In-Network Server-Directed Client Authentication and Packet Classification

Muhammad Jamshed
Electrical Engineering Department
KAIST, Korea
ajamshed@ndsl.kaist.edu

Jose Brustoloni
Computer Science Department
University of Pittsburgh, USA
jcb@cs.pitt.edu

***Abstract*—Defenses against Distributed Denial-of-Service (DDoS) attacks are commercially available and deployed by Internet Service Providers (ISPs) at the network and transport layers. However, attackers increasingly target vulnerabilities at the application layer. Launched from bots, these attacks seek to exhaust server resources, such as CPU and disk bandwidth. Because these attacks use normal-looking requests, ISP defenses can't distinguish them. We describe Forward Sentinel (FS), a novel device that enables ISPs to protect servers against such attacks. When load on a server reaches a level suggestive of attack, FS intercepts traffic and requires the server's clients to authenticate. Moreover, protected servers can signal to FS the desired class of service for a client's packets (e.g., after client authentication by the server). FS can be configured to mark packets for different classes of service or drop them according to the results of client authentication, number of packets forwarded, and server signaling. Experiments demonstrate that FS can effectively protect servers against DDoS attacks at the network, transport, and application layers.**

I. Introduction

The Internet does not authenticate clients' identities, or even IP addresses. Client authentication, if any, should be performed by servers. Moreover, the Quality-of-Service (QoS) that a client's packets receive, if not best-effort, is usually assumed to be a function of the client's Service Level Agreement (SLA) with its Internet Service Provider (ISP). Servers typically do not affect the QoS received by packets targeting them.

These design decisions make Internet servers vulnerable to Denial-of-Service (DoS) attacks. By the time a server has the opportunity to authenticate a client and drop or lower the QoS for processing the client's requests, the client will have already wasted network and server resources. If a server has enough malicious clients, it may have no resources left for legitimate clients' requests.

DoS attacks that exploit vulnerabilities at the network or transport layers, such as smurf and TCP SYN attacks, typically cause visible network anomalies. Effective defenses (e.g., [1]) have been developed against them and are now offered commercially by many Internet Service Providers (ISPs). Consequently, DoS attacks are becoming increasingly stealthy, using normal application-layer requests to waste server CPU or disk bandwidth. Launched from bots, i.e., compromised hosts under the control of an attacker, these attacks often mimic flash crowds[3]. ISPs are not able to distinguish them from other traffic.

This paper proposes and evaluates Forward Sentinel (FS), a novel device that enables ISPs to protect web servers from DDoS attacks at the network, transport, and application layers. FS can downgrade packets destined to a server according to client authentication by FS and signaling from the server. For example, FS can require a server's clients to solve CAPTCHAs when the server's response time reaches levels suggestive of DDoS attack. CAPTCHAs are perceptual tasks designed to be easy for humans and difficult for computers, including bots[4]. FS can be configured such that, after a client passes FS authentication, FS forwards only a limited number of bytes from the client before receiving a signal from the server. The server may use these bytes to authenticate the client using a second method that further limits bot-based attacks. For example, the server may use two-factor authentication with hardware tokens. Hardware tokens are difficult for bots to obtain and read. FS forwards at lower quality of service (QoS) or drops packets from clients that fail FS or server authentication. Because FS downgrades those packets before they waste network and server resources, FS can greatly improve server resilience against DDoS attacks.

II. Design

FS is installed between a web server or server farm and its clients. FS does not require modifications in client software and may be configured as a bridge, such that no network reconfiguration is needed. FS can provide security benefits without server software modifications, but coordinated FS configuration and server modifications enable greater resilience. FS protects from malicious exhaustion not only server resources but also network resources between itself and the protected servers. For example, if FS is installed at the ISP's side of a server's access link, botnets cannot induce DDoS by congesting that link, which often is a bottleneck.

FS uses SYN cookies[5] to thwart SYN attacks and attacks using spoofed client IP addresses. To mitigate other DDoS attacks, FS is designed so that it does not hold per-client state before having authenticated a client.

