GAMING THE GAME: DEFEATING A GAME CAPTCHA WITH EFFICIENT AND ROBUST HYBRID ATTACKS

Song Gao, Manar Mohamed, Nitesh Saxena, and Chengcui Zhang

The University of Alabama at Birmingham, USA
{gaos, manar, saxena, zhang}@cis.uab.edu

ABSTRACT

Dynamic Cognitive Game (DCG) CAPTCHAs are a promising new generation of interactive CAPTCHAs aiming to provide improved security against automated and human-solver relay attacks. Unlike existing CAPTCHAs, defeating DCG CAPTCHAs using pure automated attacks or pure relay attacks may be challenging in practice due to the fundamental limitations of computer algorithms (semantic gap) and synchronization issues with solvers. To overcome this barrier, we propose two hybrid attack frameworks, which carefully combine the strengths of an automated program and offline/online human intelligence. These hybrid attacks require maintaining the synchronization only between the game and the bot similar to a pure automated attack, while solving the static AI problem (i.e., bridging the semantic gap) behind the game challenge similar to a pure relay attack. As a crucial component of our framework, we design a new DCG object tracking algorithm, based on color code histogram, and show that it is simpler, more efficient and more robust compared to several known tracking approaches. We demonstrate that both frameworks can effectively defeat a wide range of DCG CAPTCHAs.

Index Terms – CAPTCHA, web security, hybrid attack, visual processing, multi-object tracking

1. INTRODUCTION

Web-based services are vulnerable to different forms of attacks, including denial of service, registration of spam email accounts and password brute force. Such abuse or resource waste can be reduced through a commonly deployed defense mechanism, called "CAPTCHA", which is *a program of an AI problem that is relatively easy for human to solve but very difficult for a computer program* [1]. An AI problem in a CAPTCHA can be represented in many ways, such as texts, image, audios, videos, and logical puzzles. Out of these, text-based CAPTCHA is widely used in many web-based industries.

However, many real world CAPTCHAs have been broken using *automated attacks* with noticeable success rates in recent years (e.g., [2, 3]). They have also been subject to human-solver *relay attacks* [4] whereby the CAPTCHA challenge is outsourced to, and solved by, a third-party remote human-solver in real-time.

These vulnerabilities provide a sound motivation to explore CAPTCHA alternatives. This paper focuses on a new generation of CAPTCHA, called Dynamic Cognitive Game (DCG) CAPTCHA [6][1], which challenges the user to perform a game-like cognitive task interacting with moving objects within a series of images in a static scene (see examples in Figure 1). "are you a human" [5] released a series of instances of such DCGs, named "PlayThru".[2] DCGs provide several potential improvements in security compared with other existing CAPTCHAs. *First*, it is difficult for a computer to recognize moving objects and games through game instructions due to the semantic gap. *Second*, the dynamic nature requires synchronization between a user and a game, thereby making DCGs more resilient to *relay attacks*. *Third*, the user interaction with the game, such as the total number of drag-and-drop attempts and the error rate, can be utilized as a second layer defense to assess a user's "human likeness."

In this paper, we show that all the above improvements in security could still be effectively challenged. We propose a novel and powerful *hybrid attack* framework that can bypass the security advantages offered by DCGs by carefully combining the strengths of automated algorithms and human intelligence. Our **main contributions** are:

1. ***Two Novel Hybrid Attack Frameworks***: We develop two novel hybrid attack frameworks (*Section* 4) that can effectively subvert DCGs by utilizing real-time object tracking and human-solver based offline/online learning. Both attack frameworks require maintaining the synchronization only between the game and the bot similar to a pure automated attack, while solving the static AI problem (i.e., bridging the semantic gap) behind the game similar to a pure relay attack.
2. ***Efficient and Robust DCG Object Tracking***: As a crucial component of the two framework models, we design a new object tracking algorithm, based on *color code histogram* (CCH), and demonstrate that it is simpler, more efficient and more robust compared to several known tracking algorithms for DCG object tracking (*Section* 3).

