# Shoot for the Moon and You Will Never Miss: Characterizing and Detecting Aimbots in Online Games

Daiping Liu*, Xing Gao*†, Mingwei Zhang‡, Haining Wang*

*University of Delaware {dpliu,hnw}@udel.edu
†College of William and Mary xgao01@email.wm.edu
‡Intel Labs mingwei.zhang@intel.com

## 1. INTRODUCTION

The online game market is one of the fastest growing entertainment industries in the world. By 2016, the size of global game market is forecasted to reach $86.1 billion. Among the many genres of online games, first person shooter (FPS) is the second most popular. Due to the popularity and economic importance of FPS games, it is imperative to shield them from cheating for the benefit of both game operators and players, especially considering that the competitive nature of FPS games provides strong incentives to cheat. With regard to FPS games, the most common cheat is the use of automated tools to aim and shoot more accurately, also known as *aimbots*. While people have proposed different approaches for detecting fully automated bots, none of them can detect aimbots. Aimbot detection is notoriously difficult because aimbots are completely passive (i.e., no tampering with game data or code) and interact with human players intensively (i.e., semi-automatic).

In this work, we propose an accurate and robust server-side aimbot detector, called AimDetect. We define eight novel features to model the differences between aimbots and honest players from two aspects: (1) the anomalous behaviors stemming from the automation of aimbots and (2) the blatant nature of cheaters. All these features can be precisely measured on the server-side and also be universally extracted across most modern FPS games. These features are then fed into a novel two-layer classification framework, which accurately determines if a player is using aimbot. In the first step, a machine-learning algorithm is invoked every time a kill event occurs. The learned model classifies each kill as from aimbots or honest players. We currently utilize Support Vector Machine (SVM) for this step. AimDetect considers every kill event as an independent observation. To further reduce false alarms, in the second step, a sequential hypothesis testing algorithm is used to compute a comprehensive anomaly score over a sequence of observations. Evaluation results show that our two-layer classifier significantly outperforms traditional single-layer approach.

To demonstrate the efficacy of AimDetect, we implement a prototype and evaluate its performance using the traces collected on a public game server. One challenge here is to collect ground truth data from our field deployment. To this end, we devise a novel and generic technique called *bait-target* that can effectively identify modern aimbots. We set up a public game server hosting CS-1.6 and we ran our field measurement for one month. The evaluation results show that AimDetect is able to capture almost all aimbots with only 0.7% false positive rate.



*SIGMETRICS '16 June 14-18, 2016, Antibes Juan-Les-Pins, France*



ACM ISBN 978-1-4503-4266-7/16/06.

DOI: http://dx.doi.org/10.1145/2896377.2901503

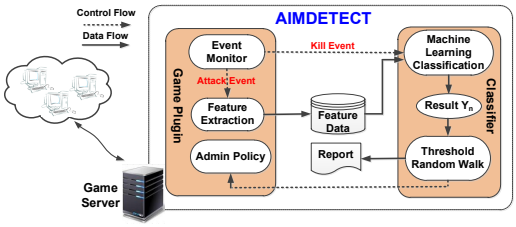


Figure 1: Overview of AIMDETECT

## 2. AIMDETECT DESIGN

At a high level, AimDetect is motivated by our measurement results that aimbots usually manifest themselves through a series of suspicious behavioral patterns. Figure 1 illustrates the architecture of AimDetect, which completely resides on game servers and consists of an in-game plugin and a stand-alone classifier. The in-game plugin has three modules: event monitor, feature extraction, and administration policy. The first module monitors in-game events like aiming at an enemy, death and hit. These events drive the feature extraction module, which collects predefined in-game features as listed in Table 1. Administrators can set policies that are executed when aimbots are detected. The stand-alone classifier will be invoked once a kill event is captured in the event monitor. Our classifier comprises two steps. First, the collected features are fed into a machine learning classification system, which determines whether the killer uses aimbot. Each kill could be considered as an independent observation. Then, to reduce false positives, which are less tolerable than false negatives in production games, we compute a comprehensive anomaly score over a sequence of observations, instead of deciding on one single observation. A sequential hypothesis testing algorithm, specifically Threshold Random Walk (TRW), is utilized in the second step. When an aimbot is detected, the classifier will notify the in-game plugin to execute preset policies.

## 3. AIMDETECT EVALUATION

**Server Deployment.** We host CS-1.6 on our public server. CS-1.6 represents one of the most popular modern FPS games. Our CS-1.6 server is published on www.game

| Feature | Description |
|---|---|
| $F_1$ | Num of attack attempts before a kill |
| $F_2$ | Num of deaths before a kill |
| $F_3$ | Time to kill |
| $F_4$ | Cursor acceleration |
| $F_5$ | Ratio of hits when the attackers are crouching |
| $F_6$ | State-aware hits |
| $F_7$ | State-aware hit ratio |
| $F_8$ | Suspiciousness of headshots |

Table 1: Feature set defined in AimDetect.