FS also monitors the response time of the servers it protects. If a server's response time consistently exceeds a configured threshold, FS considers the server to be under attack. When a server is under attack, FS forwards to it only client requests with a valid *FS cookie*. An FS cookie is a cryptographic function of its creation time, a nonce, and a secret known

only to FS. FS also keeps track of each valid FS cookie's client IP address, expiration time, and maximum number of simultaneous requests. Therefore, FS cookies cannot be guessed or replayed by other clients or outside its validity period.

When a server is under attack, FS replies with an *FS authentication challenge* to client requests without a valid FS cookie. The challenge is designed to be infeasible for bots to solve – e.g., a CAPTCHA[4]. When a client provides a valid response to such a challenge, FS redirects the client to the originally requested URL and provides an FS cookie to the client. The client retries its request, this time with a valid FS cookie. FS then opens a connection to the server, splices together the client and server FS connections, and forwards the request to the server and the response to the client.

If a client at a certain IP address repeatedly fails to solve FS authentication challenges, FS *blacklists* the client's IP address. FS drops packets from blacklisted addresses without processing their contents. Because blacklisted addresses might eventually be reused by legitimate clients, FS also estimates periodically how much load the blacklist is shedding from servers under attack. When the sum of each server's current and shed loads is less than the threshold for considering the server to be under attack, FS clears the blacklist.

To identify cookies when clients use HTTP/1.1 persistent connections or pipelining, FS performs packet scrubbing[6] and deep packet inspection. Usually, only the first segment of a request will contain a cookie. To thwart attacks using acknowledgments or other segments without an FS cookie, FS performs stateful filtering (i.e., FS drops segments whose sequence or acknowledgment number is inconsistent with connection state)[7].

By adding a *CoS cookie* to an FS cookie in a response, a server can signal to FS what class of service (CoS) the server desires for connections using that FS cookie. The values of an FS cookie and a CoS cookie can be distinguished by being placed between the symbols "fs=" or "cos=", respectively, and "&" or new line. A CoS cookie may, e.g., specify that the desired class of service is High, Low, Expire, or Blacklist. FS forwards to a server a request using an FS cookie of High or Low CoS only if the request also contains the respective CoS cookie. If the CoS is Expire, FS immediately expires the current FS cookie and the client would need to reauthenticate before it can reuse the web service. If the CoS is Blacklist, FS blacklists the client's IP address: all packets from the IP address are dropped until FS is instructed otherwise by the server.

On a connection with valid FS cookie but no CoS cookie, FS may be configured to forward to a server only a limited number of bytes (called *preamble*), at a configurable default CoS (e.g., High). A server may use the preamble to further authenticate the client. For example, FS may initially authenticate the client using a SYN cookie and then a CAPTCHA, and then the server may further authenticate the client using password and hardware token. If the client does not pass the server's authentication, the server may decide, for example, (1) to serve

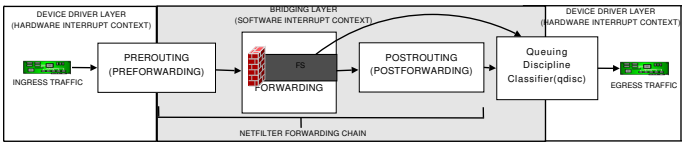

Fig. 1. Netfilter API Showing FS

the client at a Low CoS, (2) to expire the client's FS cookie, so that the client has to retry FS and server authentication, or (3) to blacklist the client, so that the client cannot retry authentication. These layered defenses can make it impractical for bots to exhaust resources used by authenticated High CoS clients.

To signal CoS to intermediate nodes (e.g., routers), FS and servers also mark the DSCP (differentiated services codepoint) field in IP headers[8]. In the case of server farms, some servers may be dedicated to High or Low CoS requests. If so, FS may be configured to forward requests directly to servers of the respective CoS. This solution is particularly attractive if the servers' operating system is not CoS-aware (e.g., lack resource containers[9] or signalled receiver processing[10]).

III. Implementation

We developed an FS prototype in Linux as an ebtables-compliant module. Linux provides a netfilter library for inserting customized forwarding modules in its network or link layers. Such modules are respectively called iptables- or ebtables-compliant. Implementing FS at the link layer, as we did, can be advantageous for two reasons. First, forwarding decisions are made earlier and with less overhead than would be the case at the network layer. Second, the resulting device can be installed in existing networks without network reconfiguration.