[1] Henceforth, we use "DCG(s)" to denote "DCG CAPTCHA(s)"
[2] All images of the PlayThru challenges in this paper (Figure 2) are used for illustration only; these games were not attacked.

3. ***Attack Implementation and Evaluation***: We provide an implementation of the two attack frameworks, and evaluate their performance from the automated side (*Section* 5). The results indicate our attacks to be very efficient (< 4s of computation time) and robust (100% accuracy) in breaking current broad category of DCGs.

2. BACKGROUND

We first present a broad class of DCG prototypes we designed for the purpose of our study. We then review our recent prior work [6], which evaluated the usability of DCGs, and the security of DCGs against *pure auto-attacks* and *pure relay attacks*. This work [6] showed that DCGs are very easy for legitimate users to solve, and suggested the difficulty of pure auto-attacks and pure relay attacks against DCGs. The latter challenge is addressed in this paper by introducing *novel hybrid attacks*.

2.1. The DCG prototypes

The legal restriction on attacking commercial DCGs drives us to design and implement four animation-based DCG prototypes representing a broad class of DCGs, as shown in Figure 1. The games have the dimension 130×360 pixels, so they can easily fit into a web page. The components in a frame image of a DCG challenge (as indicated in Figure 1) include: (1) *Answer object*: a moving object (e.g., a ship) that should be dragged to the corresponding target object (e.g., sea) in order to pass the game. (2) *Target object*: an object onto which the corresponding answer object is dragged. (3) *Activity area*: the activity area of all moving foreground objects. (4) *Target area*: part of the background *that* includes the target object(s).

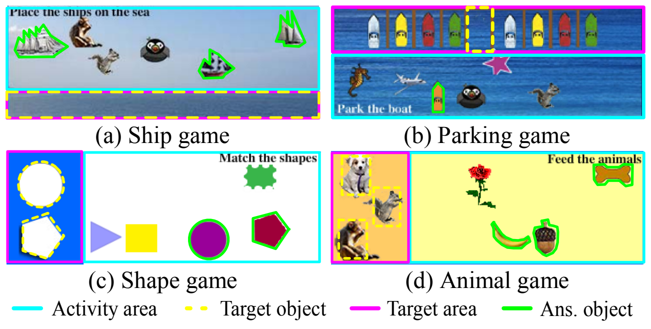

Fig. 1. Sample frame images of the four DCG prototypes

The general parameters in the four DCG prototypes include: (1) Three frame rates (10, 20, or 40 frames per second (fps)). (2) Three different numbers of moving objects (4, 5, or 6). (3) Average moving distance per frame: 1.207 pixels per frame (ppf) (1 ppf for orthogonal directions, i.e., N, S, E, and W, while 1.414 ppf for diagonal directions, i.e., NE, NW, SE, and SW). In total, 4×3×3=36 game challenges will be used in experiments. Each moving object is initialized with a randomized location and an orthogonal/diagonal moving direction. An object will move to a new random direction when colliding with another object or the game border. Generally, a DCG challenge has a static background; all of its moving objects have a constant speed; moving objects may overlap one another to a small extent; moving objects will not move to the target area unless they are dragged to.

2.2. Usability of DCGs

Three measurements, namely *success rate*, *completion time*, and *error rate per click*, are used to evaluate the proposed hybrid attacks compared with the performance of legitimate human users in playing the game. Error rate per click is defined as the ratio of the number of incorrect drag-and-drop attempts to the total number of drag-and-drop attempts in a game challenge.

The usability study of the above three measurements from [6] show that: 1) Honest users can always complete each challenge of the four game prototypes 100% successfully in about 11 seconds at most. 2) Increasing the number of moving object and/or frame rate may prolong the completion time. 3) Although users complete the game with 100% success rate, they are likely to make mistakes (i.e., range of average error rate per click: 3-9%) in a game, which requires the game server to have some tolerance on the incorrect drag-and-drop attempts. Once the number of incorrect attempts exceeds a certain threshold, the game server may consider the player as a bot (not human). Thus, the error rate per click is one of the key measurements to evaluate whether a game can be completed successfully.

