Modeling Human Behavior for Defense against Flash-Crowd Attacks

Georgios Oikonomou
Computer and Information Sciences
University of Delaware
oikonomo@cis.udel.edu

Jelena Mirkovic
Information Science Institute
University of Southern California
sunshine@isi.edu

***Abstract*—Flash-crowd attacks are the most vicious form of distributed denial of service (DDoS). They flood the victim with service requests generated from numerous bots. Attack requests are identical in content to those generated by legitimate, human users, and bots send at a low rate to appear non-aggressive — these features defeat many existing DDoS defenses.**

We propose defenses against flash-crowd attacks via human behavior modeling, which differentiate DDoS bots from human users. Current approaches to human-vs-bot differentiation, such as graphical puzzles, are insufficient and annoying to humans, whereas our defenses are highly transparent. We model three aspects of human behavior: a) *request dynamics*, by learning several chosen features of human interaction dynamics, and detecting bots that exhibit higher aggressiveness in one or more of these features, b) *request semantics*, by learning transitional probabilities of user requests, and detecting bots that generate valid but low-probability sequences, and c) *ability to process visual cues*, by embedding into server replies human-invisible objects, which cannot be detected by automated analysis, and flagging users that visit them as bots. We evaluate our defenses' performance on a series of web traffic logs, interlaced with synthetically generated attacks, and conclude that they raise the bar for a successful, sustained attack to botnets whose size is larger than the size observed in 1-5% of DDoS attacks today.

I. Introduction

A popular form of DDoS today are application-level floods, aka "flash-crowd attacks" [1], [2] that target application resources. Their name originates from a legitimate phenomenon, known as a "flash crowd," where many users simultaneously access a server because of some popular event. Attackers mimic this by deploying a large, distributed bot network and generating legitimate application requests that overwhelm the victim.

Flash-crowd attacks are extremely challenging because they request legitimate and business-critical content. Thus their traffic appears legitimate-like, which makes defenses that detect and filter malicious traffic ineffective against flash-crowd attacks. Further, attack bots can send at a low-rate or infrequently, leading to failure of techniques that monitor each client's rate and blacklist aggressive clients. The usage of many bots that send at a low rate is the main strategy adopted by today's attackers to bring down a service application.

This research was supported in part by the National Science Foundation under contract number 0430228. Views expressed in this paper are those of the authors and do not necessarily reflect views of the NSF.

A. Our Scope

Our goal is not to detect *all* bots but only those that may participate in flash-crowd attacks. Traditional DDoS attacks that flood a victim with junk traffic can be handled by existing defenses. Bots involved in legitimate activities (e.g., search bots) and in non-DDoS malicious activities (e.g., spamming) will likely be missed by our approaches. We play the attacker's need to generate a steady, high-rate flow of service requests to deny service, against his need to use stealthy bots to avoid detection. Our models detect all but the stealthiest of bots. This forces an attacker to either face detection and replace detected bots with new ones, or to increase stealth. Both these strategies increase the minimum botnet size needed for a successful, sustained attack. We show that the required size, when our defenses are active, is higher than the one observed in 95-99% of DDoS attacks today.

B. Related Work

Most popular defenses (e.g., [3]) against flash-crowd attacks use graphical puzzles like CAPTCHAs [4] to differentiate between humans and bots. A graphical puzzle is such that humans can easily solve it but machines cannot, e.g., recognizing numbers and letters on a distorted image. Only users who solve the puzzles get access to the service. While puzzles are useful, they are not sufficient. Puzzles are annoying to humans and thus must be served infrequently. Because a small rate is needed for a successful flash-crowd attack, a motivated attacker can solve puzzles himself on a few bots, then launch the attack. Some puzzles have also been solved by machines [5], and puzzle-breaking mechanisms are improving.

Jung et al. [6] identify features that can be used to detect flash-crowd attacks: small number of requested files, high and stable client rate, and highly distributed clients over subnets that did not previously communicate with the server. While these help detect DDoS *events*, they are not precise enough to detect *bots* involved in DDoS, which is our focus.

Kruegel et al. [7] build several models of user input for detection of Web server exploits, and one model has a slight resemblance to our request semantics approach. They model the probability of a single request's parameter sequence (e.g., in database queries), while we model the probability of *a sequence of requests*.