Our FS module is hooked as a jump target in the FORWARD chain of ebtables. Fig. 2 illustrates its components and how the client frames pass through several submodules before FS decides whether to drop or forward them to the protected web servers.

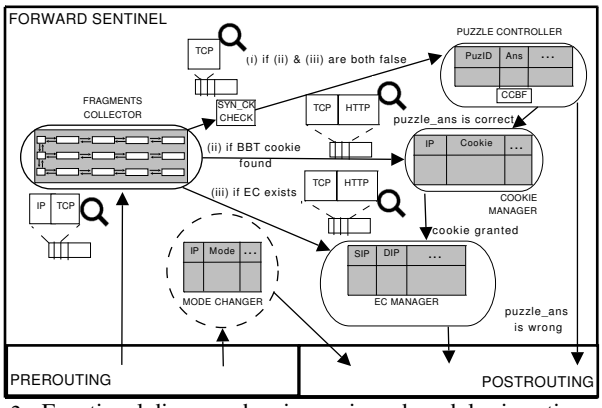

Fig. 2. Functional diagram showing major submodules in action when one of the registered servers behind FS is under a *suspected_attack* mode

IV. Performance Evaluation

This section reports the results of experiments testing FS's performance and scalability under a variety of DDoS attacks.

A. Experimental Setup

Figures 4 and 5 illustrate the testbeds used in our experiments. Our experiments compare FS's performance when FS is installed (i) in customer premises, as a bridge in front of protected servers (*CPE*), vs. (ii) in the ISP side of the protected servers' access link (*outsourced*). The testbed's link x was configured to emulate CPE with bandwidth of 1.5 Mbps, a maximum latency of 70ms and burst of 8KB using Linux's *traffic classifier's* (tc) token bucket filter. On the other hand, for emulating the outsourced configuration, we set x's bandwidth to 1 Gbps.

We wrote a simple test program, `authclients.c`, that uses aliased IP addresses to send authenticated (i.e., legitimate users') HTTP GET requests to a web server at a user-specified rate (in our experiments, one request per second for ten seconds). When the test program terminates, it prints the response time for each request. The response time may include client authentication by FS.

In the first two experiments, we set the service times of HTTP GET requests to 10 ms plus the time to transmit an 8 KB web page. We report results using HTTP digest for client authentication. Results using CAPTCHAs were similar. To emulate WAN conditions (specifically, queueing and transmission delays totaling 100 ms), we used netem[11]. Unlike the first two experiments, the third and fourth experiments test FS defenses that require server software modifications.

B. Results

The first two experiments were conducted in testbed 1 and compare FS's resilience against some popular DoS attacks. The remaining experiments were performed in testbed 2 and test coordination between FS and server authentication and consequent classification of client packets.

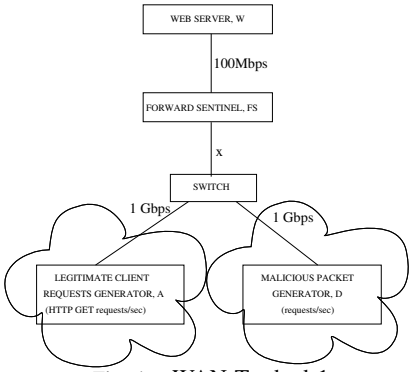

Fig. 4. WAN Testbed 1

Fig. 3a shows when **D** ran a DoS traffic generator that used asynchronous TCP sockets to send HTTP GET requests with aliased IP addresses. To allow transmission of packets at a steady rate, **D**'s kernel was updated to allow up to $24,000$ file descriptors per process. We also installed a fragroute daemon[12] in **A** with fragment size set to 96 bytes. Under this testbed FS withstands well against such fragmentation attacks (note that, in this test, FS cookies were roughly in the fourth fragment of the first datagram of each HTTP request). Moreover, the figure shows that outsourced FS outperformed CPE FS also in this case. The next experiment evaluates the effectiveness of server signaling to FS, when bots pass FS but not server authentication. This scenario could occur when a botmaster breaks FS's authentication puzzles(*e.g.* CAPTCHAs) but not the server's(*e.g.* hardware tokens).

This scenario was emulated using WAN testbed 1 with a CAPTCHA authentication scheme installed in FS. `authclients.c` was configured to send CAPTCHA answers to FS. For experimental purposes, FS's CAPTCHA puzzle module was defeated by reducing its library to only one puzzle entry. Thus, the automated `authclients.c` tool could always answer FS's puzzle.

