Battle of Botcraft: Fighting Bots in Online Games with Human Observational Proofs

Steven Gianvecchio, Zhenyu Wu, Mengjun Xie, and Haining Wang
Department of Computer Science
The College of William and Mary
Williamsburg, VA 23187, USA
{srgian, adamwu, mjxie, hnw}@cs.wm.edu

ABSTRACT

The abuse of online games by automated programs, known as game bots, for gaining unfair advantages has plagued millions of participating players with escalating severity in recent years. The current methods for distinguishing bots and humans are based on human interactive proofs (HIPs), such as CAPTCHAs. However, HIP-based approaches have inherent drawbacks. In particular, they are too obtrusive to be tolerated by human players in a gaming context. In this paper, we propose a non-interactive approach based on human observational proofs (HOPs) for continuous game bot detection. HOPs differentiate bots from human players by passively monitoring input actions that are difficult for current bots to perform in a human-like manner. We collect a series of user-input traces in one of the most popular online games, World of Warcraft. Based on the traces, we characterize the game playing behaviors of bots and humans. Then, we develop a HOP-based game bot defense system that analyzes user-input actions with a cascade-correlation neural network to distinguish bots from humans. The HOP system is effective in capturing current game bots, which raises the bar against game exploits and forces a determined adversary to build more complicated game bots for detection evasion in the future.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*

General Terms: Security

Keywords: Game Bots, Human Observational Proofs

1. INTRODUCTION

The online gaming market has experienced rapid growth for the past few years. In 2008, online gaming revenues were estimated at $7.6 billion world-wide [30]. The most profitable online games are subscription-based massive multiplayer online games (MMOGs), such as World of Warcraft. In 2008, World of Warcraft reached 11.5 million subscribers [7]. Each subscriber has to pay as much as $15 per month. It is no surprise that MMOGs make up about half of online gaming revenues [30]. As MMOGs gain in economic and social importance, it has become imperative to shield MMOGs from malicious exploits for the benefit of on-line game companies and players.

Currently the most common form of malicious exploit and the most difficult to thwart, is the use of game bots to gain unfair advantages. Game bots have plagued most of the popular MMOGs, including World of Warcraft [29,39,41,44,54], Second Life [37], and Ultima Online [18,46], and some non-MMOGs such as Diablo 2 [14]. The primary goal of game bots is to amass game currency, items, and experience. Interestingly, game currency can be traded for real currency[1], making cheating a profitable enterprise. Since MMOGs are small economies, a large influx of game currency causes hyper-inflation, hurting all players. Thus, the use of game bots is a serious problem for not only giving some players unfair advantages but also for creating large imbalances in game economies as a whole. With a large investment in development costs, game service providers consider anti-cheating mechanisms a high priority.

The existing methods for combating bots are not successful in the protection of on-line games. The approaches based on human interactive proofs (HIPs), such as CAPTCHAs, are the most commonly used to distinguish bots from humans. However, the inherent interactive requirement makes HIP-based approaches inadequate to apply in MMOGs. In particular, multiple tests are needed throughout a game session to block the login of bots; otherwise, a malicious player can pass the one-time test and log a bot into the game. Although multiple tests can foil the malicious player's attempt for bot login, they are too obtrusive and distractive for a regular player to tolerate as well. A different approach, taken by some game companies, makes use of a process monitor to scan for known bot or cheat programs running on a player's computer. Blizzard, the makers of World of Warcraft, developed such a system called the Warden that scans processes and sends information back to their servers. A number of similar anti-cheat systems have been built for other games [16,17,38,49]. However, this scan-based approach has proven ineffective, and even worse, raises privacy concerns. The Electronic Frontier Foundation views the Warden as spyware [28]. Besides technical approaches, Blizzard has pursued legal action against bot makers [3], claiming over $1

[1]The exchange rate for World of Warcraft is 1,000 gold to $11.70 as of July 25th, 2009 [48].

CCS'09, November 9–13, 2009, Chicago, Illinois, USA.

million per year in additional operating costs caused by game bots in their lawsuit [6]. Moreover, Blizzard has banned thousands of accounts for cheating [8], yet many players continue cheating via bots and slip through the cracks [39, 41].

In this paper, we introduce an approach based on human observational proofs (HOPs) to capture game bots. HOPs offer two distinct advantages over HIPs. First, HOPs provide continuous monitoring throughout a session. Second, HOPs are non-interactive, i.e., no test is presented to a player, making HOPs completely non-obtrusive. The use of HOPs is mainly motivated by the problems faced by HIPs and methods used in behavioral biometric systems [1, 20, 40, 43]. Similar behavior-based approaches have been used in many previous intrusion detection systems [21, 23, 26, 42, 51]. We collect a series of user-input measurements from a popular MMOG, World of Warcraft, to study the behaviors of current game bots and humans. While human players visually recognize objects on the screen and physically control the mouse and keyboard, game bots synthetically generate mouse and keyboard events and cannot directly recognize most objects. Our measurement results clearly show the fundamental differences between current game bots and humans in how certain tasks are performed in the game. Passively observing these differences, HOPs provide an effective way to detect current game bots.

Based on HOPs, we design and develop a game bot defense system that analyzes user-input data to differentiate game bots from human players in a timely manner. The proposed HOP system consists of two major components: a client-side exporter and a server-side analyzer. The exporter is responsible for sending a stream of user-input actions to the server. The analyzer then processes the user-input stream and decides whether the client is operated by a bot or a human. The core of the analyzer is a cascade neural network that "learns" the behaviors of normal human players, as neural networks are known to perform well with user-input data [1, 35, 36]. Note that the latest MMOGs virtually all support automatic updates, so the deployment of the client-side exporter is not an issue. Moreover, the overhead at the client side is negligible and the overhead at the server side is small and affordable in terms of CPU and memory consumptions even with thousands of players per server. To validate the efficacy of our defense system, we conduct experiments based on user-input traces of bots and humans. The HOP system is able to capture 99.80% of current game bots for World of Warcraft within 39.60 seconds on average.

It is an arms race between game exploits and their countermeasures. Once highly motivated bot developers know the HOP approach, it is possible for them to create more advanced game bots to evade the HOP system. However, the purpose of the HOP system is to raise the bar against game exploits and force a determined bot developer to spend significant time and effort in building next-generation game bots for detection evasion. Note that, to operate the game in a human-like manner, game bots have to process complex visuals and model different aspects of human-computer interaction and behavior, which we believe is non-trivial to succeed.

The remainder of the paper is organized as follows. Section 2 describes the background of game bots and game playing behaviors. Section 3 presents the measurements and analyses of game playing inputs from human players and game bots, respectively. Section 4 details the proposed HOP system. Section 5 evaluates the effectiveness of our HOP system for detecting game bots. Section 6 discusses the limitations of this work. Section 7 surveys related work. Section 8 concludes the paper.

2. BACKGROUND

In this section, we first briefly present the evolution of game bots. Then, we describe the game playing behaviors of human players and game bots, respectively, and highlight their differences in a qualitative way.

2.1 Game Bots

A variety of exploits have appeared in the virtual game world for fun, for win, and for profit. Among these game exploits, game bots are regarded as the most commonly-used and difficult-to-handle exploit. The earliest game bots were developed for the first generation MMOGs such as Ultima Online [46]. Even at that time, bot operators were already quite sophisticated, creating small server farms to run their bots [18, 46]. At the early era of game bots, most of bot programmers wrote their own game clients. However, as a countermeasure, game companies often update games, breaking operations of those custom game clients. Bot programmers were forced to update their game clients, keeping up with the latest game version. This cycle proves to be very tedious for game bot programmers. Moreover, the complexity of game clients has grown continuously, making it increasingly difficult to develop and maintain a standalone custom game client.

The arms race between game vendor and bot developer has led to the birth of an interesting type of game bots that, much like humans, play games by reading from screen and using the mouse and keyboard. These advanced bots operate the standard game client by simply sending mouse and keyboard events, reading certain pixels from the screen, and possibly reading a few key regions in the memory address space of the game application. Most bots are equipped with macro scripting capabilities, similar to programs like AutoIt [4], which enables bots to be easily reprogrammed and quickly adapted to the changes made by game companies.

2.2 Game Playing Behaviors

MMOGs, such as World of Warcraft, entertain players by providing a large degree of freedom in terms of actions a player can perform. In the game world, a player controls a virtual character (avatar) to explore the landscape, fight monsters, complete quests and interact with other players. In addition, a player can further customize the character by learning skills and purchasing items (such as armor, weapons, and even pets) with virtual currency. Each game activity requires a player to interact with the game in a different fashion. As a result, it is expected that the inputs of a human player will exhibit burstiness with strong locality and the input contents vary significantly for different tasks through game play. However, when a bot is used to play the game, its main purpose is to gain rewards (level and virtual currency) without human intervention by automating and repeating simple actions (such as killing monsters). Being much less sophisticated than human, bot actions would show regular patterns and limited varieties.