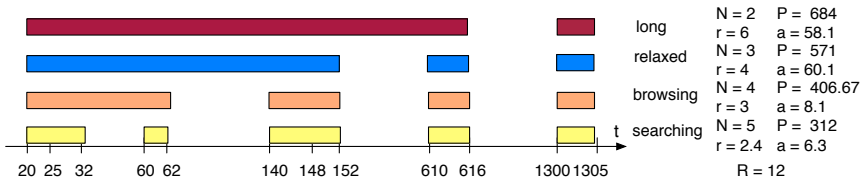

Fig. 1. Categorization of a user's requests into multi-level sessions

Honeytokens, as a more general form of our human-invisible objects, were proposed by Spitzner [8]. Gavrilis et al. [9] use decoy hyperlinks, as one form of human-invisible objects. Their decoys, however, can easily be detected by a bot via page source analysis while ours cannot. Gavrilis et al. examine request sequences to detect bots that request pages randomly instead of following links, while we also detect bots that follow links rarely followed by humans, even when bot accesses occur in a non-random manner. Park et al. [10] proposed several ways to detect bots such as looking for a lack of mouse activity, lack of Javascript rendering actions, random requests instead of link-following, and lack of style file and image file requests. Their methods have higher likelihood of false positives — e.g., when users disable Javascript — and false negatives — because bots can modify their behavior easily to fit required patterns.

C. Contributions

We propose three novel approaches to distinguish between flash-crowd bots and human users: (1) *Request dynamics* models that capture a human's interaction with the server via multiple parameters, and detect bot aggressiveness. (2) *Request semantic* models that learn common human request patterns, and detect bots that generate unlikely sequences. (3) *Deception* approaches that plant human-invisible items into server replies and blacklist addresses that request these items. All three defenses are transparent to humans, and can be engaged on each service request.

Our defenses are orthogonal and could be combined for a stronger protection. We envision they would be integrated with the application being protected, e.g., Web server. Because of a small but present false positive probability, defenses should be activated only when the server is overloaded to minimize collateral damage. We believe that a modest collateral damage is a small price to pay for protection against such a severe threat as flash-crowd attacks.

II. Assumptions

Current DDoS research has shown that majority of DDoS attacks involve botnets of a very limited size [11]–[13]. According to [13] an average bot has at most 100 Kbps bandwidth. Investigation of DDoS attacks at a tier-1 ISP [12] has shown that 80% of incidents generated an aggregate rate lower than 50 Mbps, 90% had lower rate than 80 Mbps, 95% lower than 100 Mbps and 99% lower than 200 Mbps. To be conservative, we assume that each bot uses only 1/10 of its available capacity and thus arrive at the following figures for botnet sizes: 80% of botnets have less than 5,000 bots, 90% have less than 8,000, 95% have less than 10,000 and 99% have less than 20,000 bots. With a less conservative assumption our performance results would be even better than presented in this paper.

In the rest of the paper we focus on attacks on Web servers, since we had access to public Web logs, needed to build and evaluate our models. We expect that our approaches could easily be extended to other types of servers. We assume that a Web server's resources can be exhausted with a 1,000-request-per-second attack [3]. This is a conservative limit and many servers today can support higher rates, but this only means that performance presented in this paper could be even better in real deployment.

III. Request Dynamics Model

We observe a user's interaction with a Web server as time-series of requests. A *request* is a human-initiated event, such as a click on a Web page. Requests for embedded objects, such as pictures on the page, are considered part of the original request. Looking at the distribution of request interarrival time from different Web logs we have noticed that values group at five segments: less than 10 seconds, 10-60 seconds, 60-300 seconds, 300-600 seconds and more than 600 seconds. We used this observations to introduce four session types: *searching* sessions, with pauses between sessions longer than 10 seconds, *browsing*, with pauses longer than 60 seconds, *relaxed*, with pauses longer than 300 seconds and *long*, with pauses longer than 600 seconds. For each session type we model the following features: number of sessions (N), average pause between sessions (P), average number of requests per session (r), and average request interarrival rate per session (a). For each user we also model total number of requests in the log (R). Figure 1 illustrates arrival of 12 requests for one user, their division into sessions, and calculation of our selected features.

