Defending Web Servers Against Flash Crowd Attacks

Rajat Tandon, Abhinav Palia, Jaydeep Ramani, Brandon Paulsen, Genevieve Bartlett, Jelena Mirkovic
Information Sciences Institute, University of Southern California, Marina del Rey, CA, USA
{rajattan, palia, jramani, bpaulsen}@usc.edu, {bartlett, mirkovic}@isi.edu

***Abstract*— Flash Crowd Attacks (FCAs) are DDoS attacks that flood victim services, such as Web servers, with well-formed requests, generated by numerous bots. It is hard to detect and filter such attacks because both legitimate and attack requests look identical. In our previous work [1], we proposed models of how human users interact with Web servers, and also showed in simulation that these models can detect naive FCA attacks. We significantly extend these proposed models to make them more robust, simpler, and applicable to a wider variety of FCA attacks in this paper. We implement the models in a system called FRADE, and evaluate it on three Web servers with different server applications and different content. We show that FRADE can detect both naive and sophisticated bots within seconds and successfully filters out attack traffic. Therefore, FRADE significantly raises the bar for a successful attack by requiring attackers to deploy botnets that are at least three orders of magnitude larger than the botnets today.**

I. Introduction

DDoS attacks that target an application's resources are called Layer-7 attacks. The attacker floods a popular application with legitimate-like requests using a large number of bots. This results in a severe impact on the application, and impairs the server's ability to serve its legitimate clients. We propose FRADE, as a novel defense against FCAs to identify and blacklist malicious clients. It observes three main differences between humans and bots. First, humans browse in a bursty manner, as they alternate between searching for and reading the content that interests them, while bots usually attempt to maximize their request rate. FRADE learns the dynamics of human interaction with a given server over several time scales, and builds its *dynamics* model. Second, humans follow popular content across pages, while bots may access arbitrary content. FRADE learns content popularity over time, and builds its *semantics* model. Third, humans only visit content, which is visible when rendered, while bots may mine hyperlinks at random and visit invisible content. FRADE's *deception* module embeds invisible hyperlinks into the server's replies. Clients whose behavior mismatches FRADE's dynamics or semantics model, or who access deception hyperlinks are classified as bots and blacklisted, when the load on the server is high.

II. FRADE

Figure 1 shows the architecture of FRADE and how its modules interact. FRADE operates in two modes: learning and classification. During periods of normal load, it learns how human users interact with the Web server, which FRADE protects. It builds the semantics and dynamics models by monitoring Web server logs to learn how humans

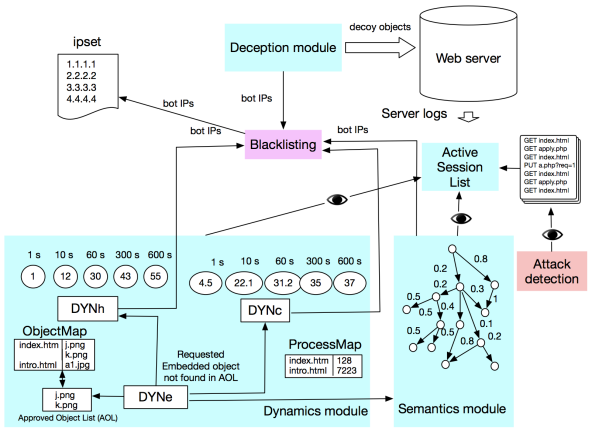


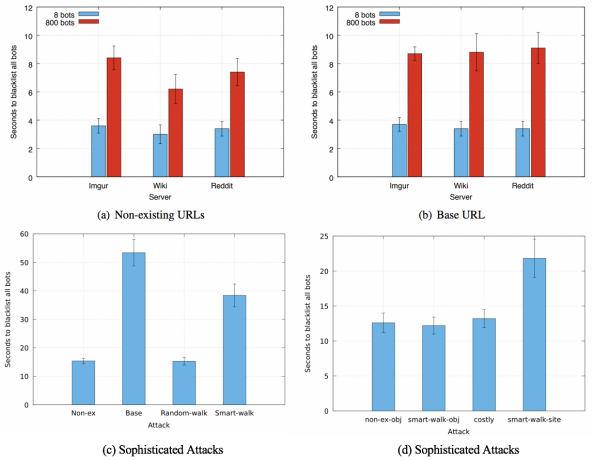
Fig. 1. Architecture of FRADE

interact with the server's resources. Deception objects, invisible to humans, are inserted in Web pages' source code. When a potential attack is detected, FRADE enters the classification mode wherein if a user's behavior deviates from one of the learned models, it is regarded as a bot and blacklisted.

Dynamics module: It models the rate of a user's interaction with a server in a given time interval. It contains three sub-modules: DYN_h, which models the rate of human-action requests, such as clicking on a hyperlink, DYN_e, which learns which embedded objects are associated with which Web pages, and DYN_c, which models the time it takes to serve a user's requests per a given time period. As humans browse in a bursty manner, we use multiple windows to learn DYN_h and DYN_e models at different time scales. A high percentile of these values learned is used as the threshold during classification. Clients that exceed one of these thresholds or that access embedded objects not associated with their previously accessed Web pages, are classified as bots.

