair of adjacent waypoints in the movement sequence, a straight line from one waypoint until the character reached the next one. The number of times a path segment is passed shows how often a character has moved on a path segment, independent of the direction. We calculate the number of average path segment passes by dividing the total number of path segment passes by the number of distinct path segments.

It is trivial to conclude that, when the character keeps moving on the same path, this number rises for the segments on that path. As moving on the same path also means that no new waypoints are created (after all, the path is already described by a set of waypoints), the number of distinct path segments remains constant, while the number of total and average path segment passes rises steadily. This is the typical case for bots.

In the case of human players, on the other hand, new path segments are created continuously, because random movement creates new segments between old waypoints, or touches yet unexplored areas. Thus, the number of both distinct and total path segments keeps increasing. This leads to a low number of average path segment passes, a number that remains constant over the run of the game, usually well under 2 average passes per path segment.

By harnessing this difference, we can easily distinguish between bots and human players. While the bot game traces show steadily rising numbers as the game commences, the number for the human game traces settles down at a constant low level. See Section 4 for a detailed evaluation of this concept in *World of Warcraft*.

We chose average path segment passes over average waypoint passes, because we found average path segment passes to be a more robust indicator. Although both numbers describe the same principle and lead to similar graphs, path segment passes are significantly less likely to be repeated accidently. To pass a waypoint a second time, the character has to visit the same location a second time. To pass a path segment for the second time, the character has to be at the same location *and* move to the same waypoint next.

3.7.2 Decision based on repetitions in the waypoint sequence

Bots always take the same paths, and while they may not move through the exact same coordinates, their route still contains many and long repeating subsequences when compared to human routes. The LCP values from the waypoint sequence suffix array exactly captures this property. We calculate the average LCP value to ensure that the route needs to contain not only long but also *many* repetitions to display clearly suspicious numbers. To calculate the average LCP value, we sum up the LCP values of all entries in the suffix array and divide it by the number of entries. An average LCP of 5 means that for *every* waypoint in the waypoint sequence, the subsequence of the next 5 waypoints is repeated somewhere else. Humans typically never get close to the LCP values that bots reach, as it is very difficult to consistently keep passing the same waypoints in the same sequence. The average LCP for humans settles on a low level, while bot LCPs rise as they visit the same path over and over again. This provides a good discriminator between humans and bots. In the next section, we evaluate how well this approach fares in a real-world environment.

4 Implementation and evaluation

We chose *World of Warcraft*, today's by far most popular MMOG, to evaluate our approach. To this end, we set up a WoW-Server and collected game traces of two different bots and ten different human players. For the server, we modified the *ArcEmu* open source WoW server emulator [9] and logged all incoming and outgoing traffic and additional information about the game state in human-readable format to a text file.

4.1 Obtaining test game traces

We prepared template game characters at level 12 for four different character classes (warrior, paladin, mage, and warlock). For obtaining human test data, we instigated a LAN party for an evening (four hours), where seven people played *World of Warcraft* on the LAN with three more joining over the Internet. The probands ranged from regular WoW players to beginners that had never played the game before. During the test gaming session, the players were instructed to concentrate on farming, so their gaming style was likely to resemble that of bots.

To create bot game traces, we ran both the *Glider* [10] bot as well as the free *ZoloFighter* [11] bot, using the same template characters as the human players. We created two game traces per bot, using different bot paths each time. Glider is the most-commonly-used bot for WoW, sold on a subscription basis, while ZoloFighter is available for free. The company that develops Glider was recently sued by Blizzard [12], which clearly shows how seriously the botting threat is taken by the game provider.

4.2 The WOWalyzer

The obtained *game traces* were then fed to the *WOWalyzer*, a Java program we developed to display and analyze game traces, based on the methods described above. As expected, the movement of humans looked random, with very few waypoints or path segments that were passed more than once. On the other hand, bots quickly showed repeating movement patterns. Depending on the length of the set bot path and the number of enemies along the way, it took between 10 and 45 minutes until the beaten track was clearly visible from the movement graph.

4.2.1 Detection based on average line segment passes

Looking at the number of average line segment passes, the bot samples exhibited steadily increasing numbers as displayed in Figure 2, the slope depending on the length of the set bot path. The noticeable dip of the `ZoloFighter trace B - Bot` sample around time 1,800 owes its shape to the fact that a human player took over around packet 1,200. The goal was to purposely disrupt the bot path and evaluate the influence on our detection mechanism. As a result, the curve dropped as the character left the bot path. Later, around packet 1,800, the bot was taught a new path, which let the number of average line segment passes rise again. In sharp contrast, the human samples show steadily low numbers, settling down below 2.