2.3. Pure auto attack and pure relay attack on DCGs

The work presented in [6] proposed a dictionary-based auto-attack framework that consists of two phases, the online learning phase and the attacking phase. The static background of a game is detected based on the observation that at each pixel in a game scene, the true background color almost always appears as the most frequent color observed for a pixel over a consecutive set of frames. Thereby, foreground moving objects can be detected through background subtraction. The target area is derived from the remaining area after subtracting the activity area that is the superimposition of MBRs (Minimum Bounding Rectangles) of foreground objects collected from a set of consecutive game frames. The learning phase repeatedly scans different CAPTCHA challenges and continuously updates the dictionary of game scenes and the answer objects.

The auto-attack framework applies a random guessing for solving the semantic gap problem between the computer understanding of the game and the game semantics, which is not sufficient to collect enough knowledge before the session expires, even though the attacking phase could work effectively for a known game. Moreover, it also exposes a *set of serious weaknesses* due to the semantic gap:

1. *The auto-attack framework cannot detect the target area that overlaps the activity area* (e.g., Figure 2(a)).
2. *Dragging and dropping a moving object always follows a straight-line path, which does not work for more*

complex paths (e.g., Figure 2(b)).

3. *The auto-attack framework cannot detect dynamic target objects* (e.g., the butterfly net in Figure 2(c)).

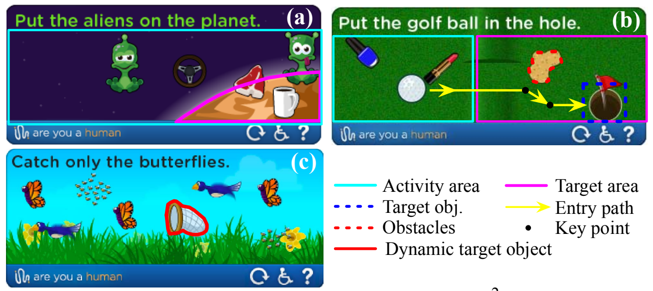

Fig. 2. Complex games from "are you a human"[2]. (a) The target area is in the activity area. (b) Obstacles (i.e., sand trap) on the entry path to the target object (i.e., hole). (c) The movement of the target object (i.e., net) is controlled by the mouse.

Similar to a pure relay attack on text-based CAPTCHAs, a bot, faced with a DCG challenge, needs to relay the challenge images to a human solver at a time and quickly receive the response (i.e., locations of target objects and the corresponding answer objects) in order to perform a correct drag-and-drop. This process needs to be done back and forth several times to break the game. However, there is a natural loss of synchronization between the bot relaying the challenge images and the human solver due to the dynamic nature of DCGs, resulting in increased game completion time as well as high error rate [6].

An alternative way for relay attack is to maintain the synchronization between the DCG server and the remote solver by employing a streaming approach that is similar to cloud gaming [6]. However, such an approach will increase the complexity on the bot side, which goes against the inherent simplicity and practicality of a relay attack [6].

3. REAL-TIME OBJECT TRACKING

A key component in both of our proposed frameworks is robust object tracking that preserves the synchronization between the game and the bot. The tracking efficiency affects whether a timely completion of a game could be done like a human, thereby becoming one of the key factors that determines the success rate. In this section, we introduce a simple and efficient color code histogram based tracker, and compare it with several existing tracking algorithms in the context of DCG object tracking.

Based on the characteristics of DCG prototypes, i.e., (1) the background is static, and (2) the moving object has unchanging appearance, we propose a simple and efficient tracking algorithm, named color code histogram based tracker (CCH) that generates the foreground object mask by utilizing the detected background, and associates the track to the same object based on 6-bit color code features [7]. Moving objects as well as the game scene are represented as a normalized 64-bin color code histogram. The tracking is thereby simplified as a histogram matching task in the subsequent frames by using histogram intersection [8] that is known to be robust against scaling and rotation. To alleviate the problem of object occlusion, CCH will abandon the current foreground mask if fewer objects than the number of tracks can be found. The extent of tolerance to partial occlusion relies on tuning the similarity threshold in histogram intersection.