We use a collection of Web server logs interlaced with synthetically generated logs of flash-crowd bots for training and testing with decision trees. Statistics of our logs are shown in Figure 2. Logs ISI1-ISI3 are obtained from USC ISI's Web server for three months in 2008, Logs OA1-OA2 are obtained from a small non-profit organization for two months in 2007. Logs WC1-WC4 are public logs of World Cup 1998 Web server for four days, with WC3 covering the day of the finals. Logs NS1-NS2 are public logs of a NASA Web server for two months in 1995. Logs CL1-CL2 are public logs of a busy ISP's Web server for two weeks in 1995. Public logs are available

	ISI1	ISI2	ISI3	OA1	OA2	WC1	WC2	WC3	WC4	NS1	NS2	CL1	CL2
duration	1 month			1 month		1 day				1 month		1 week	
reqs (M)	3.6	4.2	4.3	0.02	0.02	1.1	1.1	5.4	0.5	0.9	0.7	0.5	0.5
users (K)	190	210	221	3	4	58	57	175	26	81	75	87	92
reqs/user	19	20	19	7	5	19	19	31	19	11	9	6	5
leg bots (K)	18	20	19	1	1	0	0	0	0	0	0	0	0
att. bots (K)	9	9	10	13	10	11	10	10	10	9	10	8	10
att. reqs (M)	10.1	12.8	12.9	14.7	11.6	10.9	12.8	17.9	11.4	10.5	11.0	8.4	15.1

Fig. 2. Statistics of training and testing logs

train\test	ISI1	ISI2	ISI3	OA1	OA2	WC1	WC2	WC3	WC4	NS1	NS2	CL1	CL2
ISI1		0.02	0.02	0.00	0.00	0.24	0.24	2.04	0.49	0.15	0.11	0.04	0.02
ISI2	0.01		0.01	0.00	0.00	0.09	0.10	0.13	0.25	0.03	0.02	0.00	0.00
ISI3	0.01	0.01		0.00	0.00	0.01	0.02	0.11	0.06	0.03	0.03	0.00	0.00
OA1	2.46	2.62	2.33		0.08	12.44	10.88	20.69	9.38	6.22	4.57	1.84	1.68
OA2	1.66	1.76	1.58	0.01		9.89	8.65	17.73	8.37	5.91	4.37	1.99	1.83
WC1	0.04	0.05	0.05	0.00	0.00		0.02	0.33	0.04	0.04	0.04	0.00	0.00
WC2	0.04	0.06	0.06	0.00	0.00	0.02		0.10	0.03	0.07	0.06	0.01	0.01
WC3	0.04	0.06	0.06	0.00	0.00	0.00	0.01		0.01	0.03	0.04	0.01	0.00
WC4	0.04	0.05	0.05	0.00	0.00	0.03	0.02	0.07		0.07	0.05	0.01	0.01
NS1	0.12	0.21	0.17	0.00	0.00	0.12	0.15	0.82	0.26		0.02	0.00	0.00
NS2	0.10	0.18	0.16	0.00	0.00	0.08	0.09	0.74	0.24	0.01		0.00	0.00
CL1	0.12	0.20	0.18	0.00	0.00	0.07	0.09	0.26	0.18	0.03	0.05		0.00
CL2	0.17	0.22	0.21	0.01	0.00	0.24	0.16	1.12	0.24	0.06	0.09	0.01	

Fig. 3. False positives (0-100%)