We also created a custom Apache server application that authenticates prospective clients using the HTTP digest scheme. This authentication ran in the primary web server **W1**. High priority clients, after W1's authentication, are immediately granted access. On the contrary, bots fail to pass server W1's authentication tests. Consequently, W1 informs FS to blacklist bots' IP addresses.

We set the web page size and and the average service times of the client connections to 4KB and 100 milliseconds, respectively.

Results in Fig. 3b show that outsourced FS thwarts the bots' attack thoroughly, even when bots defeat FS's authentication, if bots don't pass server authentication. CPE FS also reduces the effects of such attacks, although not as much as does outsourced FS. Experiments testing FS packet classification

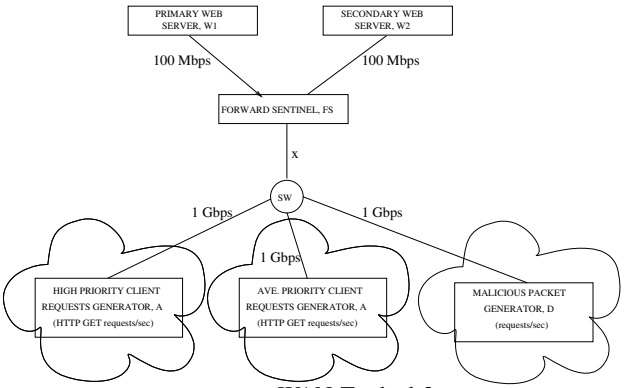

Fig. 5. WAN Testbed 2

were conducted in testbed 2. We again used FS's CAPTCHA authentication module. While `authclients.c` was used in **A**, **B** used a modified `authclients.c` version which can vary HTTP GET transmission rates. **A**'s traffic was set to high-priority, while **B**'s was set to regular priority. `iproute`'s `tc` was utilized to prioritize outgoing traffic in FS. The tests were conducted using the Priority FIFO (PFIFO) scheduling scheme, which uses separate queues for **A** and **B** based on Assured Forwarding (AF) classes.

Initially, both the servers were in *normal* modes and the clients in **A** and **B** accessed the primary web server to get their pages. However, D's malicious requests put the server farm in *suspected_attack* mode. Since the primary web server W1 is FS- and QoS-aware, it authenticates and prioritizes prospective clients with a second authentication scheme. We installed a custom Apache server which authenticates and

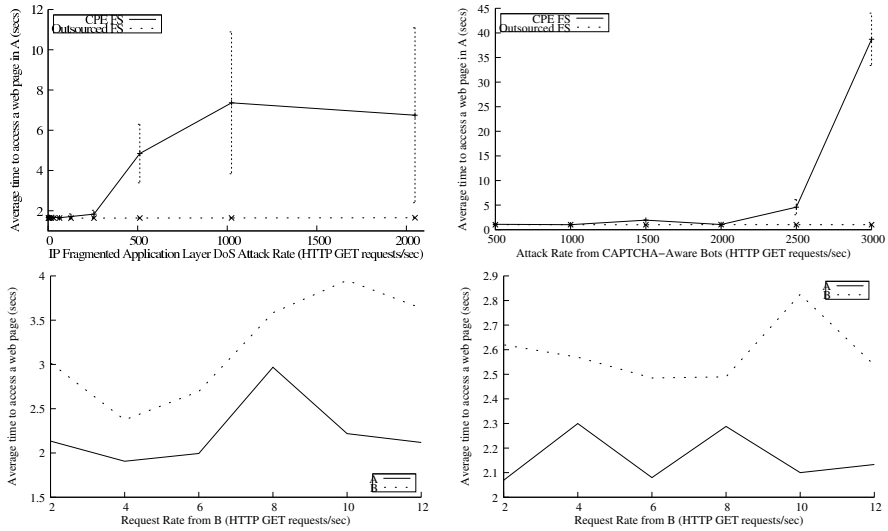

Fig. 3. (a) Application-level DDoS attack I(Exp. 1), (b) Application-level DDoS attack II(Exp. 2), (c) PFIFO scheme when FS is outsourced(Exp. 3) and (d) PFIFO scheme when FS is in CPE bridge(Exp. 4). **FS** machine is Intel Pentium 4 2.4 GHz (256 MB RAM). **W/W1** machine is Intel Pentium 2 400 MHz (128 MB RAM). **W2** is Intel Pentium 3 600 MHz (128 MB RAM)

prioritizes clients based on the HTTP digest scheme. High-priority clients, after the server's authentication, are immediately granted access to the web page in W1, while regular-priority clients are redirected to the secondary server **W2**.