Such a simple design also exposes two weaknesses: (1) The quality of the foreground mask in each frame completely relies on the quality of the detected background; and (2) Tracking may become inaccurate when multiple objects have similar color histograms. Future research may consider other clues in matching, such as motion, to alleviate this problem.

We also explore two representative tracking algorithms, Kanade–Lucas–Tomasi tracking algorithm (KLT) [9] and Kalman filter tracking algorithm (Kalman filter) [10] on their applicability and performance for DCG object tracking.

The effects of the number of objects and the object moving speed on the efficiency and effectiveness of each tracking algorithm (*without attacking*) are evaluated by applying each tracking algorithm on 9 cases of the Ship game initialized as described in Section 2.1. Each test case includes 200 consecutive frames sampled with a fixed interval (e.g., 0.1s). Therefore, a higher frame rate results in longer moving distance of the same object between two consecutive sampled frames. The background of the Ship game is learned in advance. Based on the average moving speed, i.e., 1.207 ppf, the average object moving speed in pixels-per-second (pps) unit under each frame rate (i.e., 10, 20, and 40 fps) are:

- **1x**: 12.07 pps = 1.207 ppf × 10 fps
- **2x**: 24.14 pps = 1.207 ppf × 20 fps
- **4x**: 48.28 pps = 1.207 ppf × 40 fps

Figures 3 and 4 exhibit the tracking completion times of the 9 test cases of each tracking algorithm from two different viewpoints. One group has fixed number of moving objects while another group has fixed object moving speed. Both results indicate that: (1) Compared with KLT, the efficiencies of Kalman filter and CCH remain relatively stable with the variation of the two parameters; and (2) CCH achieves the best efficiency due to the fact that it is based on direct object-level matching between two consecutive frames instead of time-consuming pixel-level feature based iterative local search, so it does not need to spend time on reviewing the previous states (like Kalman filter), while KLT is always the most time-consuming tracker.

In terms of effectiveness (robustness), only KLT loses track(s) (Figure 5) quickly when increasing the number of objects or the object moving speed (e.g., in average, 11.07s for 2x moving speed, and 4.41s for 4x speed.) It is possible that the corner points on the boundary of an object are misclassified as the corner points of another object by KLT tracker when the two objects overlap with each other. This may happen frequently due to the small game window size, and KLT tracker gradually loses tracking points (i.e., corner points) of some objects, and finally lose track of them.

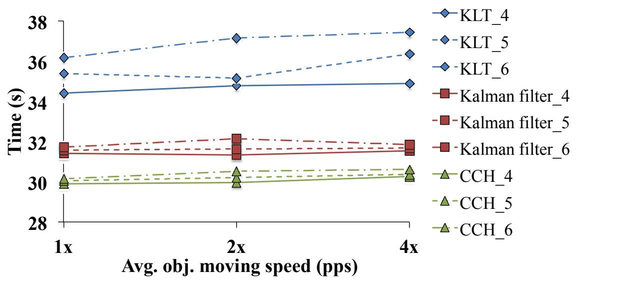

Fig. 3. Effect of the object moving speed on efficiency of the three tracking algorithms with fixed number of objects.

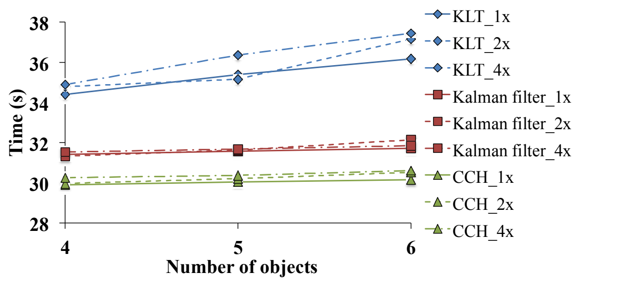

Fig. 4. Effect of the number of objects on efficiency of the three tracking algorithms with a fixed object moving speed

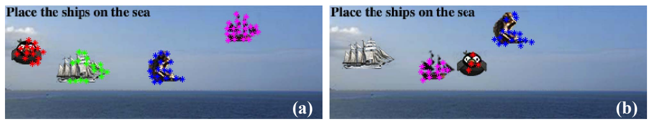

Fig. 5. KLT tracking. (a) Initial corner points on the moving objects; (b) KLT loses tracks on the "white ship" after 30s (10 fps).