Besides the high-level behavioral differences, humans and bots also interact with the game very differently, despite that both interact with the game via mouse and keyboard.

As biological entities, humans perceive the graphical output of the game optically, and feed input to the game by physically operating devices such as keyboard and mouse. In contrast, bots are computer programs that have no concept of vision and are not bounded by mechanical physics. While bots can analyze game graphics, it is computationally expensive. To avoid this computation cost, whenever possible, bots attempt to obtain necessary information, such as the locations of the avatar, monsters and other characters, and the properties (health, level, etc.) of the avatar, by reading the memory of the game program.

In general, bots control the avatar by simulating input from devices via OS API calls, such as setting key press state or repositioning mouse cursor. The techniques used by bots are often crude, but in most cases, quite effective. For example, without reading the graphics or scanning the terrain, a bot can navigate to a target location by knowing just two coordinates—the current location of the avatar and that of the target. The bot then tries to approach the target location by steering the avatar to go forward, left and right, and then checks its progress by polling the two coordinates. If the avatar location does not change in a given amount of time, the bot assumes that an obstacle (trees, fences, steep terrain, etc.) is in the way and tries to navigate around it by moving backward a few steps, turning left or right, and going forward. Occasionally, graphics analysis can be useful, such as when picking up items on the ground. The bot can again handle this situation in a simple and efficient manner by exploiting the game user interface. When the cursor is placed on top of an object, the game would display a small information window on the lower-right corner. Thus, the bot moves the mouse cursor in grid patterns, and relies on the change of pixel colors on the lower-right corner of the screen to know if it has found the object.

3. GAME PLAYING CHARACTERIZATION

In this section, we examine how bots and humans behave in the game, in order to have a deep understanding of the differences between humans and bots. Based on our game measurements, we quantitatively characterize the game playing behaviors of human players and bots, respectively. The behavioral differences between bots and humans form the basis for our HOP-based system.

3.1 The Glider Bot

We select the Glider bot [29] as the sample game bot for our research. The Glider bot is a very popular game bot for World of Warcraft. It runs concurrently with the game client, but requires system administrator privileges. This escalated privilege helps the Glider bot to circumvent the Warden anti-bot system, and enables it to access the internal information of the game client via cross-process-address-space reading. It operates by using a "profile"—a set of configurations including several waypoints (map coordinates in the game world) and options, such as levels of monsters to fight. When in operation, the game bot controls the avatar to repeatedly run between the given waypoints, search and fight monsters that match the given criteria, and collect bonus items after winning fights.

3.2 Input Data Collection

We collect player input data for both human and bot using an external program in a non-intrusive manner, i.e., no modification to the game client program. The input data collection program, a modified version of RUI [27], runs concurrently with the game, polling and recording the keyboard and mouse input device status with clock resolution close to 0.015625 second (approximate 64 times/sec). Each input event, such as key press or cursor position change, is recorded along with a time stamp relative to the starting time of the recording.

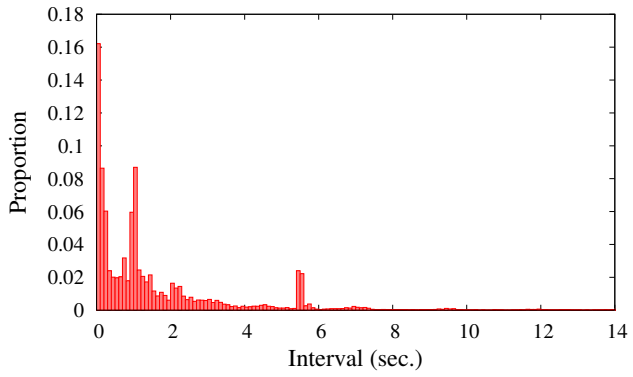
Table 1: Definitions of User-Input Actions

Action	Definition
Keystroke	The press and release of a key.
Point	A series of continuous mouse cursor position changes with no mouse button pressed; the time-stamps for each pair of cursor position changes are no more than 0.4 seconds apart.
Pause	A period of 0.4 seconds or longer with no actions.
Click	The press and release of a mouse button; the cursor travels no more than 10 pixels between the press and release.
Point-and-Click	A point followed by a click within 0.4 seconds.
Drag-and-Drop	The press and release of a mouse button; the cursor travels more than 10 pixels between the press and release.

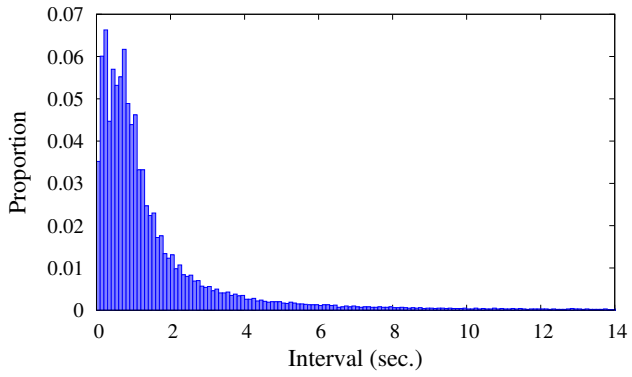
We invite 30 different human players to play World of Warcraft and collect 55 hours of their user-input traces. The players are a group of 25 men and 5 women with different ages and different levels of gaming experience. The players are mostly college-aged, i.e., undergraduate and graduate students, with 9 players from *18-24* years of age, 17 from *25-34*, 3 from *35-44*, and 1 *over 45*. The players' levels of computer gaming experience (described as "regular play") are 6 players with *none*, 2 with *less than 1 year*, 6 with *2 to 5 years*, 7 with *5 to 10 years*, and 9 with *more than 10 years*.

While the players are allowed to play from their home computers, most players, 27 out of 30, play in the lab. The players are free to select their characters (existing or new) and their characters' classes, items, skills, and so on. The players are encouraged to farm, i.e., kill monsters and gather treasures, but some instead explore or do quests. Most players, 20 out of 30, play as tank or physical-damage classes, e.g., warrior, rogue, and hunter, while a few players, 10 out of 30, play as magic-damage or healing classes, e.g., mage, warlock, druid, and priest. The human characters range from level 1 to mid-30s in the traces, with most characters, 23 out of 30, under level 10. The few higher level characters, 7 out of 10, in the 20s and 30s are existing characters and not new ones starting from level 1.

Correspondingly, we run the game bot with 10 different profiles in 7 locations in the game world for 40 hours and collect its input traces. The 10 profiles are bot configurations with different sets of waypoints that the bot follows while farming, i.e., killing monsters and gathering treasure. The profiles are setup in 7 locations with different monster levels (from levels 1 to 40), monster densities (sparse to dense),

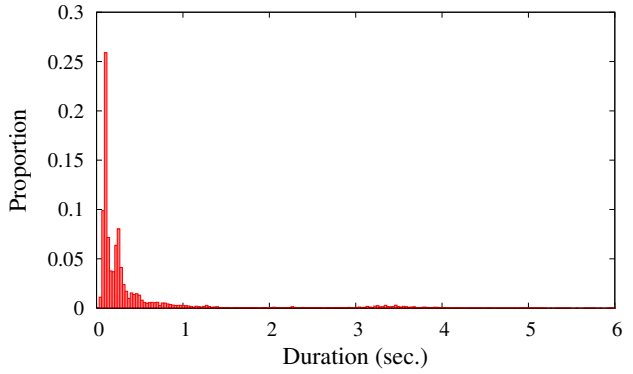


(a) Bot

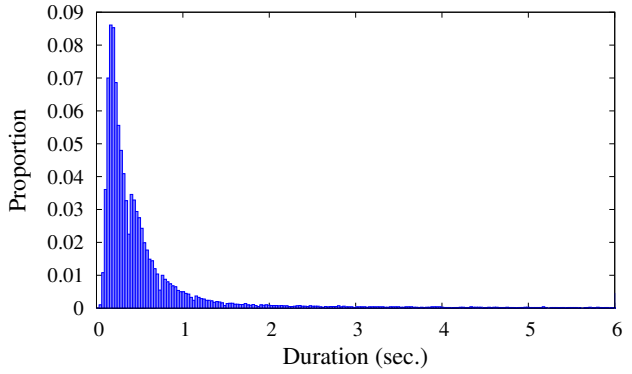


(b) Human

Figure 1: Keystroke Inter-arrival Time Distribution



(a) Bot



(b) Human

Figure 2: Keystroke Duration Distribution

and different obstacles (barren plains to forest with lots of small trees). The game bot profiles are half run with a warrior and half run with a mage. These two bot characters range from level 1 to over 30 in the traces.