Fig. 3c shows the response times for high- and regular-priority clients when servers were under an attack rate of $2,000$ HTTP GET requests/sec and FS was outsourced and configured for PFIFO marking. Fig. 3d similarly shows response times for high- and regular-priority clients when servers were under an attack rate of 20 HTTP GET requests/sec and FS was configured as CPE with PFIFO marking. (The attack rate used was much less in the latter case because CPE FS withstands network DDoS attacks less well than does outsourced FS.) In both configurations, the figures show that FS correctly gave high-priority clients lower response times.

V. Related Work

DDoS attacks mitigation has been an area of active research. Forward Sentinel's closest match is KillBots[3]. However, it overcomes several limitations and refines algorithms and data structures used in Kill-Bots. Kill-Bots in only compatible with legacy HTTP/1.0 connections. On the contrary, FS works correctly with HTTP/1.1, because it supports persistent connections and pipelining features. Kill-Bots also implicitly assumes that attackers do not spoof fields other than the IP address or maliciously fragment packets. On the contrary, FS performs stateful packet filtering and packet scrubbing to avoid such attacks. Also, unlike FS, Kill-Bots drops requests from blacklisted IP addresses even after malicious clients stopped using them.

VI. Conclusions

We described and evaluated Forward Sentinel, a novel device that similarly distinguishes malicious clients' requests, but does so in the network, upstream both of server and network bottlenecks targeted by attackers. FS uses lightweight mechanisms to signal to servers whether each packet's origin has been authenticated to be human, and to drop or classify clients' packets in accordance with further authentication by servers. Because FS needs to perform client authentication in the network without modifications in clients or servers and at line rates even during DDoS attacks, its implementation presents many challenges. Our experiments demonstrated that FS's client authentication and server signaling mechanisms effectively preserve server availability for legitimate clients during botnet-based attacks. FS's mechanisms are flexible, coordinate well with protected servers' own client authentication mechanisms, and add little overhead.

References

[1] Cisco (2009). Cisco guard. http://www.cisco.com/en/US/products/ps5888.

[2] PowerTech Information Systems AS (2009). Smurf Amplifier Registry. http://www.powertech.no/smurf/.

[3] Kandula, S., Katabi, D., Jacob, M., and Berger, A. (2005). Botz-4-sale: Surviving organized DDoS attacks that mimic flash crowds. In *Networked Systems Design and Implementation (NSDI '05)*. USENIX.

[4] von Ahn, L., Blum, M., Hopper, N., and Langford, J. (2003). CAPTCHA: Using hard AI problems for security. In *Eurocrypt '03*.

[5] Bernstein, D. (1996). Syn cookies. In *SYN-Cookie Archives (http://cr.yp.to/syncookies.html)*.

[6] Handley, M., Paxson, V., and Kreibich, C. (2001). Network intrusion detection: Evasion, traffic normalization and end-to-end protocol semantics. In *Proc. 10th USENIX Security Symposium*.

[7] van Rooij, G. (2001). Real stateful TCP packet filtering in IP filter. In *10th USENIX Security Symposium*.

[8] Nichols, K., Blake, S., Baker, F., and Black, D. (1998). Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. In *RFC 2474 (http://tools.ietf.org/html/rfc2474)*.

[9] Banga, G. and Druschel, P. (1999). Resource containers: A new facility for resource management in server systems. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI '99)*. USENIX.

[10] Brustoloni, J., Gabber, E., Silberschatz, A., and Singh, A. (2000). Signaled receiver processing. In *USENIX Annual Technical Conference (USENIX '00)*. USENIX.

[11] TheLinuxFoundation (2009). Net:netem. http://www.linuxfoundation.org/en/Net:Netem.

[12] Song, D. (1998). Fragroute. http://www.monkey.org/ dugsong/fragroute/.