In summary, the proposed CCH approach outperforms the other two approaches in efficiency and effectiveness. In light of this result, only the CCH approach has been integrated with our proposed hybrid attack frameworks discussed next. In addition, latest tracking methods may achieve similar or better (though not necessarily more efficient) results. Our main focus is to have a tracking method that satisfies efficiency and accuracy requirements for real-time response. Any tracking method that satisfies those requirements can be applied. On the other hand, more complex countermeasures, such as animated background/foreground and object occlusion, may disable the current tracking. However, the goal here is on developing a generalized attack model rather than testing all possible countermeasures. Moreover, more complex countermeasures may compromise the usability as well.

4. HYBRID ATTACK MODELS

4.1. Auto-attack with offline learning

Model I of our proposed hybrid attack framework (Figure 6), Auto-attack with Offline human Learning (AOffL), attacks a known game with the help of real-time tracking and offline knowledge.

In the *learning phase* of AOffL, the necessary knowledge related to a game scene is learned in advance from a remote human solver. In the *first step*, the bot starts a new game and keeps scanning frame images from the server. In the *second step*, called initial analysis, the bot detects the game background, moving objects and potential target areas, and keeps tracking the moving objects.

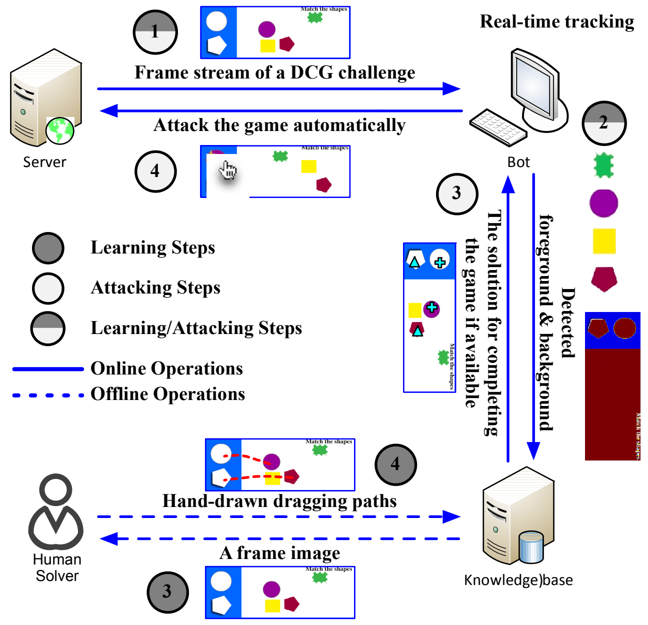
Real-time tracking
Frame stream of a DCG challenge
Attack the game automatically
Server
Bot
1
2
3
4
Learning Steps
Attacking Steps
Learning/Attacking Steps
Online Operations
Offline Operations
The solution for completing the game if available
Detected foreground & background
Hand-drawn dragging paths
A frame image
Human Solver
Knowledge)base

Fig. 6. Hybrid attack model I: Auto-attack with Offline human Learning (AOffL).

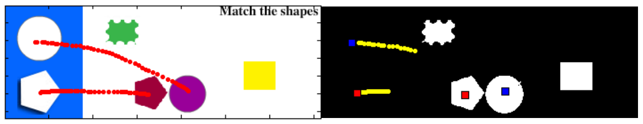

Fig. 7. (a) Interactive GUI for offline learning. Red lines indicate the solver's hand-drawn paths. (b) Extracted knowledge from the solver's response displayed in the foreground mask.