train\test	ISI1	ISI2	ISI3	OA1	OA2	WC1	WC2	WC3	WC4	NS1	NS2	CL1	CL2
ISI1		0.00	0.00	0.02	0.06	0.01	0.01	0.00	0.01	0.01	0.01	0.01	0.01
ISI2	0.00		0.00	0.03	0.04	0.01	0.01	0.00	0.01	0.00	0.01	0.00	0.01
ISI3	0.00	0.00		0.02	0.04	0.01	0.01	0.00	0.01	0.00	0.01	0.00	0.00
OA1	0.00	0.00	0.00		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
OA2	0.00	0.00	0.00	0.00		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
WC1	0.01	0.00	0.00	0.06	0.04		0.01	0.00	0.03	0.00	0.01	0.00	0.01
WC2	0.00	0.00	0.00	0.02	0.00	0.00		0.00	0.01	0.00	0.01	0.00	0.00
WC3	0.00	0.00	0.00	0.01	0.00	0.00	0.00		0.00	0.00	0.00	0.00	0.00
WC4	0.00	0.00	0.00	0.02	0.02	0.00	0.01	0.00		0.00	0.00	0.00	0.01
NS1	0.00	0.00	0.00	0.02	0.06	0.01	0.01	0.00	0.02		0.01	0.00	0.01
NS2	0.00	0.00	0.00	0.04	0.03	0.01	0.01	0.00	0.01	0.01		0.00	0.00
CL1	0.00	0.00	0.00	0.05	0.06	0.01	0.01	0.00	0.01	0.00	0.01		0.01
CL2	0.01	0.00	0.00	0.10	0.06	0.02	0.01	0.00	0.02	0.01	0.02	0.01	

Fig. 4. False negatives (0-100%)

train\test	ISI1	ISI2	ISI3	OA1	OA2	WC1	WC2	WC3	WC4	NS1	NS2	CL1	CL2
ISI1		24	24	79	79	79	24	24	24	24	24	79	24
ISI2	46		46	116	53	46	46	101	46	53	46	46	46
ISI3	184	133		220	101	101	101	101	133	101	133	27	133
OA1	26	26	24		26	26	25	25	26	25	27	26	27
OA2	46	26	46	26		27	26	INF	25	27	26	27	26
WC1	34	34	34	34	67		56	34	34	67	56	88	34
WC2	99	88	26	88	25	88		88	88	27	88	88	133
WC3	233	79	233	233	INF	79	79		27	INF	79	79	79
WC4	26	26	26	26	26	89	24	24		26	30	30	24
NS1	46	79	46	220	79	46	44	79	46		46	46	46
NS2	79	72	79	79	72	79	79	79	79	72		79	72
CL1	46	46	46	46	46	46	46	46	46	156	46		46
CL2	126	233	233	102	102	102	233	335	126	335	233	126	

Fig. 5. Minimum number of bots (thousands) for a 10 min attack

from LBNL trace archives [14]. Where we had access to full content of the logs, i.e., in ISI and OA data, we have identified and removed search bots using known user-agent identification strings from http://user-agents.org/

Our attack model consists of generating flash-crowd bot requests synthetically and interleaving them with our Web logs. Synthetic generation allows us to study an entire range of stealthy behavior by trying all the possible combinations of attacking parameters, such as low rate, infrequent arrivals, long inter-session pauses, etc. Because each user is modeled separately, and there is no correlation among users, we generated only one user for each unique attack behavior. An attacker chooses the number of clicks to generate in a session between 5 and 50, uniformly at random. The interarrival rate of these clicks is also chosen uniformly at random between 1/10 seconds and 1/60 seconds. After a session, the attacker chooses the pause value uniformly at random between 60 and 300 seconds. Each attack ceases after an hour. Figure 2 shows the number of attackers and number of attack clicks inserted into each log. We emphasize that our synthetic attacks are much stealthier than attacks seen in the real Internet today. This creates a realistic, challenging data for evaluation of our system. Further, there are no public logs of flash-crowd attacks today so synthetic generation was the only possible evaluation approach.

Our attack parameters ensure that we generate very stealthy attack behaviors, to challenge the defense. Bots that behave more stealthily that these will be missed in our tests, but we show that the attacker will need larger botnets than our tests indicate ($>$ 25,000 bots) to perform successful attacks. Bots that behave more aggressively than those we tested, such as bots in contemporary attacks, will always be detected. While a bot may vary other aspects of their behavior (e.g., which files they request) it must exhibit features from our model — we thus claim that we exhaustively test the bot behavior.

Our measures of success are percentage of false positives and false negatives, and the minimum required botnet size to sustain the request rate of 1,000 per second for 10 minutes. According to [11], half of the DDoS attacks they observed last longer than 10 minutes. We calculate the botnet size by selecting each bot we missed during a given test and calculating how many similar bots we need for a 10-minute attack using the formula:

$$\text{MB} = 1,000 * (a + \frac{P}{r}) * \lceil \frac{600}{P_t * (N-1) + N * r * a} \rceil \quad (1)$$

where values a, P, N and r are calculated for searching sessions only, and P_t is the sum of pauses between those sessions. We report the minimum value per test data.