Semantics module: It learns the probability of each sequence of requests generated by humans. We learn the probabilities for sequences of given length, and take a low percentile to be the threshold. We classify clients with the sequences probabilities lower than the learned threshold as bots. It may not see all the transitions during learning. Hence, it views Web pages as organized into groups of related content. It also learns transitions from pages to groups, groups to pages, and groups to groups. During classification, for missing page-to-page transitions we leverage page-to-group, group-to-page or group-to-group transitions.

Deception module: It follows the key idea of honey-



(a) Non-existing URLs (b) Base URL

(c) Sophisticated Attacks (d) Sophisticated Attacks

Fig. 2. Time to Blacklist Naive and Sophisticated Attacks

Server	Training			Testing		
	Users	HA	Emb	Users	HA	Emb
Wikipedia	243	5K	24K	107	2K	14K
Imgur	243	5K	35K	107	2K	15K
Reddit	243	5K	20K	107	2K	6K

TABLE I

TRAINING (LEARNING) AND TESTING (CLASSIFICATION) DATA AFTER CLEANING. HA=HUMAN-ACTION

tokens [2], special objects meant to be accessed only by attackers. The module inserts decoy objects, such as overlapping/small images, into a page's source code such that they do not stand out among other embedded objects in that page. We make these hyperlinks hard to identify from the page's source code by creating separate styles for them in the site's CSS file. We also craft the names of the pages pointed to by decoy hyperlinks, so that they are similar to the names of other, non-decoy pages on the server. During classification, users that access a deception object are classified as bots.

Blacklisting module: It learns IP addresses of clients that are classified as bots from other modules. It inserts firewall rules to filter all incoming traffic from those addresses. In our prototype, we use the `ipset` utility for the firewall.

A. Using a Proxy To Speed Up Detection

A server when overwhelmed under FCA, may not accept new connections. This slows down logging and delays FRADE's action. To speed up detection, we propose using the **Take-a-break (TAB)** proxy which completes the 3-way handshake with the client, logs Web page requests but drops each connection after the logging.

III. EVALUATION

We evaluate FRADE on the Emulab testbed [3]. We replicate content for a few popular Web sites: Imgur, Wikipedia and Reddit. We engage human users, using Amazon Mechanical Turk, to browse these websites and collect data for training and testing. Table I shows the number of users and requests that were split between training and testing. Our experimental topology includes 8 physical attack nodes (each emulating 1–1,000 virtual attackers), 1 legitimate client node (emulating 100 legitimate users), 1 node for the dropping proxy and 3 nodes for the servers. In our experiments, we replay human user data in a controlled environment maintaining 100 active, simultaneous virtual clients throughout the run. After 60 seconds, the virtual attackers start sending requests to the server at the aggregate rate of 8,000 rps. After 600 seconds we stop the attackers, and let the legitimate clients run for another 60 seconds. We measure how long FRADE takes to identify bots and blacklist them. We launch attacks with different bot behavior and botnet sizes. All tests are done with FRADE coupled with take-a-break proxy.

For our initial set of tests, we launch **Naive attacks**, which resemble today's attacks as noted by [4]. Attacks repeatedly request **non-existing URLs** or the **base URL**. We test botnets of 8 and 800 bots. FRADE blacklists all bots within 10 seconds on all the servers, as shown in Figure 2(a) and 2(b).

We also investigate **sophisticated attacks**, assuming an attacker familiar with FRADE. These attacks use a larger botnet (8,000 bots; 1 rps per bot) and smarter strategies to launch the attacks. The attack variants include: (s1) repeated requests for **non-existing URLs** that are identified as human actions, (s2) repeated requests for **base URL**, (s3) a **random walk** on the website graph, including all pages, (s4) a **smart walk** on the website graph, which is a random walk that avoids decoy links, (s5) requests for **non-existing-objects**, (s6) a **smart-walk-object**, which chooses at random from all embedded objects on the site, (s7) **smart-walk-site**, which is a smart walk including all non-decoy embedded objects one each page, and (s8) **costly** attack that repeatedly requests the web page, which is most costly to serve. Figure 2(c) and 2(d) show the time FRADE took to blacklist all 8,000 bots for the different sophisticated attacks on Imgur using TAB. 6 out of the 8 attack variants are blacklisted within 25 seconds. The other 2 variants, smart walk and base URL get blacklisted in 38 seconds and 54 seconds respectively. Attacks on other servers show a similar trend. In all our tests we had **zero false positives and false negatives**.

IV. CONCLUSIONS

FRADE models how human users interact with servers and detects bots as they deviate from this expected behavior. Our tests show that FRADE stops naive attacks and sophisticated attacks within seconds with a dropping proxy.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 13139215.

REFERENCES

[1] G. Oikonomou and J. Mirkovic, "Modeling human behavior for defense against flash-crowd attacks," in *2009 IEEE International Conference on Communications*. IEEE, 2009, pp. 1–6.

[2] L. Spitzner, "Honeytokens: The other honeypot," July 2003, https://www.symantec.com/connect/articles/honeytokens-other-honeypot.

[3] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*. Boston, MA: USENIX Association, Dec. 2002, pp. 255–270.

[4] D. Cid, "Analyzing popular layer 7 application ddos attacks," Sucuri blog, https://blog.sucuri.net/2015/09/analyzing-popular-layer-7-application-ddos-attacks.html.