4.2.2 Detection based on repetitions in the waypoint sequence

To measure repeated subsequences in the waypoint sequences, we calculated the average LCP of the suffix array created from the waypoint sequence. As shown in Figure 3, the bot samples show significantly higher values than the human samples, because they contain a lot more and also longer repeated subsequences. According to the graph, the human players did not move in repeated patterns, which kept the average LCP on a stable low level below 2, sometimes even below 1.

4.2.3 Accuracy of our approach

The test results show that our technique can reliably distinguish between bots and humans. In general, we see that after a short time, the numbers for bots and humans start to diverge, as bots begin repeating their movement pattern on their second run of a path. A bot alert is triggered whenever a certain threshold is reached. In our implementation, we set the threshold for both metrics to 5. This detects all bots within 12 to 60 minutes, while ensuring that no humans are falsely classified as bots. Once these values have been reached, the trend continues and the values never go back to normal as long as a bot is in control.

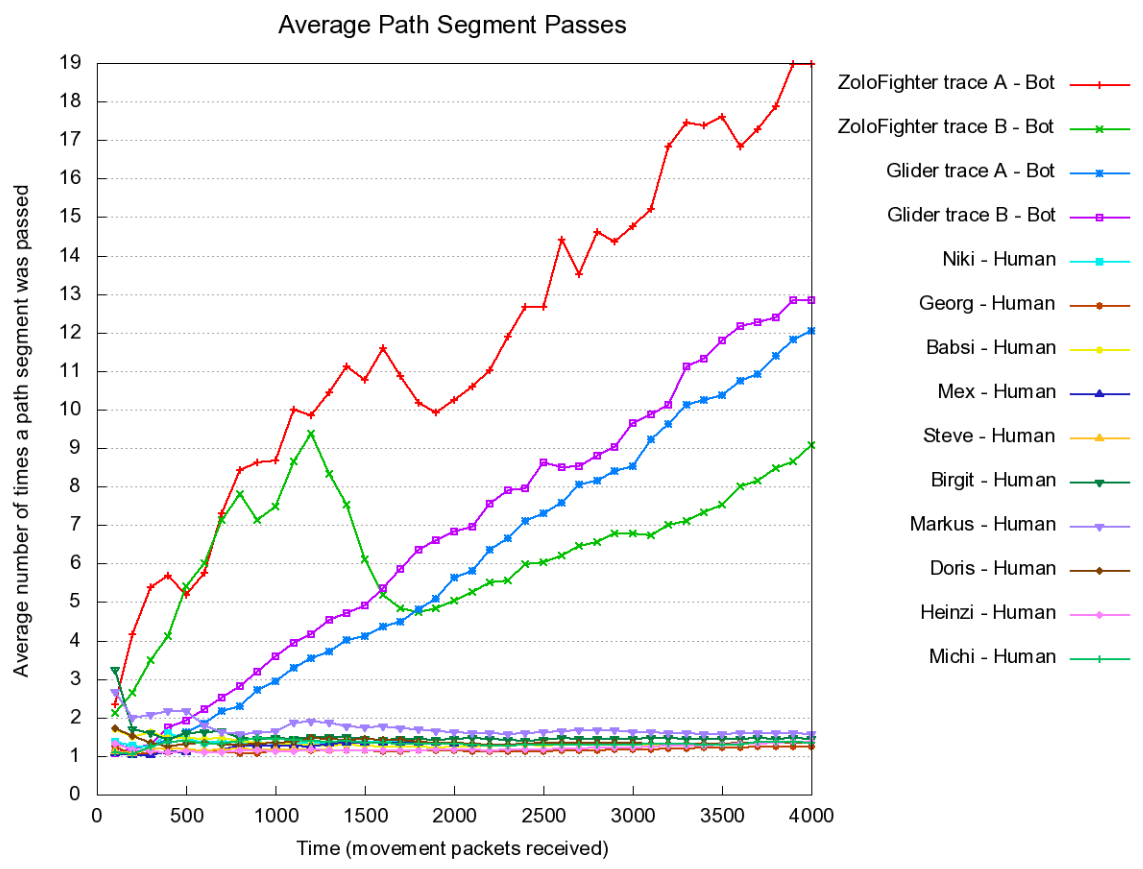

Figure 2: Number of average line passes.

4.3 Performance considerations

Our testing showed that it takes WOWalyzer less than 2 seconds to process 4 hours of gaming time on a single core Pentium-M with 1.6 GHz. However, we propose to use a sliding window of two hours to make the approach scale well, because time consumption of some processing steps rise at a quadratic rate. A shorter observation window also improves the approach's results for scenarios where a human player later switches over to a bot, in a sense that it becomes harder to ”hide” bot movements behind human activity.