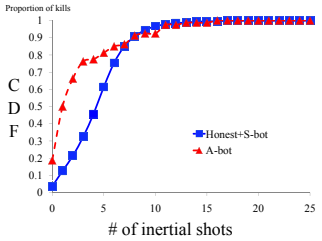


Figure 2: Statistics of inertial shots.

tracker.com/games/cs. We allocate 32 slots, the maximum number a server could host.

**Bait-Target Design.** To collect ground truth data on our field deployment, we devise bait-targets. The basic idea is to simulate an opponent using a human-invisible bot, which is referred to as a bait-target, as a trap to catch aimbots. We design bait-target by leveraging our key observation: when a player kills an opponent, he will highly likely shoot one or more bullets, which we call *inertial shots*. Figure 2 presents the statistics of inertial shots for aimbots and honest players. We find that only 3.6% of kills come with no inertial shots and for each player, most of their kills have at least one inertial shot. Thus, our observation would hold in most cases.

The position where a bait-target is placed can also affect the detection accuracy. If a bait-target is placed very close to a player's crosshair or it is placed near a real opponent, many false alarms could be raised. However, if a bait-target is placed far away from a player's crosshair, we may miss most aimbots. We decide to place a bait-target *exactly above the victim* upon a kill event. When a player just kills an opponent, its crosshair should be close to the victim and thus an honest player's inertial shots are usually gathered around the victim. However, aimbots can *immediately* catch a next valid target, causing their inertial shots to hit the bait-target. Figure 3 illustrates the placement of a bait-target.

**Dataset.** Our dataset contains one-month traces. During our deployment, 294 non-empty connections from 131 distinct IP addresses play in our server. These IP addresses are mapped to 31 countries. In total, using bait-targets, we can identify 38 aimbots.

**Evaluation Results.** We first evaluate if SVM is able to correctly classify those kills from honest players and aimbots. The accuracy of the classifier is affected by the size of its training set. Thus, we evaluate the performance of SVM by varying the training set size from 300 to 1,100 kills. For each size, we test 10 times with kills from randomly selected honest and cheating players. The averaged results are shown in Figure 4. We can see that the accuracy increases as the training set grows. However, after the training set reaches 900, the accuracy will stabilize at around 92%. So, our further evaluation is based on a training set of size 900. And we randomly generate five pairs of training and testing sets. In this setting, our SVM classifier achieves true positive rate ($TPR_s$) of 83.6% and false positive rate ($FPR_s$) of 5.8%.


Figure 3: An example of bait-target. The bait-target is rendered as visible for illustration purpose.

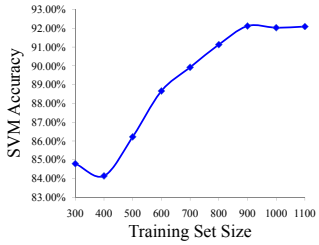


Figure 4: Training set size *v.s.* SVM accuracy.

Although our SVM classifier only achieves acceptable performance, SVM + TRW can significantly improve the detection accuracy of AimDetect. TRW has four parameters, $\theta_0$, $\theta_1$, $\alpha$, and $\beta$. Based on the results of the SVM classifier, $\theta_1$ can be set to 83.6% and $\theta_0$ to 5.8%; $\alpha$ and $\beta$ are user-chosen false positive and false negative rates for TRW. Table 2 lists the final accuracy of AimDetect with different settings of $\alpha$ and $\beta$. The results are averaged on the five pairs of training and testing sets.

AimDetect is able to achieve a quite promising $TPR_{s+t}$ in all settings. For specific training sets, when $\alpha$ and $\beta$ decrease, some false negatives occur because more observations are needed. The $FPR_{s+t}$ in all settings are acceptable, and when $\alpha$ and $\beta$ are set quite low (i.e., 0.001), the $FPR_{s+t}$ even drops to 0.7%.

| $\alpha$ | $\beta$ | $TPR_{s+t}$ | $FPR_{s+t}$ |
|---|---|---|---|
| 0.05 | 0.05 | 100% | 7.5% |
| 0.01 | 0.01 | 99% | 4% |
| 0.005 | 0.005 | 98.5% | 1.9% |
| 0.001 | 0.001 | 97.6% | 0.7% |

Table 2: Detection performance of SVM + TRW.

## 4. CONCLUSION

We have designed and implemented an accurate and robust server-side aimbot detector, AimDetect. AimDetect utilizes a set of novel features to model the behavioral patterns of aimbots and honest players, which stem from both the automation of aimbots and the blatant nature of cheaters. These features are then fed into a two-step classifier, which first classifies each kill event as from either honest players or aimbots using SVM and then employs TRW to compute a comprehensive anomaly score. Our evaluation results show that AimDetect is able to capture almost all of the aimbots with quite low false positive rate.