In the *third step*, similar to the static relay attack against a text-based CAPTCHA, only *one frame image* is sent from the bot to the solver, thereby avoiding the complexity of running as a game-streaming server. The interactive area of the AOffL GUI for the remote solver is shown in Figure 7(a). The bot-detected moving objects and the operational instruction are also provided to the remote solver. Through this GUI, the solver only needs to perform a very simple task of drawing line(s) from the answer object(s) to the corresponding target object(s), similar to what a legitimate user has to perform. These lines provide several clues for completing this game: (1) The start and the end points of each line label the locations of the answer object and the target object at that moment, respectively (Figure 7(b), red and blue squares); (2) They provide dragging path candidates such that in the attack phase, we can at least drag the same answer object to the starting point of the corresponding path, and follow the same path to the target object if the same game is seen (The same game means that a game challenge has the same game scene (background) and answer object(s).) The end portion of each line (e.g., the portion connected to the target object) can also be used as the basic entry path to the target object if straight-line paths are workable in this game. For example, 40% of each hand-drawn dragging path (starting from the end point) is treated as the basic entry path as shown in Figure 7(b) indicated by the yellow dots. Therefore, an answer object can always be dragged to the start point of the entry path first, and follow the entry path to the target finally. If a more complex path, such as the one in Figure 2(b), is required, a curvature threshold could be defined to identify those critical turning points with curvatures larger than the threshold, which finds the key points that must be passed in turn in a new path (Figure 2(b)).

In the *fourth and final step*, the above clues together with the initial analysis, i.e., background and foreground detection, will be recorded in the knowledge base. Answer objects as well as the background are represented in color code histograms. A continuous learning from the game server is required to build an up-to-date dictionary.

As shown in Figure 6, the *attacking phase* consists of the following steps. The initial analysis is performed (Step 1) followed by submitting a query to the knowledge base (Step 2). If a match can be found, the bot will drag an answer object from its current location provided by the real-time tracking to the corresponding target object (Step 3). The drag-and-drop attempt iterates until completing a game (success) or time out (failure) (Step 4). If a match cannot be found, which indicates that the game or answer objects are completely new, the framework will learn the game as the dictionary based auto-attack framework mentioned in Section 2.3. Moreover, the attacking phase is converted into offline learning just in case that brute-force based learning cannot work out the puzzle.

Our paper did not consider attacking the complex DCG games that involve dynamic target objects controlled by the mouse (e.g., game shown in Figure 2(c)). However, our framework can still be used to attack this game by making the following changes to the learning and attacking phases: (1) In the initial analysis, the background of the complex game is detected twice by putting the mouse (i.e., target object) in two different locations of the game window, respectively. If there is an obvious difference between these two backgrounds, the game has a dynamic target object. The target object can also be easily obtained from both backgrounds based on the mouse's location, eliminating unnecessary tracking of the target object. (2) The attacking phase is simplified as moving the mouse (e.g., moving the net) to the answer objects (e.g., butterfly) that are learned from the solver's response as usual.

4.2. Auto-attack with online learning

Model II of our hybrid attack framework (Figure 8), Auto-attack with Online human Learning (AOnL), attacks any game, seen or unseen, with the help of real-time tracking and online knowledge. Compared with AOffL, a human solver must be available when the game starts, similar to what is required in a static relay attack. Moreover, there is no knowledge base for future attacking as the remote solver provides the required knowledge in real-time.

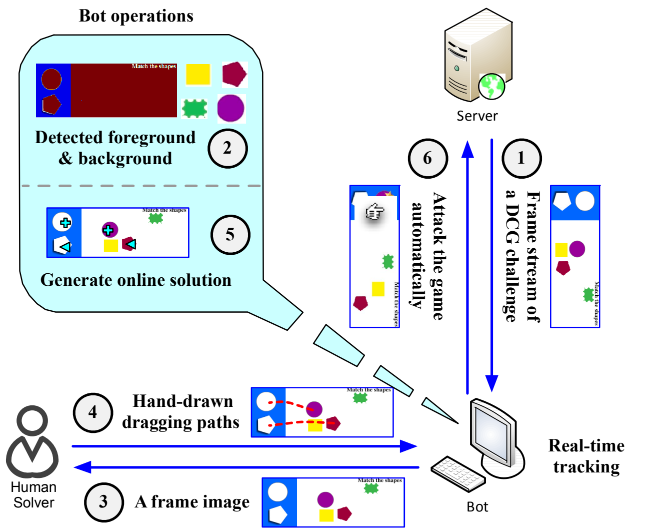
Bot operations
Detected foreground & background
2
Generate online solution
5
Server
6
Attack the game automatically
1
Frame stream of a DCG challenge
4
Hand-drawn dragging paths
Human Solver
3
A frame image
Bot
Real-time tracking