Figure 3 shows our false positives, on the scale 0-100%. We have colored values higher than 10% in pink, values 1-10% in orange, and values 0.1-1% in yellow. Cells with borders show results obtained when the testing and training data originate from the same server. We first observe that most of the values are less than 0.1% especially when the same server's data is used for training and testing. This is expected, since a user's interaction with a server depends on its content and more popular servers, such as WC, will see more dynamic usage. We also observe from Table 2 that OA, NS and CL data has fewer clicks per user (and overall) than the rest of the logs, indicating that these servers are not very popular. When this data is used for training and a more popular server's data is used for testing, we obtain higher numbers of false positives. This is especially the case for OA training data and WC test data. We conclude that the best approach is to use the server's past logs for learning behavior specific to its users' — a standard approach for many anomaly detection systems. Figure 4 shows the false negatives, on the scale 0-100%, with the same coloring scheme as Figure 3. Almost all the values are below 0.1%. Figure 5 shows the minimum botnet size for a 10 minute attack, rounded to thousands. We color fields with

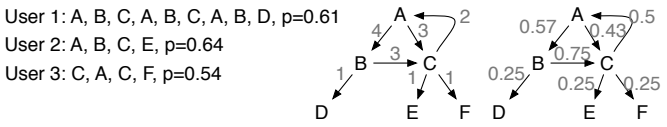

Fig. 6. A sample request graph built from access logs

values less than 20 K yellow, 20-50 K in pink, 50-100 K in orange, and $>$ 100 K in white. The lowest value shown in 25 K, which is higher than the botnet size observed in 99% of DDoS attacks today. This significantly raises the bar for an average attacker to launch a successful, sustained attack without it being detected and filtered.

IV. Request Semantics

Web pages have an abundance of links, many leading to content poorly related to the page's main topic, such as copyright notices. Humans rarely follow such links and human interests tend to coincide, making a few links on a page popular. A random-browsing bot cannot repeatedly hit popular links because they are a minority of all the links on a server's pages. A bot with a hard-coded request sequence may hit popular links only, but we expect popular sequences to be short forcing the bot into repetitions that are unusual for humans. A human may access a less popular link, but we show that this happens rarely, making the overall probability of human request sequences higher than that of a bot's.

We model link popularity via a *request graph*, whose nodes are Web pages and its directed edges are links connecting a page with a URL to the page where the URL points. We assume that the probability of each page being visited as the first page is 1, and we mine users' transitional (edge) probabilities from Web logs. For each edge we count the number of times it was traversed in the log file. The probability of an edge traversal can be estimated as the ratio of the edge's count and the sum of counts of all edges originating from the common parent node.

We define the *sequence probability* as the average of the first page's visit probability (1) and the sum of the edge traversal probabilities from the request graph. Figure 6 illustrates the calculation of edge and sequence probabilities for three user sequences, given a request graph. Any transitions that appear in testing or in the request graph, but not in training have zero probability. In our example in Figure 6 training and testing data is identical.

To build a request graph we needed to crawl a server's Web site, then populate the edge probabilities based on users' visit data extracted from the same server's logs. We could only do so for the ISI Web site since we had its recent Web logs that corresponded to its current content. We used the ISI1 log for training and the ISI2 and ISI3 logs for testing. We injected attacks into the test data by performing 10,000 walks on the request graph, starting from the main Web page, and following randomly chosen links. If we reached a dead-end we would return to the main page and continue. The length of the walk was chosen uniformly at random between 2 and