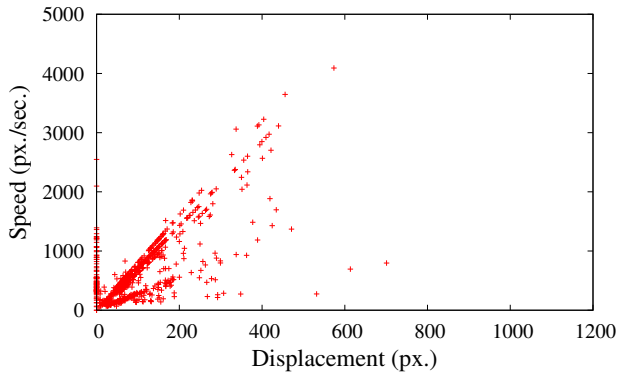
We conduct post processing on the input trace data to extract information with regard to high-level user-input actions. For example, we pair up a key press event with a subsequent key release event of the same key to form a `keystroke` action; we gather a continuous sequence of cursor position change events to form a `point` action (mouse movement action). Table 1 gives a complete list of high level actions we derive and their corresponding definitions.

3.3 Game Playing Input Analysis

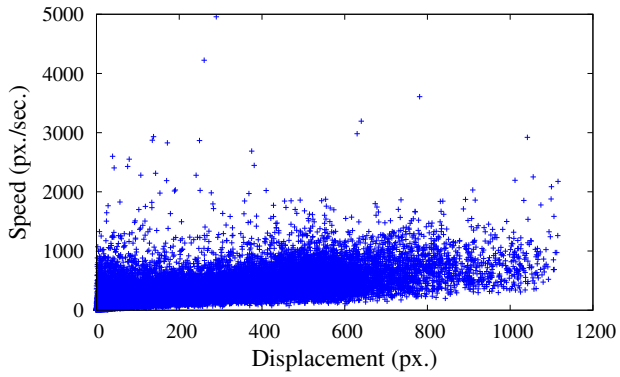
We analyze the Glider bot and human keyboard and mouse input traces with respect to timing patterns (duration and inter-arrival time) and kinematics (distance, displacement, and velocity). Our bot analysis below is limited to the current game bots.

Two keyboard usage metrics for human and bot are presented in Figures 1 and 2, respectively. Both figures are clipped for better presentation, and the trailing data clipped away contribute less than 3% of the total for either human or bot. Figure 1 shows the distribution of `keystroke` inter-arrival time, i.e., the interval between two consecutive key presses, with a bin resolution of 0.1 seconds. There are two major differences between the bots and humans. First, the bot issues keystrokes significantly faster than humans. While 16.2% of consecutive keystrokes by the bot are less than 0.1 second apart, only 3.2% of human keystrokes are that fast. This is because human players have to initiate keystroke action by physical movement of fingers, and hence, pressing keys at such high frequency would be very tiring. Second, the keystrokes of the bot exhibit obvious periodic patterns. The empirical probabilities of the bot pressing a key every 1 or 5.5 seconds are significantly higher than their neighbor intervals, which provides us some insights into the internals of the bot: it uses periodic timers to poll the status of the avatar (i.e., current coordinate), and issue keyboard commands accordingly (e.g., bypass possible obstacles by turning left/right and jumping). However, for human players, their keystroke intervals follow a Pareto distribution, which matches the conclusions of previous research [53]. Figure 2 shows the distribution of `keystroke` durations, with the bin resolution of 0.03 second. These figures reassures our previous observations: the bot presses keys with much shorter duration—over 36.9% of keystrokes are less than 0.12 seconds long, while only 3.9% of human keystrokes are completed within such a duration; the bot exhibits the periodic keyboard usage pattern—keystrokes with around 0.25 second duration are significantly more than its neighbor durations.

Figure 3 shows the relationship between the mouse speed and the displacement between the origin and target coordinates for the `point-and-click`. Less than 0.1% of the total

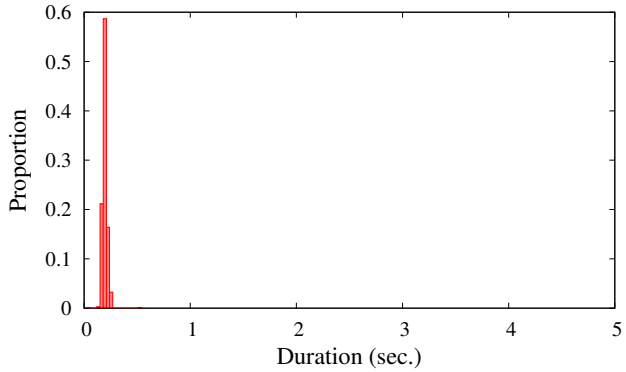


(a) Bot

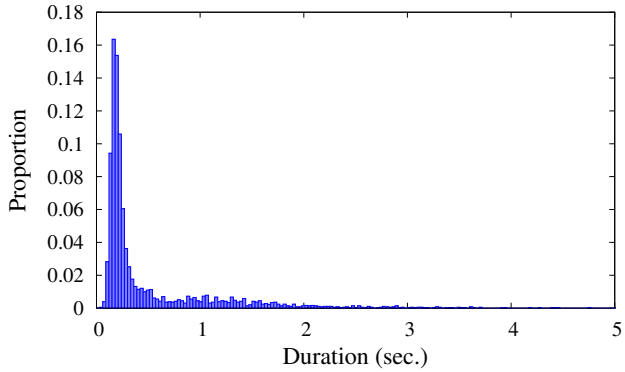


(b) Human

Figure 3: Average Speed vs. Displacement for Point-and-Click



(a) Bot



(b) Human

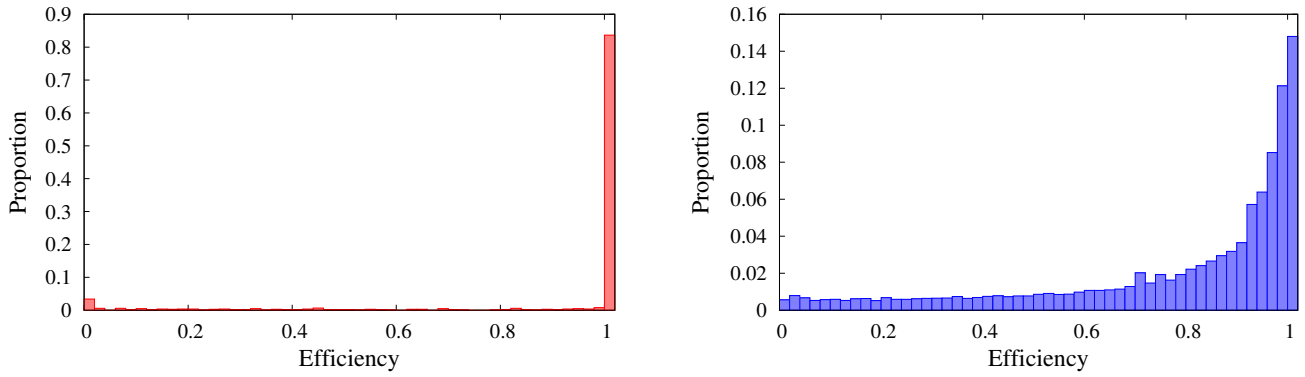
Figure 4: Drag-and-Drop Duration Distribution

data points for either human or bot are clipped away. The bots exhibit two very unique features. First, unlike human players, who move the mouse with very dynamic speed at all displacement lengths, the bots tend to move the mouse at several fixed speeds for each displacement, and the speed increases linearly as displacement lengthens. This feature implies that, again, the bots use several fixed length timers for mouse movements. Second, we also observe that the bots make a significant amount of high speed moves with zero displacement, that is, after a series of fast movements, the cursor is placed back exactly at its origin. Such a behavior is absent in the human data, because it is physically difficult and unnecessary.

Figure 4 shows the distribution of mouse `drag-and-drop` duration, with the bin resolution of 0.03 second. For the bots, 100% of actions are accomplished within 0.3 second. However, for human players, only 56.6% of `drag-and-drop` actions finish within the same time window; over-one-second actions contribute 25.5% of the total, within which, about 0.8% of actions are more than 5 seconds long, and are thus clipped away from the figure.