Fig. 8. Hybrid attack model II: Auto-attack with Online human Learning (AOnL).

As indicated in Figure 8, when attacking a game challenge, the bot keeps receiving frames from the server (Step 1), and performs initial analysis (i.e., detect background and foreground) on the game without drag-and-drop attempts (Step 2). Meanwhile, it sends one frame image to the solver once the game starts (Step 3). The solver performs the same operation as the learning phase in AOffL (Step 4). Once the solver submits his/her responses, the bot can learn the answer objects and the dragging paths for this particular challenge based on the initial analysis and the solver's response (Step 5), and complete the game automatically with the help of real-time tracking (Step 6). The interactive area of the GUI for the online solver is the same as the one in Figure 7(a). One concern in AOnL is that the success rate for completing an unseen game relies heavily on correctness and efficiency of the solver's response (the same concern underlies the relay attack on text-based CAPTCHA.) However, the usability study of DCG games indicates that honest users hardly make mistakes in playing these games. As we mentioned before, the drawing operation is very similar to the drag-and-drop in the real game. In fact, the task for the solver is likely even simpler due to the presence of a static image (rather than moving objects in frames). Therefore, it is fair to assume that the solver will perform the task very efficiently and without error most of the times.

5. IMPLEMENTATION & EXPERIMENTS

In this section, we evaluate the performance of our AOffL and AOnL attack models, both of which integrated with the CCH tracker. In AOffL, the time delay introduced by the remote human-solver does not have any impact on successfully attacking a game challenge that has been seen before. Similarly, in AOnL, we can assume that involvement of a real-time human-solver will not degrade the attack performance (time delay and accuracy) because the solver's task is very similar to playing the original game itself, and the usability study of such games [6] shows that humans easily solve the game quickly and accurately. As a result, in our experiments, we mainly focus on the performance of the automatic attack module in both models, given that the game has been seen before.

The attack module is tested on a Mac Air laptop with hardware configuration: 2 GHz Intel Core i7, 8GB 1600MHz DDR3, and 250G SSD; and software configuration: OS X 10.8.5, MATLAB R2012b. Each game prototype has 9 challenges as mentioned in Section 2.1. Each game challenge is tested three times, and the average measurement is taken as the final result.

Table 1. Completion time for the four games

		Mean Completion Time (s) (*std dev*)		
		# of Moving Obj. (# of Answer Obj.)		
Ship Game	**Moving Speed**	**4 (2)**	**5 (2)**	**6 (3)**
	1x	**2.53 (0.02)**	2.53 (0.01)	3.28 (0.04)
	2x	2.56 (0.02)	**2.71 (0.17)**	3.34 (0.06)
	4x	2.61 (0.12)	2.53 (0.09)	**3.25 (0.03)**
Parking Game	**Moving Speed**	**4 (1)**	**5 (1)**	**6 (1)**
	1x	**2.27 (0.11)**	2.13 (0.04)	2.23 (0.24)
	2x	2.38 (0.58)	**2.28 (0.17)**	2.34 (0.34)
	4x	2.22 (0.11)	2.18 (0.16)	**2.39 (0.16)**
Shape Game	**Moving Speed**	**4 (2)**	**5 (2)**	**6 (2)**
	1x	**2.57 (0.03)**	2.58 (0.00)	2.58 (0.01)
	2x	2.62 (0.01)	**2.62 (0.03)**	2.66 (0.02)
	4x	2.67 (0.06)	2.71 (0.02)	**2.73 (0.03)**
Animal Game	**Moving Speed**	**4 (3)**	**5 (3)**	**6 (3)**
	1x	**3.61 (0.05)**	3.64 (0.04)	3.74 (0.07)
	2x	3.68 (0.12)	**3.84 (0.04)**	3.90 (0.07)
	4x	3.78 (0.18)	3.99 (0.02)	**3.95 (0.10)**