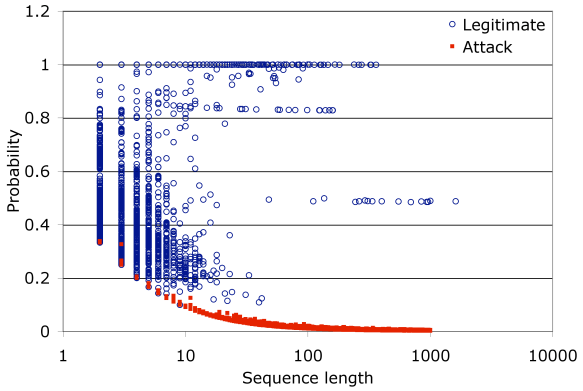

Fig. 7. Sequence probability vs length

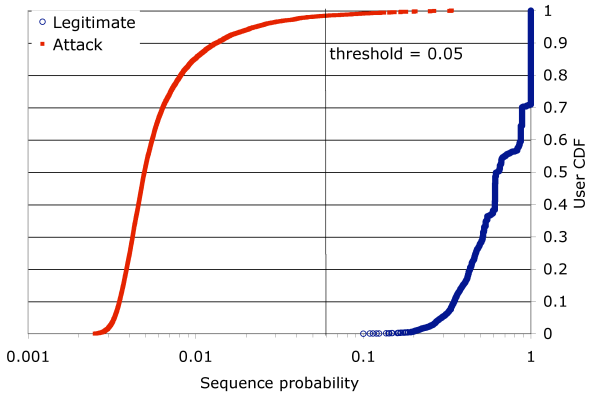

Fig. 8. User cdf vs sequence probability

1,000 requests, which corresponded to the length of human user request sequences in the training logs.

Figure 7 shows the sequence probability versus length for legitimate and attack sequences. While probability for both types of sequences declines with length, attack sequences have much lower probability for a given length than do legitimate sequences. Figure 8 shows the cumulative distribution function for legitimate and attack users versus sequence probability. All legitimate sequences had the probability greater than 0.05 but 98.06% of attack sequences had lower probability than 0.05. Thus we would achieve zero false positives and 1.94% false negatives with the 0.05 threshold. We then examined the cumulative distribution function of the average number of clicks a bot makes before it is detected. 92% of bots are detected before 24 requests, 99% before 31 requests and all before 48 requests. We calculate the minimum botnet size for a 10-minute attack by dividing 600,000 (total number of requests needed for the 10-minute 1,000-requests-per-second attack) by the largest number of requests a bot can make before being detected (47). Our results are 12.7 K for both ISI1 and ISI2, which is higher than the botnet size observed in 95% of DDoS attacks today. We conclude that this defense technique would significantly raise the bar for an average attacker, leading to successful attack filtering in 95% of today's incidents.

V. Deception

Our last approach detects bots by embedding invisible objects with hyperlinks or action triggers into Web server's replies, and flagging those users that request linked files or trigger action as bots. We have developed several deception techniques, described below, all with the goal to reduce or eliminate a possibility of automated detection of invisible objects via page source analysis by bots. All of these techniques are automated, and invisible objects to be placed in a reply are randomly chosen with randomly generated names. Placement of objects may be restricted by some criteria, but is random within a set of locations that fit that criteria.

1) Invisible images. Tiny images surrounded by a hyperlink are randomly placed close to visible text and images of the website. The color of the invisible image is randomly selected, and the file size is kept comparable to visible images by shrinking high-resolution graphics. These measures are necessary to minimize a bot's ability to differentiate visible from invisible images via page source analysis and simple image analysis.

2) Invisible forms. Forms are added, with elements such as textboxes, images and submit buttons shrunk in size so that they are invisible to humans. We randomly choose form element types and input variable names, and we also insert some hidden variables. All this makes a form invisible to humans but similar to visible forms for a bot doing source analysis. Users that click on form buttons are flagged as bots. Because mainstream browsers add a line break after each form, this technique slightly changes the layout of the original page. This is why we insert invisible forms only at the beginning or at the bottom of the page.

3) Tiny hotspots. Depending on the size of visible images, a number of 1-by-1 pixel maps are randomly generated and associated with hyperlinks. A sophisticated bot could parse the HTML code for pixel maps, extract their area size and detect deception. But such bots have not yet been seen in the wild.

4) Invisible text. We place on top of the page's images a text of the same color as the image's background, and associate it with a hyperlink. To avoid detection by bots via page source analysis we place the text on top of existing background images, instead directly on the page's background. A bot could only detect deception if it performed image analysis.

5) Layering. Different objects are made invisible by being placed between image layers. A sophisticated bot could parse the style files and deduce from differences in the z-level attribute the layer (and visibility) of objects. We address this by placing some "cover images" at the top layer with a shape that covers only those objects inserted for deception, but leaves the original page visible. Cover images are made invisible by making them of the same color as the page's background. Deception objects are placed close to the visible images so that the cover can be of a rectangular shape and its placement can be automatically calculated so that only deception objects are covered.