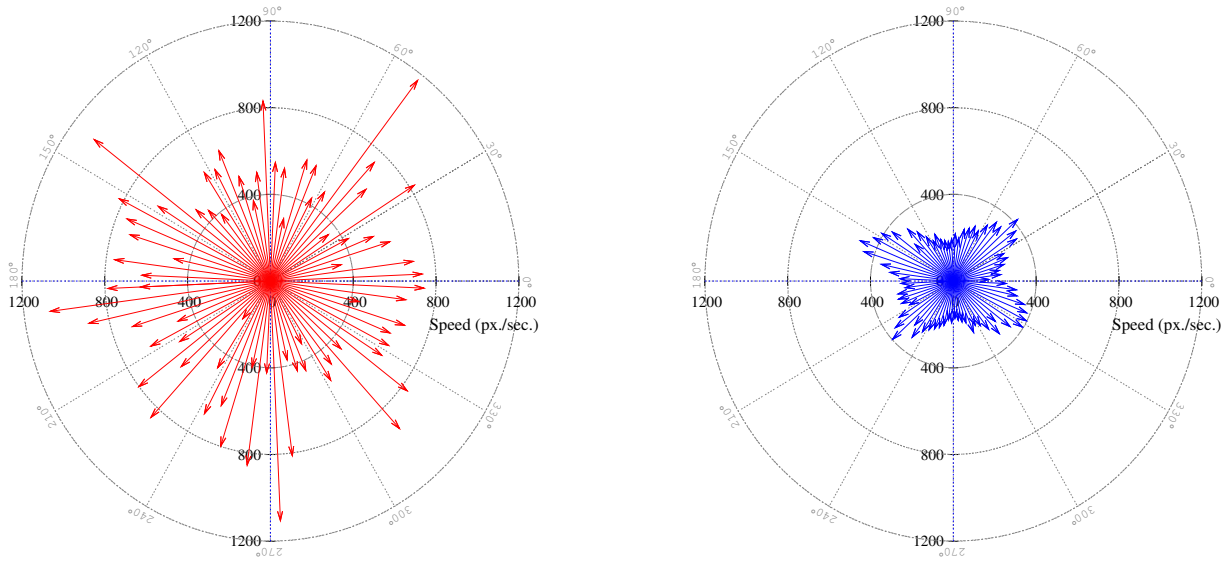
Figure 5 illustrates the distribution of mouse movement efficiency for `point-and-click` and `drag-and-drop`. We define *movement efficiency* as the ratio between the cursor displacement and the traversed distance over a series of movements. In other words, the closer the cursor movement is to a straight line between the origin and target coordinates, the higher the movement efficiency. Note that, while the bin width is 0.02, the last bin only contains the actions with efficiency of 1.0. Bots exhibit significant deviation from human players on this metric: 81.7% of bot mouse movements have perfect efficiency, compared to that only 14.1% of human mouse movements are equally efficient. Aside from 3.8% of mouse movements with efficiency less than 0.02 (most of which are zero efficiency moves, due to the cursor being placed back to the origin), a bot rarely moves the mouse with other efficiencies. However, for human players, the observed probability of mouse movement efficiency follows an exponential distribution.

Finally, Figure 6 presents the relationship between the average mouse move speed and the direction of the target coordinate, plotted in polar coordinate with angular resolution of 10 degrees ($\pi/36$). Each arrow represents the average velocity vector of mouse movements whose target position is ± 5 degrees in its direction. For the bots, there is no evident correlation between the speed and the direction. In contrast, for human players, there is a clear *diagonal, symmetric, and bounded* movement pattern: diagonal movements are generally faster than horizontal and vertical movements, upward movements are slightly faster than downward movements, and leftward movements are slightly faster than rightward movements; overall, the movement speed is bounded to a certain value. The diagonal and symmetric pattern is attributed to the human hand physiology, and the speed boundary is due to the physical constraint of human arms.



(a) Bot (b) Human

Figure 5: Point-and-Click and Drag-and-Drop Movement Efficiency Distribution



(a) Bot (b) Human

Figure 6: Average Velocity for Point-and-Click

4. HOP SYSTEM

In this section, we describe the design of our proposed HOP system. The HOP system consists of client-side exporters and a server-side analyzer. Each client-side exporter collects and sends a stream of user-input actions taken at a game client to the game server. The server-side analyzer then processes each input stream and decides whether the corresponding client is operated by a bot or a human player. Figure 7 illustrates the high-level structure of the HOP system.

4.1 Client-Side Exporter

Since each game client already receives raw user-input events, the client-side exporter simply uses the available information to derive input actions, i.e., `keystroke`, `point`, `click`, and `drag-and-drop`, and sends them back to the server along with regular game-related data. Ideally, the client-side exporter should be implemented as an integral part of the game executable or existing anti-cheat systems [16, 17, 38, 49]. For the prototype of our HOP system, we implement it as a standalone external program, as we do not have source code access to the World of Warcraft.

4.2 Server-Side Analyzer

The server-side analyzer is composed of two major components: the user-input action classifier and the decision maker. The work-flow of the server-side analyzer is as follows. For each user-input action stream, the system first stores consecutive actions into the action accumulator. A configurable number of actions form an action block, and each action block is then processed by the classifier. The output of the classifier contains the classification score for the corresponding action block, i.e., how close the group of actions look to those of a bot, and is stored into the output accumulator. Finally, when the output accumulator aggregates a configurable amount of neural network output, the decision maker makes a judgment. Each judgment reflects whether the player is possibly operated by a bot since the last judgment. The output accumulator is refreshed after each decision is made. The analyzer continuously processes user-input actions throughout each user's game session.

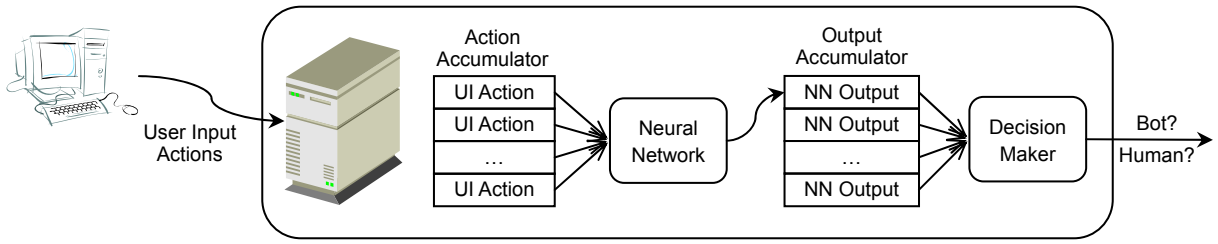


Figure 7: Overview of the HOP system

4.2.1 Neural Network Classification

We employ artificial neural networks for user-input action classification due to the following two reasons. First, neural networks are especially appropriate for solving pattern recognition and classification problem involving a large number of parameters with complex inter-dependencies. The effectiveness of neural networks with user-input data classification has already been demonstrated in behavioral biometric identification systems [1,35,36]. Second, neural networks are not simple functions of their inputs and outputs. While the detection methods based solely on those metrics with clearly defined equations are susceptible to inverse function attacks, neural networks, often described as a "black box", are more difficult to attack. Note that our HOP system is not necessarily tied to neural networks, and we will consider other classification methods, such as support vector machines (SVMs) or decision trees, in our future work.

The neural network we build for the HOP system is a cascade-correlation neural network, a variant of feed-forward neural networks that use the idea of cascade training [19]. Unlike standard multi-layer back-propagation (BP) perceptron networks, a cascade correlation neural network does not have a fixed topology, but rather is built from the ground up. Initially, the neural network only consists of the inputs directly connected to the output neuron. During the training of the neural network, a group of neurons are created and trained separately, and the best one is inserted into the network. The training process continues to include new neurons into the network, until the neural network reaches its training target or the size of the network reaches a pre-defined limit.

Figure 8 illustrates the general construction of the cascade-correlation neural network. There are eight input values for each user-input action, including seven action metric parameters and a bias value that is used to differentiate the type of action, e.g., keyboard action or mouse action. The neural network takes input from all actions in an action block. The connections between the input node and neurons, and among neurons, are represented by intersections between a horizontal line and a vertical line. The weight of each connection is shown as a square over the intersection, where larger size indicates heavier weight.

The seven action metric parameters are: action duration, mouse travel distance, displacement, efficiency, speed, angle of displacement, and virtual key (a numeric value corresponding to a keyboard key or a mouse button). The speed and efficiency are derived parameters from the basic parameters, such as duration, distance and displacement. These derived parameters are used mainly to help the neural network capture the inherent association between input parameters, reduce the network complexity, and thus, speedup the

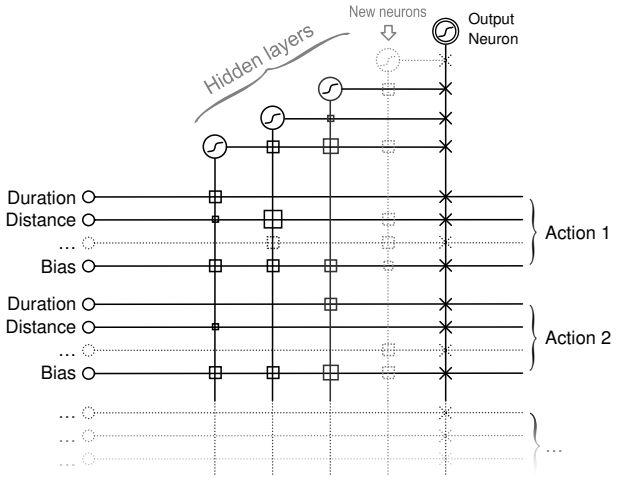


Figure 8: A Cascade Neural Network

training process. The number of actions in an action block directly affects the total amount of input data to the neural network. Increasing the block size provides the neural network with more context information and can, up to a certain point, further improve the classification accuracy of the trained network. However, too many input actions can also increase the overall complexity of the neural network and slow down the training process.