The auto-attack module in both models, implementing our CCH algorithm, can achieve 100% success rate and 0% error rate per click for each game given that correct online/offline responses from human solvers are available. The experiment results shown in Table 1 indicate that, in general (as indicated by the diagonal cells in each table), the time cost of auto-attack module increases along with the increase of the number of moving objects and object moving speed. However, all tested game challenges can be completed successfully within 4 seconds, which indicates that with availability of game semantics (learned via AOffL or AOnL), auto-attack is faster than that of honest users (i.e., 4s vs. 11s). AOffL can utilize more time for extra computation in different tasks, such as processing extra tracking features besides color, and more complex but more robust tracking algorithms. In contrast, AOnL has more tolerance on the response delay caused by the network communication, or by the solver through offsetting the delay with the saving time in auto-attack model.

The two models of the hybrid attack framework have their respective advantages. AOffL can complete a known game efficiently and effectively, but it requires continuously updating the knowledge base for unseen games or answer objects. The delay issue in the manual learning phase is not a problem in AOffL due to its offline nature. Therefore, AOffL is a significant threat to DCGs that do not have a large database (e.g., manually extended database), but has a low tolerance on completion time. On the other hand, AOnL is insensitive to the database size. That is, it is possible for AOnL to complete a game challenge even if the game has never been seen before, largely attributed to the instant solution provided by the solver. However, response delay from the solver may be a bit of a bottleneck for AOnL. Therefore, AOnL could be a significant threat to those DCGs that has a relatively high tolerance on playing time.

6. CONCLUSION AND FUTURE WORK

We demonstrated a significant threat to broad classes of DCGs by proposing two novel hybrid attack frameworks, the Auto-attack with Offline human Learning (AOffL) and the Auto-attack with Online human Learning (AOnL), which provide a sweet spot between pure auto and pure relay attacks, in terms of computational efficiency, time synchronization and human-solver work-load. The experiment results indicate the robustness and effectiveness of the proposed models in significantly undermining the security of DCGs. In future research, the learning/attacking method on complex games involving dynamic target object as described in Section 5.1 may be explored. In addition, new categories of DCGs may be designed that could resist the current hybrid attack framework.

7. REFERENCES

[1] L. V. Ahn, et. al, "CAPTCHA: Using hard AI problems for security," *Advances in Cryptology—EUROCRYPT* 2003.

[2] S. Prasad, "Google's CAPTCHA busted in recent spammer tactics," Websens, 2008. http://web.archive.org/web/20080822032312/http://securitylabs.websense.com/content/Blogs/2919.aspx: .

[3] G. Keizer, "Spammers' bot cracks Microsoft's CAPTCHA: Bot beats Windows Live Mail's registration test 30% to 35% of the time, says Websense," *Computerworld*, 2008.

[4] J. Yan, "Bot, cyborg and automated turing test," *Security Protocols*, 2009.

[5] *are you a human*. http://areyouahuman.com/

[6] M. Mohamed et. al, " A Three-Way Investigation of a Game-CAPTCHA: Automated Attacks, Relay Attacks and Usability", ACM Symposium on Information, Computer and Communications Security (AsiaCCS), 2014.

[7] C. Zhang, et. al, "A multimodal data mining framework for revealing common sources of spam images," *J. of Multimedia,* vol. 4, 2009.

[8] M. J. Swain and D. H. Ballard, "Indexing via color histograms," in *Active Perception and Robot Vision*, 1992 .

[9] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, 1981.

[10] G. Welch and G. Bishop, "An introduction to the Kalman filter," UNC at Chapel Hill, 1995.