The proper approach to evaluate effectiveness of deception techniques would be to conduct a study with human users and bots. Since such studies take a long time to prepare and perform, we plan to do so in our future work. In the meantime we resort to estimating the effectiveness of deception techniques via a simple analysis. Let $p = \frac{\text{\# deceptive objects}}{\text{\# all objects}}$, be the density of deceptive objects on each page. The probability of a bot accessing a deception object after at most k requests is $P_d = 1 - (1-p)^k$ and the expected value of k is $1/p$. With $p = 0.5$, it takes $k = 7$ requests to catch a bot with 0.99 probability, and the expected value of k is 2. The attacker thus needs on the average $600,000/2 = 300,000$ bots for a 10-minute 1,000-requests-per-second attack. This is well beyond the reach of 99% of attackers today and significantly raises the bar for a successful, sustained attack.

Fig. 9. A portion of the Amazon's home page revealing deception

We illustrate our deception techniques 1, 4 and 5 in the Figure 9 by showing a modified homepage of Amazon.com with revealed deception. We use red colors and red dots to indicate where the deception objects lie. Without revealing, the new page is visually identical to the original one.

VI. Conclusions and Future Directions

We have proposed three defenses against flash-crowd attacks which differentiate humans from bots based on the uniqueness of human behavior with regard to (1) request dynamics, (2) choice of content to access and (3) ability to ignore invisible content. Our tests show that the proposed defenses significantly raise the bar for a successful, sustained attack. Botnets observed in 95-99% attacks today would be discovered and their traffic filtered by our techniques. We plan to investigate a combination of our approaches in the future, hoping to achieve synergistic effect and further improve performance. We also plan to implement our techniques in the Apache Web server and test them in two environments (1) with synthetic legitimate

and attack traffic to evaluate the server's overhead, (2) with human users to investigate likelihood of false positives.

REFERENCES

[1] K. Poulsen, "FBI Busts Alleged DDoS Mafia." [Online]. Available: http://www.securityfocus.com/news/9411

[2] J. Leyden, "East European Gangs in Online Protection Racket." [Online]. Available: www.theregister.co.uk/2003/11/12/east_european_gangs_in_online/

[3] S. Kandula, D. Katabi, M. Jacob, and A. Berger, "Surviving Organized DDos Attacks That Mimic Flash Crowds," in *USENIX Symposium on Network Systems Design and Implementation*, May 2005.

[4] L. von Ahn, M. Blum, and J. Langford, "Telling Humans and Computers Apart Automatically," *Communications of the ACM*, vol. 47, no. 2, pp. 57–60, February 2004.

[5] G. Mori and J. Malik, "Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA," in *Computer Vision and Pattern Recognition*, June 2003.

[6] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites," in *WWW'02*, 2002.

[7] C. Kruegel and G. Vigna, "Anomaly Detection of Web-based Attacks," in *10th ACM Conference on Computer and Communications Security (CCS'03)*, 2003.

[8] L. Spitzner, "Honeytokens: The Other Honeypot." [Online]. Available: http://www.securityfocus.com/infocus/1713

[9] D. Gavrilis, I. Chatzis, and E. Dermatas, "Flash crowd detection using decoy hyperlinks," *2007 IEEE International Conference on Networking, Sensing and Control*, pp. 466–470, April 2007.

[10] K. Park, V. Pai, K. Lee, and S. Cal, "Securing Web Service by Automatic Robot Detection," in *Proceedings of Usenix Annual Technical Conference*, 2006.

[11] D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity," Proceedings of the 2001 USENIX Security Symposium, 2001.

[12] V. Sekar, N. Dufield, O. Spatscheck, J. van der Merwe, and H. Zhang, "LADS: Large-scale automated DDoS detection system," in *USENIX Technical Conference*, June 2006.

[13] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "DDoS Defense by Offense," in *ACM SIGCOMM 2006*, Pisa, Italy, September 2006.

[14] L. B. N. Laboratory, "The internet traffic archive." [Online]. Available: http://ita.ee.lbl.gov/html/traces.html