4.2.2 Decision Making

The decision maker refers to using accumulated output from the neural network to determine whether the corresponding user-input data is likely from a bot or a human player. Different algorithms can be applied to consolidate accumulated classifications. We employ a simple "voting" scheme: if the majority of the neural network output classifies the user-input actions as those of a bot, the decision will be that the game is operated by a bot, and vice versa. The decision process is a summary of the classifications of user-input actions over a period of time. While individual classification cannot be 100% correct, the more accumulated output, the more confidence we have in the decision. On the other hand, the more accumulated output, the more user-input actions are required, which translates to more data storage and longer time for decision making.

4.3 Performance Impact and Scalability

The nature of MMOGs dictates our design of the HOP system to be scalable and light-weight, limiting performance impacts on game clients and the server. At the client side,

the system resource consumed by the collection of user-input actions is minor. In addition to the system resource of a game client, an MMOG player's gaming experience also depends on network performance. Since the user-input actions are short messages, 16 bytes of data per user-input action, the additional bandwidth consumption induced by the client-side exporter is negligible. The presence of the exporter thus is imperceptible for end users. At the server side, the scalability is critical to the success of our HOP system. The server-side analyzer is very efficient in terms of memory and CPU usage, which is shown in Section 5.4. The size of additional memory consumed per player is comparable to the size of the player's avatar name. A single processor core is capable of processing tens of thousands of users simultaneously in real-time. Therefore, the HOP system is scalable to the heavy workload at a game server.

5. EXPERIMENTS

In this section, we evaluate the efficacy of our HOP system through a series of experiments, in terms of detection accuracy, detection speed, and system overhead. The metrics we use for detection accuracy include true positive rate and true negative rate. The true positive rate is the percentage of bots that are correctly identified, while the true negative rate is the percentage of humans that are correctly identified. The detection speed is determined by the total number of actions needed to make decisions and the average time cost per action. In general, the larger the number of actions required for decisions and the higher the average time cost per action, the slower the detection speed becomes.

5.1 Experimental Setup

Our experiments are based on 95 hours of traces, including 55 hours of human traces and 40 hours of game bot traces. In total, these traces contain 3,000,066 raw user-input events and 286,626 user-input actions, with 10 bot instances and 30 humans involved. The 10 bot instances are generated by running the Glider bot with 10 different profiles. The human players are a diverse group, including men and women with different ages and different levels of gaming experience. The more detailed trace information has been given in Section 3.2.

The experiments are conducted using 10-fold cross validation. Each test is performed on a different human or bot that is left out of the training set for that test. Therefore, to validate a given configuration, 20 different partitions are created, one for each of the 10 bots and 10 sets of three humans. The partitions consist of a training set of either 9 bots and 30 humans or 10 bots and 27 humans, and a test set of either one bot or three humans. Thus, each test is performed on unknown data that the system has not yet been trained on.

5.2 Detection Results

The HOP system has four configurable parameters: the number of actions per block, the number of nodes, the threshold, and the number of outputs per output block. The first two parameters mainly determine the size and complexity of the neural network, while the second two parameters largely affect the detection performance of the entire system. The threshold determines how a neural network output is interpreted: a value over the threshold indicates a bot, while a value under the threshold indicates a human. Note that humans have a value of 0.0 and bots have a value of 1.0 in the training of the neural network.

We first configure the number of actions per block and the number of nodes. The true positive and true negative rates with different numbers of actions and different numbers of nodes are shown in Figure 9 (a) and (b), respectively. These tests are performed with a default threshold of 0.5. The neural network becomes more accurate as more actions are provided, but we see diminishing returns in accuracy as the number of actions increases, e.g., going from 4 actions to 6 actions requires 50% more input but only provides a relatively small increase in the overall accuracy.

In most cases, the binomial theorem predicts that combining three decisions for the 4-action neural network should be more accurate than combining two decisions for the 6- or 8-action neural networks. Therefore, we choose to use a neural network with 4 actions as input, which gives true positive and negative rates of 0.971-0.977 and 0.959-0.973, respectively.

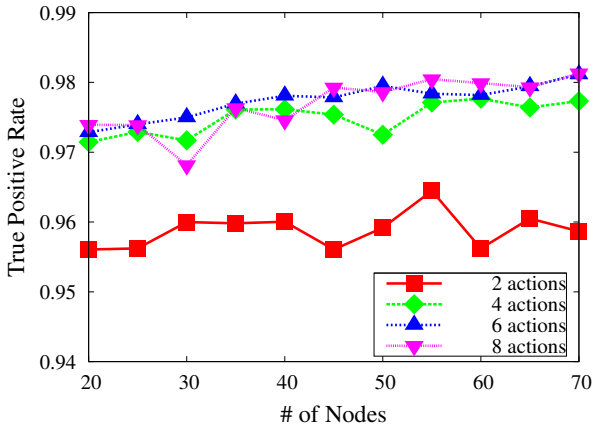
The overall true positive and negative rates do not always grow as the number of nodes increases. At some points, increasing the number of nodes no longer improves the true positive or negative rates and the neural network starts to over-fit the training set. A neural network of 40 nodes provides a true positive rate of 0.976 and a true negative rate of 0.961, which is the best combination of true positive and true negative rates with 4 actions as input. Therefore, we set up the neural network based on this configuration.

With the neural network configured, the threshold and the number of outputs per block determine the overall performance of the system. The threshold can be increased or decreased from the default value of 0.5 to bias the neural network towards bots or humans, improving the true positive rate or the true negative rate, respectively. The number of outputs per block affects both the detection accuracy and the detection speed of the system. As the number of outputs per block increases, the detection accuracy of the system increases, but the detection speed decreases as more neural network outputs are needed to make decisions.

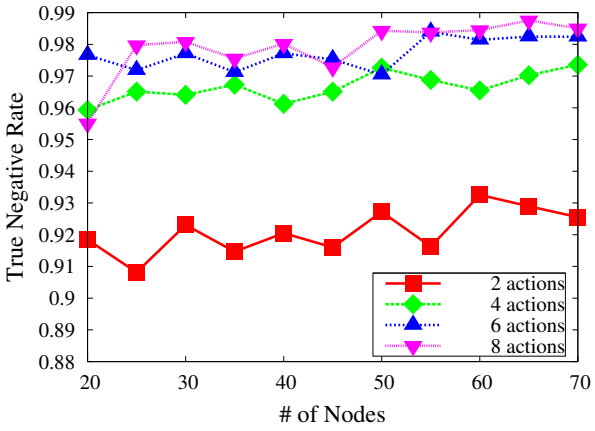
The true positive and negative rates with different thresholds and different numbers of outputs for bots and humans are listed in Table 2. The top number in each cell is the true positive rate and the bottom number is the true negative rate rate. The neural network has 40 nodes and takes 4 actions as input. There are a number of settings that allow for a true positive or true negative rate of 1.0, though not both. To avoid a false positive—mistaking a human for a bot, we prefer a high true negative rate. The smallest number of outputs per block that achieves a true negative rate of 1.0 is 9 outputs per block with the threshold of 0.75, which gives a true positive rate of 0.998.

With the fully configured system (40 nodes, 4-action input, the threshold of 0.75, and 9 outputs per block), Table 3 lists the true positive and negative rates for each of the individual bots in our traces. The true negative rates are 1.0 for all of the humans, so none of the human players in our traces are misclassified as bots. The true positive rates are between 0.988 and 1.000 for the bots in our traces, with the average true positive rate of 0.998.