4.4 Evading detection

The current system showcases the ideal behavior for server-side bot detection mechanisms, as it works completely transparently to the player. This makes it difficult for the player to discover what exactly triggered an account ban. However, discovery of the system would definitely lead to the bot programmers trying to implement countermeasures. While our preliminary tests have shown that the introduced system is able to distinguish reliably between bots and humans, it might still be possible for a bot to avoid detection if it is tailored specifically to the analysis mechanism of our approach.

In the case of our system, one evasion vector for bots would be to use very long paths, which makes the number of repetitions rise slower. However, those long paths need to be created by the player. Having the bot travel those paths only once means that the amount of time a player saves

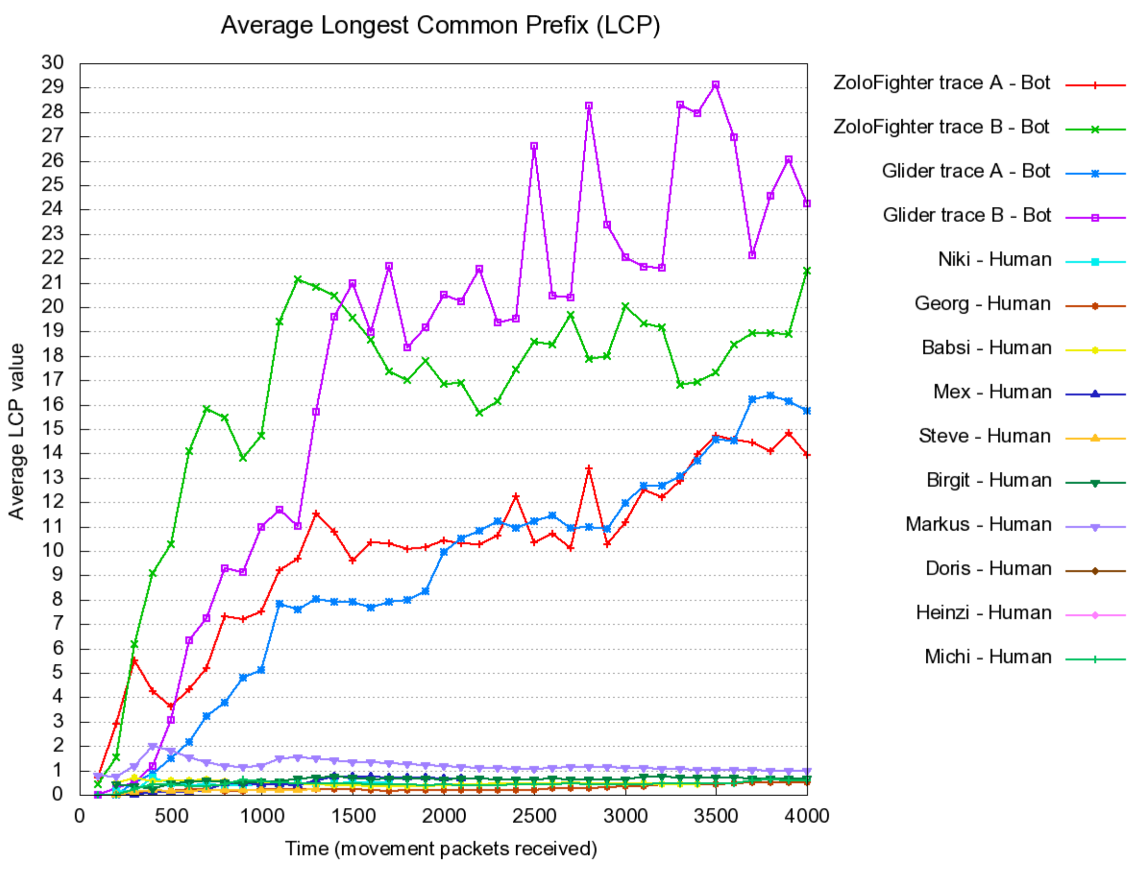

Figure 3: Average length of repeating waypoint subsequences (LCP).

by using a bot decreases drastically, which makes this method less attractive. Simply using paths shared on the internet increases the risk that those paths are also known to the game provider's bot detection system.

Another way for bots to make detection less likely would be to move the avatar more randomly than they do already. However, the crucial point remains that all current bots for *World of Warcraft* use a navigation system relying on waypoints, which inherently makes them susceptible to our and related approaches. Even though randomizing the movement along a path would impact the LCP values, the number of average line passes would hardly change. More general random movement would make the avatar susceptible to acting in a very dumb way again, arousing suspicion from human players.

As long as bots do not manage to replace their waypoint oriented navigation with a different paradigm, we predict that we can stay competitive in this arms race between bot programmers and game providers by continuously extending our method, for example with statistical heuristics.

5 Related work

Research on online gaming security has recently started to receive interest by the security research community. There has not been much work on the detection of bots in online gaming.