The detection speed of the system is a function of the total number of actions required for decision making and the average time cost per action. The total number of actions is 36 (i.e., 9 outputs $\times$ 4 actions per output). The time cost per



(a) True Positive Rate



(b) True Negative Rate

Figure 9: True Positive and Negative Rates versus # of Accumulated Actions and # of Nodes

Table 2: True Positive and Negative Rates versus Thresholds and # of Accumulated Outputs

Threshold	# of Accumulated Outputs										
	1	3	5	7	9	11	13	15	17	19	21
0.25	0.978	0.995	0.997	0.999	0.999	1.000	1.000	1.000	1.000	1.000	1.000
	0.959	0.977	0.983	0.987	0.990	0.992	0.994	0.994	0.996	0.996	0.997
0.5	0.961	0.991	0.997	0.998	0.999	1.000	1.000	1.000	1.000	1.000	1.000
	0.976	0.990	0.994	0.996	0.998	0.998	0.998	0.998	0.999	0.998	0.999
0.75	0.926	0.980	0.992	0.997	0.998	0.998	0.998	1.000	1.000	1.000	1.000
	0.985	0.995	0.997	0.998	1.000	0.999	0.999	1.000	1.000	0.999	1.000
0.9	0.859	0.935	0.964	0.980	0.985	0.996	0.995	0.996	0.995	0.998	0.998
	0.991	0.998	0.999	0.999	1.000	0.999	0.999	1.000	1.000	0.999	1.000
0.95	0.775	0.856	0.895	0.922	0.940	0.947	0.958	0.969	0.976	0.975	0.983
	0.994	0.999	0.999	0.999	1.000	0.999	0.999	1.000	1.000	0.999	1.000
0.975	0.624	0.668	0.700	0.723	0.737	0.757	0.770	0.776	0.792	0.796	0.804
	0.996	0.999	0.999	0.999	1.000	0.999	0.999	1.000	1.000	0.999	1.000

action varies. The average time cost per action, ignoring idle periods longer than 10 seconds, is 1.10 seconds. If a player is idle, strictly speaking, no one is "operating" the game, so no decision can be made. Of course, idle players (bots or humans) are not performing any actions and should not be a concern. Based on the total number of actions and the average time cost per action, Figure 10 illustrates the decision time distribution for bots and humans. From the decision time distribution, we can see that our HOP system is able to make decisions for capturing bots within 39.60 seconds on average.

Note that we perform the same experiments with BP neural networks and observe that the cascade neural network is more accurate in bot classification than BP neural networks that use incremental, quick propagation, or resilient propagation method. The results for BP neural networks are not included in the paper due to space limit.

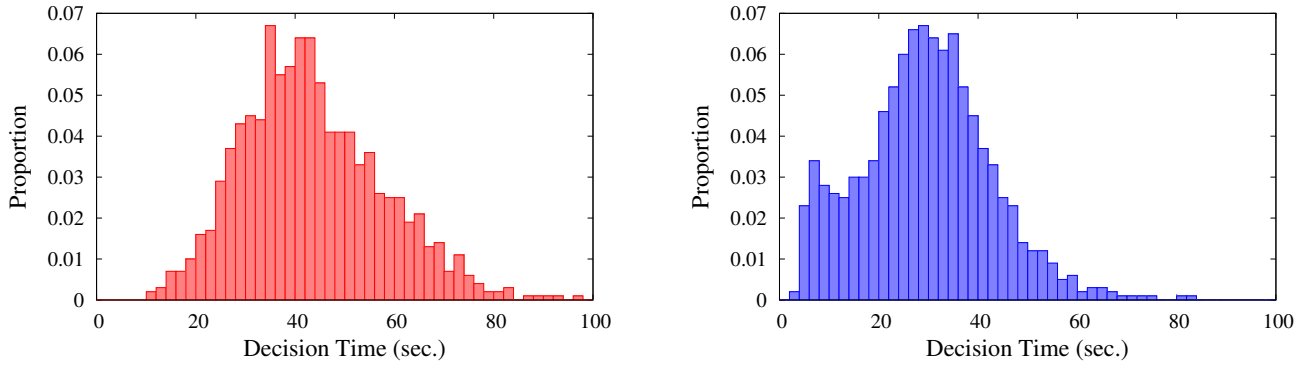
5.3 Detection of Other Game Bots

To further test our HOP system, without retraining the neural network, we perform a smaller experiment on a different game bot from a different game. While Diablo 2 is not an MMOG, it has an MMOG-like economy (items may be traded with thousands of other players) and is also plagued by game bots. This set of experiments studies MMBot, a popular free bot for Diablo 2 that is built using the AutoIt scripting language [4]. Similar to Glider, MMBot automates various tasks in the game to accumulate treasure or experience. However, unlike Glider, MMBot does not read the memory space of the game, but instead is based entirely on keyboard/mouse automation, and pixel scanning. As Diablo 2 has a much different interface (top-down isometric view rather than first person view like World of Warcraft) and much different controls, the purpose of these experiments is to test how general our system is and to show that it is not limited to any specific bot or game.

We collect a total of 20 hours of Diablo 2 traces, both bot and human. We run MMBot for 10 hours and have 5 humans play Diablo 2 for a total of 10 hours. We then reuse our existing neural network (40 nodes, 4 action-input, 9 inputs per block) with the adjusted threshold value to optimize our detection results. Without retraining, the neural network achieves a true positive rate of 0.864 on the bot and a true negative rate of 1.0 on the human players. This result shows that our HOP system is able to capture certain invariants in the behavior of bots across different games and different bot implementations, indicating the possible potential of HOP-based systems for other applications.

Table 3: True Positive Rates for Bots

Bots									
#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
0.988	1.000	0.998	1.000	1.000	1.000	1.000	1.000	0.998	1.000



(a) Bot (b) Human

Figure 10: Decision Time Distribution

5.4 System Overhead

Our proposed system at the server side (i.e., the server-side analyzer) is required to process thousands of users simultaneously in real-time, so it must be efficient in terms of memory and computation. Now we estimate the overhead of the analyzer for supporting 5,000 users, far more than the regular workload of a typical World of Warcraft server. The analyzer process, which includes the neural network, is profiled using `valgrind` and consumes only 37 KBytes of memory during operation. The prototype of our system is designed to use a single-thread multiple-client model with time-multiplexing, and thus only one process is used. Of course, additional processes could be used to process in parallel.

The primary memory requirement is to accommodate the accumulated user-input actions and neural network outputs for each online user. A single user-input action consumes 16 bytes, 4 bytes each for distance, duration, and displacement, and 2 bytes each for virtual key and angle. A block of 4 user-input actions consumes 64 bytes. A block of up to 16 neural network outputs requires 2 bytes as a bit-array. The per-user memory requirement is approximately 66 bytes, barely more than the maximum length of account names on World of Warcraft, which is 64 bytes. If 66 bytes is scaled to 5,000 online users, this is only 330 KBytes in total, which is negligible considering that the game currently stores the position, level, health, and literally dozens of other attributes and items for each user.

The computational overhead is also very low. The computation time for processing all 95 hours of traces is measured using the Linux `time` command. The analyzer can process the full set of traces, over 286,626 user-input actions, in only 385 milliseconds on a Pentium 4 Xeon 3.0Ghz. In other words, the analyzer can process approximately 296 hours of traces per second using a single CPU. A server with 5,000 users would generate approximately 1.38 hours of traces per second, a tiny fraction of the above processing rate.

6. LIMITATIONS

The limitations of this research work are mainly in two aspects: experimental limitations and potential evasion against the HOP system. In the following, we give a detailed description of these limitations.

6.1 Experimental Limitations

The size of our player group, 30, is insufficient to cover all kinds of human playing behaviors. It would be better to have a larger group size for characterizing the human playing behaviors in MMOGs. In addition, our study is mainly conducted in a lab environment, which limits the possible variations in hardware, as well as other environmental factors. Although lab settings allow for greater control, they are less ecologically valid than natural settings, where people play games on their home computers. In our future work, we plan to recruit a larger number of players with more players playing at home.

Our analysis is limited to one type of bots in one game. Although Glider is a typical World of Warcraft bot, there are a number of other bots [39,41,44,54] and their behaviors may vary. Moreover, other games may be quite different from World of Warcraft in terms of game plays, controls, and so on. A further study across multiple MMOGs with multiple bots is needed to confirm whether our HOP system is effective for broader applications. Additionally, while the bot and human characters in our study overlap in levels and classes, a more controlled experiment with the exact matchings in levels, classes, items, and skills, could lead to more accurate experimental results.

A few details of our experiments cannot be fully described due to limited space, which could hinder the reproduction of our work. In particular, the exact waypoints that the game bots followed and the monsters they fought are not included in the paper. To compensate for this limitation, we have made our bot profiles and our detection system available online at `http://www.cs.wm.edu/~hnw/hop/`.

6.2 Potential Evasion

Like other intrusion detection systems, HOPs are open to future evasions. Upon the adoption of the proposed HOP system, the bot creators will seek various ways to evade it. The focus of the following discussion is on two main potential evasions: (1) bots could either interfere with the user-input collection or manipulate the user-input stream at the client side; and (2) bots could mimic human behaviors to evade detection.

Since the user-input stream is collected at the client side, the bots can easily interfere with the user-input collection. A bot program could hinder the user-input collection either by disabling the client-side exporter or by intercepting the network traffic containing the user-input data. However, the server-side analyzer can simply block any game client that refuses to send the user-input stream. In other words, a simple policy of "no user-input, no game" can simply thwart this potential evasion. For those bot programs that attempt to manipulate the user-input stream, since they already have total control over the user-input through mouse and keyboard events, the additional benefit provided by the manipulation would be limited.

A more effective evasion for bots is to mimic human behaviors. The most obvious approach to mimicking humans would be a replay attack. That is, a bot could record a human play of the game and then simply replay the recording. However, the game environment especially in MMOGs is highly dynamic. A simple replay attack, based on a prerecorded game play, cannot successfully adapt to the constantly changing game conditions. Meanwhile, fully controlling the game via replay would require separate recordings of virtually all possible interactions with the game, which is clearly not feasible due to the large variety of different game tasks.

A more sophisticated approach is to use random models for generating the different user-input actions. However, this approach would require a separate model for each statistic and each type of user-input action. With six different action types and seven statistics, it needs more than 40 models just to capture the marginal distributions. In addition, there are complicated inter-relations between different actions, statistics, and game tasks. While there are new techniques that can generate synthetic user-input with some human behavioral characteristics, current techniques can merely generate random mouse movements (not useful for performing specific tasks) and are limited to capturing only basic statistics [35, 36].

More importantly, the HOP system is not based on any single metric of the human behavior, but rather, a collection of different kinds of behavioral metrics composed by neural networks. A successful evasion of the neural network could require a simultaneous attack on several of these different metrics. Although it is relatively easy to mimic a single metric of the human behavior, such as keystroke inter-arrival time, fully mimicking all aspects of the human behavior in a highly dynamic environment like MMOGs could require non-trivial efforts.

The threat of mimicry attacks [50] is real to behavior-based intrusion detection systems including HOPs. In general, we believe that it is possible for a highly motivated bot creator to build a more complicated game bot, which mimics multiple aspects of human behaviors, to evade the HOP system but at the cost of significant time and efforts.

7. RELATED WORK

Exploiting online games has attracted increasing interest in recent years. Yan *et al.* [56] summarized commonly-used exploiting methods in online games and categorized them along three dimensions: vulnerability, consequence, and exploiter. In addition, they pointed out that fairness should be taken into account to understand game exploits. Webb *et al.* [52] presented a different classification of game exploits. They categorized 15 types of exploits into four levels: game, application, protocol, and infrastructure, and discussed countermeasures for both client-server and peer-to-peer architectures. Muttik [34] surveyed security threats emerging in MMOGs, and discussed potential solutions to secure online games from multiple perspectives including technology, economy, and human factor. Hoglund and McGraw [24] provided a comprehensive coverage of game exploits in MMOGs, shedding light on a number of topics and issues.

7.1 Anti-Cheating

With the ever-increasing severity of game exploits, securing online games has received wide attention. The research work on anti-cheating generally can be classified into two categories: game cheating prevention and game cheating detection. The former refers to the mechanisms that deter game cheating from happening and the latter comprises the methods that identify occurrences of cheating in a game. For MMOGs, a cheat-proof design, especially the design of the game client program and the communication protocol, is essential to prevent most of game exploits from occurring. This is because (1) the client program of an MMOG is under the full control of a game player and (2) the communication at the client side might be manipulated for the advantage of player.

The prevention of game exploits has been the subject of a number of works. Baughman *et al.* [2] uncovered the possibility of time cheats (e.g., look-ahead and suppress-correct cheats) through exploiting communication protocols for both centralized and distributed online games, and designed a lockstep protocol, which tightly synchronizes the message communication via two-phase commitment, to prevent cheats. Following their work, a number of other time-cheat-resistant protocols [9, 13, 15] have been developed. In [32], Mönch *et al.* proposed a framework for preventing game client programs from being tampered with. The framework employs mobile guards, small pieces of code dynamically downloaded from the game server, to validate and protect the game client. Yampolskiy *et al.* [55] devised a protection mechanism for online card games, which embeds CAPTCHA tests in the cards by replacing the card face with text. Besides software approaches, hardware-based approaches to countering game exploits have also been proposed. Golle *et al.* [22] presented a special hardware device that implements physical CAPTCHA tests. The device can prevent game bots based on the premise that physical CAPTCHA tests such as pressing certain buttons are too difficult for bots to resolve without human involvement.

In practice, it is extremely hard to eliminate all potential game exploits. Thus, accurate and quick detection of game exploits is critical for securing on-line games. Since game bots are a commonly-used exploit, a fair amount of research has focused on detecting and countering them. Based on traffic analysis, Chen *et al.* [10] found that the traffic gen-

erated by the official client differs from that generated by standalone bot programs. Their approach, however, is not effective against recent game bots, as the majority of current MMOG bots interact with official clients. In [11, 12], the difference of movement paths between human players and bots in a first-person shooter (FPS) game is revealed and then used for the development of trajectory-based detection methods. However, it is unlikely that this type of detection method can achieve similar speed and accuracy in MMOGs, because maps used in MMOGs are much larger than those in FPS games and avatar trajectories in MMOGs are far more complicated. Indeed, Mitterhofer *et al.* [31] used movement paths in World of Warcraft and their method requires from 12 to 60 minutes to detect game bots. Thawonmas *et al.* [47] introduced a behavior-based bot detection method, which relies on discrepancies in action frequencies between human players and bots. However, compared to our work, the metric used for their detection, action frequency, is coarse-grained and has low discriminability, resulting in low detection ratio (0.36 recall ratio on average) and long detection time (at least 15 minutes).

As game clients in general cannot be trusted, usually the detection decision is made at servers. Schluessler *et al.* [45] presented a client-side detection scheme, which detects input data generated by game bots by utilizing special hardware. The hardware is used to provide a tamper-resistant environment for the detection module. The detection module compares the input data generated by input devices (mouse and keyboard) with those consumed by the game application and fires an alert once a discrepancy is found.

7.2 Behavioral Biometrics

The idea of HOPs is largely inspired by behavioral biometrics based on keystroke dynamics [5,25,33,40] and mouse dynamics [1, 20, 43]. Analogous to handwritten signatures, keystroke dynamics and mouse dynamics are regarded as unique to each person. Therefore, their applications in user authentication and identification have been extensively investigated [1,5,20,25,33,40,43]. Generating synthetic mouse dynamics from real mouse actions has also been studied [35, 36]. In spite of the fact that our system also utilizes the characteristics of keystroke and mouse dynamics, it significantly differs from aforementioned biometric systems in that our system leverages the distinction on game play between human players and game bots, which is reflected by keystroke and mouse dynamics, to distinguish human players from game bots. In contrast, those biometric systems exploit the uniqueness of keystroke dynamics or mouse dynamics for identification, i.e., matching a person with his/her identity on the basis of either dynamics.

8. CONCLUSION

In this paper, we presented a game bot defense system that utilizes HOPs to detect game bots. The proposed HOPs leverage the differences of game playing behaviors such as keyboard and mouse actions between human players and game bots to identify bot programs. Compared to conventional HIPs such as CAPTCHAs, HOPs are transparent to users and work in a continuous manner. We collected 95-hour user-input traces from World of Warcraft. By carefully analyzing the traces, we revealed that there exist significant differences between bots and humans in a variety of characteristics derived from game playing actions, which motivate the design of the proposed HOP defense system.

The HOP defense system comprises a client-side exporter and a server-side analyzer. The exporter is used to transmit a stream of user-input actions and the analyzer is used to process the action stream to capture bots. The core of the analyzer is a cascade-correlation neural network, which takes an action stream as input and determines if the stream generator is a bot or a human player. We also employed a simple voting algorithm to further improve detection accuracy. Based on the collected user-input traces, we conducted a series of experiments to evaluate the effectiveness of the defense system under different configurations. Our results show that the system can detect over 99% of current game bots with no false positives within a minute and the overhead of the detection is negligible or minor in terms of induced network traffic, CPU, and memory cost. As our detection engine only relies on user-input information, our HOP system is generic to MMOGs.

Acknowledgments

We are very grateful to our shepherd Paul C. Van Oorschot and the anonymous reviewers for their insightful and detailed comments. This work was partially supported by NSF grants CNS-0627339 and CNS-0627340.

9. REFERENCES

[1] A. A. E. Ahmed and I. Traore. A new biometric technology based on mouse dynamics. *IEEE Trans. on Dependable and Secure Computing (TDSC)*, 4(3), 2007.

[2] N. E. Baughman and B. N. Levine. Cheat-proof playout for centralized and distributed online games. In *Proceedings of the 20th IEEE INFOCOM*, Anchorage, AK, USA, April 2001.

[3] BBC News Staff. Legal battle over warcraft 'bot'. `http://news.bbc.co.uk/2/hi/technology/7314353.stm` [Accessed: Jan. 30, 2009].

[4] J. Bennett. AutoIt: Automate and script windows tasks. `http://www.autoit.com/` [Accessed: Apr. 20, 2009].