In [13], the authors concentrate on simple bots as they exist in online games such as poker. The authors propose the use of CAPTCHAs that automated scripts cannot solve. This method aims at preventing bots from playing the game automatically. However, the proposed approach for poker games is a strongly disruptive method, and has a heavy impact on the gaming experience for players. For example, many users may find it tedious and difficult to solve CAPTCHAs and may eventually even decide to quit playing the game. Our technique, in comparison, is intended for more complicated Massive Multiplayer Online Games. It is completely transparent to the end user and has no influence at all on the gaming experience. Moreover, we focus on actually *detecting* bots and not preventing them from running on the clients.

The most-closely-related work was introduced by Kuan-Ta Chen et al. [14]. In their paper, 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. Moreover, the authors attempt to distinguish traffic generated by the official game client from traffic generated by stand-alone bot programs through statistical analysis of packet transfer properties. Unfortunately, this approach no longer works for modern games, as these games mostly use ping-independent command queueing on the client-side (which changes network-level properties). In addition, the majority of MMOG bots work by interacting with the official game client (e.g., through code injection into the game process) and do not send any packets themselves. In contrast, our approach is aware of the semantics of a game, leveraging a character's behavior to identify bots. Thus, our technique is independent of traffic conditions and applicable to a wide range of MMOGs.

An important publication that is a good starting point for anyone interested in online game security is the recent book [15] by Greg Hoglund and Gary McGraw. It covers a wide section of game security topics, ranging from the legal issues over bug exploits and hacking game clients to writing bots. However, note that the book provides a good introduction into several gaming security areas, but concentrates mostly on the attacker's point of view and does not provide concrete solutions on how to detect or prevent botting.

6 Conclusion and future work

In this paper, we identify bots in Massive Multiplayer Online Games, exploiting the behavior they were made for: repetitive actions. Based on the fact that bots follow previously-recorded waypoints for moving in the virtual world, we are able to distinguish them from human players by only looking at the route they take. We extract waypoints from the character's movement data and look for repeated patterns in the route. Our approach identifies two key properties that make it easy to spot bots: how often a character travels through the locations of its route, and the frequency and length of repeating subroutes. This server-sided approach is completely transparent to the end users and has no influence on the gaming experience.

Our tests with *World of Warcraft* showed that our approach effectively distinguishes between bots and humans and is computationally fast enough to monitor large numbers of players concurrently. Hence, we believe that our technique constitutes an improvement over the sparse counter-measures currently in place in Massive Multiplayer Online Games.

To address the mentioned limitations and raise the bar for bot writers who design their bots to avoid detection, we intend to extend our system with statistical heuristics. Furthermore, we plan to test our approach on other MMOGs, such as Lineage and RuneScape.

References

[1] B. Woodcock, *An Analysis of MMOG Subscription Growth, Version 23.0*, April 2008, last accessed: 3. September 2008. [Online]. Available: http://www.mmogchart.com/charts/

[2] 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.

[3] Torak, *Bots: What *is* NCSoft doing?*, last accessed: 3. September 2008. [Online]. Available: http://l2vault.ign.com/View.php?view=Articles.Detail&id=15

[4] G. Hoglund, *4.5 million copies of EULA-compliant spyware*, October 2005, last accessed: 3. September 2008. [Online]. Available: http://www.rootkit.com/blog.php?newsid=358

[5] C. McSherry, *A New Gaming Feature?Spyware.* Electronic Frontier Foundation, October 2005, last accessed: 3. September 2008. [Online]. Available: http://www.eff.org/deeplinks/2005/10/new-gaming-feature-spyware

[6] D. H. Douglas and T. K. Peucker, "Algorithms for the Reduction of the Number of Points Required to Represent a Line or its Caricature," *The Canadian Cartographer*, vol. 10, no. 2, pp. 112–122, 1973.

[7] H.-F. Eckey, R. Kosfeld, and M. Rengers, *Multivariate Statistics.* Gabler, 9 2002.

[8] D. Gusfield, *Algorithms on strings, trees, and sequences: computer science and computational biology.* New York, NY, USA: Cambridge University Press, 1997.

[9] *ARCEmu World of Warcraft Server Emulator*, last accessed: 3. September 2008. [Online]. Available: http://arcemu.org/

[10] *Home of the Glider Bot for WoW*, last accessed: 3. September 2008. [Online]. Available: http://www.mmoglider.com/

[11] *WoW ZoloFighter Bot*, last accessed: 3. September 2008. [Online]. Available: http://www.zolohouse.com/wow/wowFighter/

[12] *Legal battle over Warcraft 'bot'*, March 2008, last accessed: 3. September 2008. [Online]. Available: http://news.bbc.co.uk/2/hi/technology/7314353.stm

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

[14] 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.

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