[5] F. Bergadano, D. Gunetti, and C. Picardi. User authentication through keystroke dynamics. *ACM Trans. on Information and System Security (TISSEC)*, 5(4), 2002.

[6] Blizzard Entertainment. MDY industries, LLC., vs. Blizzard Entertainment, Inc., and Vivendi Games, Inc. `http://gamepolitics.com/images/legal/blizz-v-MDY.pdf` [Accessed: Jan. 30, 2009].

[7] Blizzard Entertainment. World of Warcraft subscriber base reaches 11.5 million worldwide. `http://eu.blizzard.com/en/press/081223.html` [Accessed: Jul. 24, 2009].

[8] P. Caldwell. Blizzard bans 59,000 WOW accounts. `http://www.gamespot.com/news/6154708.html` [Accessed: Aug. 13, 2009].

[9] B. D. Chen and M. Maheswaran. A cheat controlled protocol for centralized online multiplayer games. In *Proceedings of the 3rd ACM SIGCOMM NetGames*, Portland, OR, USA, August 2004.

[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 *Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE'06)*, June 2006.

[11] K.-T. Chen, A. Liao, H.-K. K. Pao, and H.-H. Chu. Game bot detection based on avatar trajectory. In *Proceedings of the 7th International Conference on Entertainment Computing*, Pittsburgh, PA, USA, September 2008.

[12] K.-T. Chen, H.-K. K. Pao, and H.-C. Chang. Game bot identification based on manifold learning. In *Proceedings of*

the 7th ACM SIGCOMM NetGames, Worcester, MA, USA, October 2008.

[13] E. Cronin, B. Filstrup, and S. Jamin. Cheat-proofing dead reckoned multiplayer games (extended abstract). In *Proceedings of the 2nd International Conference on Application and Development of Computer Games*, Hong Kong, China, January 2003.

[14] Diablo 2 Guide. D2 bots. `http://www.diablo2guide.com/bots.php` [Accessed: Nov. 2, 2008].

[15] C. G. Dickey, D. Zappala, V. Lo, and J. Marr. Low latency and cheat-proof event ordering for peer-to-peer games. In *Proceedings of the 14th ACM NOSSDAV*, Cork, Ireland, June 2004.

[16] DMW World. DMW anti-cheat system. `http://www.dmwworld.com/viewfaq/show/374` [Accessed: Aug. 13, 2009].

[17] Even Balance Inc. PunkBuster online countermeasures. `http://www.evenbalance.com` [Accessed: Jul. 9, 2008].

[18] Exploits R Us. Ultima Online bots and cheats. `http://www.exploitsrus.com/uo/bots.html` [Accessed: Nov. 2, 2008].

[19] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, 1990.

[20] H. Gamboa and A. Fred. A behavioral biometric system based on human computer interaction. In *Proceedings of SPIE: Biometric Technology for Human Identification*, volume 5404, 2004.

[21] S. Gianvecchio, M. Xie, Z. Wu, and H. Wang. Measurement and classification of humans and bots in internet chat. In *Proceedings of the 17th USENIX Security Symposium*, San Jose, CA, USA, July 2008.

[22] P. Golle and N. Ducheneaut. Preventing bots from playing online games. *Computers in Entertainment*, 3(3), 2005.

[23] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium*, San Jose, CA, USA, July 2008.

[24] G. Hoglund and G. McGraw. *Exploiting Online Games: Cheating Massively Distributed Systems*. No Starch Press, 2007.

[25] R. Joyce and G. Gupta. Identity authentication based on keystroke latencies. *Communications of the ACM*, 33(2), 1990.

[26] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer. Behavior-based Spyware Detection. In *Proceedings of the 15th USENIX Security Symposium*, Vancouver, Canada, August 2006.

[27] U. Kukreja, W. E. Stevenson, and F. E. Ritter. RUI - recording user input from interfaces under Windows and Mac OS X. *Behavior Research Methods, Instruments, and Computers*, 38(4):656–659, 2006.

[28] C. McSherry. A new gaming feature? spyware. `http://www.eff.org/deeplinks/2005/10/new-gaming-feature-spyware` [Accessed: Jul. 9, 2008].

[29] MDY Industries. MMO glider. `http://www.mmoglider.com/` [Accessed: Nov. 2, 2008].

[30] W. Meloni. State of the game industry 2008. In *GameOn Finance Conference*, San Diego, CA, USA, October 2008.

[31] S. Mitterhofer, C. Platzer, C. Kruegel, and E. Kirda. Server-side bot detection in massive multiplayer online games. *IEEE Security and Privacy*, 7(3), May/June 2009.

[32] C. Mönch, G. Grimen, and R. Midtstraum. Protecting online games against cheating. In *Proceedings of the 5th ACM SIGCOMM NetGames*, Singapore, October 2006.

[33] F. Monrose and A. Rubin. Authentication via keystroke dynamics. In *Proceedings of the 4th ACM CCS*, Zurich, Switzerland, April 1997.

[34] I. Muttik. Securing virtual worlds against real attacks. `http://www.mcafee.com/us/local_content/white_papers/threat_center/wp_online_gaming.pdf` [Accessed: Nov. 2, 2008].

[35] A. Nazar. Synthesis and simulation of mouse dynamics. Master's thesis, University of Victoria, October 2007.

[36] A. Nazar, I. Traore, and A. A. E. Ahmed. Inverse biometrics for mouse dynamics. *International Journal of Pattern Recognition and Artificial Intelligence*, 22(3):461–495, 2008.

[37] P. Neva. Bots back in the box. `http://www.secondlifeherald.com/slh/2006/11/bots_back_in_th.html` [Accessed: Nov. 2, 2008].

[38] nProtect. nProtect GameGuard. `http://eng.nprotect.com/nprotect_gameguard.htm` [Accessed: Jul. 9, 2008].

[39] M. M. Owned. World of Warcraft bots and programs forum. `http://www.mmowned.com/forums/bots-programs/` [Accessed: Jul. 21, 2009].

[40] A. Peacock, X. Ke, and M. Wilkerson. Typing patterns: A key to user identification. *IEEE Security and Privacy*, 2(5), 2004.

[41] PiroX. PiroX Bot - World of Warcraft bot. `http://www.piroxbots.com/` [Accessed: Jul. 25, 2009].

[42] M. D. Preda, M. Christodorescu, S. Jha, and S. Debray. A semantics-based approach to malware detection. In *Proceedings of the 34th ACM POPL*, Nice, France, January 2007.

[43] M. Pusara and C. E. Brodley. User re-authentication via mouse movements. In *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, Washington, DC, USA, October 2004.

[44] Rhabot. Rhabot - World of Warcraft bot. `http://www.rhabot.com/` [Accessed: Nov. 2, 2008].

[45] T. Schluessler, S. Goglin, and E. Johnson. Is a bot at the controls?: Detecting input data attacks. In *Proceedings of the 6th ACM SIGCOMM NetGames*, Melbourne, Australia, September 2007.

[46] Slashdot. Confessions of an Ultima Online gold farmer. `http://slashdot.org/games/05/01/26/1531210.shtml` [Accessed: Jul. 9, 2008].

[47] R. Thawonmas, Y. Kashifuji, and K.-T. Chen. Detection of MMORPG bots based on behavior analysis. In *Proceedings of 5th ACM International Conference on Advances in Computer Entertainment Technology (ACE'08)*, Yokohama, Japan, December 2008.

[48] The MMO RPG Exchange. World of Warcraft exchange. `http://themmorpgexchange.com/` [Accessed: Jul. 25, 2009].

[49] Valve Corporation. Valve anti-cheat system (VAC). `https://support.steampowered.com/kb_article.php?p_faqid=370` [Accessed: Jul. 9, 2008].

[50] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM CCS*, Washingtion, DC, USA, November 2002.

[51] H. Wang, D. Zhang, and K. G. Shin. Detecting SYN flooding attacks. In *Proceedings of the 21st IEEE INFOCOM*, New York, NY, USA, June 2002.

[52] S. D. Webb and S. Soh. Cheating in networked computer games – a review. In *Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts*, Perth, Australia, September 2007.

[53] W. Willinger, V. Paxson, and M. S. Taqqu. Self-similarity and heavy tails: Structural modeling of network traffic. In *Statistical Techniques and Applications*, pages 27–53. Verlag, 1998.

[54] WoW Panda. ZoloFighter - World of Warcraft bot. `http://www.zolohouse.com/wow/wowFighter/` [Accessed: Jul. 25, 2009].

[55] R. V. Yampolskiy and V. Govindaraju. Embedded non-interactive continuous bot detection. *Computers in Entertainment*, 5(4), 2007.

[56] J. Yan and B. Randell. A systematic classification of cheating in online games. In *Proceedings of the 4th ACM SIGCOMM NetGames*, Hawthorne, NY, USA, October 2005.