Flow-Cookies: Using Bandwidth Amplification to Defend Against DDoS Flooding Attacks

Martin Casado, Pei Cao
Stanford University

Aditya Akella
University of Wisconsin, Madison

Neils Provos
Google, Inc.

Abstract

Distributed Denial-of-Service flooding attacks against public web servers are increasingly common. Websites without the ability to over-provision or rely on a CDN are often overwhelmed by such attacks. Existing proposals to combat flooding within the network either require substantial changes to the Internet infrastructure (e.g., Capabilities [27, 26]), or the difficult task of identifying attack aggregates near the core (e.g, Push-back [18]).

In this paper, we present an easy to deploy mechanism whereby a third party with high access to bandwidth can protect a web server against bandwidth exhaustion from illegitimate traffic. With this mechanism, all traffic to and from a web site is routed via a third party managed middlebox. The middlebox provides two simple functions: (1) determine if a TCP packet sent to the web-server belongs to a legitimate flow (i.e ., belongs to an already established connection, or originates from a non-spoofed IP address), and, (2) filter traffic from IPs blacklisted by the protected server. We show that this dual functionality can be realized in a completely stateless fashion using "Flow Cookies", a simple extension to SYN cookies[13], wherein the protection middlebox places a secure, limited lifetime cookie within the TCP timestamp of every outgoing data packet from the protected server. Flow-cookies offers strong protection against flooding, does not require modification to clients or to the network, and is resistant to source spoofing.

We have implemented flow-cookies within an existing software router and verified its compatibility with popular client operating systems, and widely used public web sites.

1 Introduction

Distributed denial of service (DDoS) attacks against web sites are an Internet reality. Not only are they a disturbingly common occurrence [16, 6], but they have been successful in widespread disruption of online commerce servers and other sites critical to our daily lives [8, 3, 9]. As a result, defending against DDoS has been an active research area in the networking community.

DDoS attacks against web sites can be partitioned into two broad categories: attacks on resources under the web sites' control, and those on resources not under the web sites' control. The former include attacks on web server computation resources, bug exploits of application software, exhaustion of TCP connections, hogging the servers outgoing bandwidth via massive *wgets*, etc. Fundamentally, defending against these attacks is a web site's responsibility. Active research in this area has yielded a variety of techniques for web sites to discern legitimate requests and protect them such as sophisticated bot-detection [23], graceful degradation under heavy load [25] and resource scheduling schemes [22].

However, web sites are powerless against the latter category of attacks. In particular, if the incoming network link to a website is filled with attack traffic, there is little that the web site can do. Indeed, a popular method of DDoS is flooding. In the common case, such flooding attacks on websites only consume tens of Mbps or, less frequently, hundreds [6]. However, if the attack target's incoming link is unable to handle such traffic, the network must take active steps to filter the attack. Unfortunately, in-network active filtering and aggregate signature detection for low bandwidth floods are notoriously difficult and prone to source spoofing. Also, in cases where the site relies on dynamic or otherwise deep content (e.g. online gaming sites, or siets with large backend databases), offloading traffic to a CDN [1, 17] is not possible.

In general, network elements, such as routers, lack the semantic knowledge to determine what level of activity is legitimate and what is not for a given website. For example, a web site may allow a high rate of requests from some IP address (e.g. its business partners) but not others. Therefore, the capability to discern normal from abnormal, or legitimate from illegitimate use, rests with the website. A website simply needs a mechanism to *push* these decisions deep into the network, where traffic deemed malicious can be filtered at high speeds. Our proposal, "Flow Cookies", provides one such mechanism.

"Flow-cookies" is a lightweight, low-state mechanism that provides legitimate flow detection and reliable filtering at very high speeds. In this approach, a third party provider installs a "flow-cookies" enabled middlebox, called the cookie box, with high bandwidth link(s) to a tier-1 ISP(s). All traffic, to or from the protected webserver, must traverse the cookie box. The protected websites are guaranteed that all packets received from the box belong to a legitimate TCP flow. Further if a web-server deems a client to be misbehaving, it can request the cookie box to filter the offending IP.

The cookie box can provide many collaborating web servers with the protection bandwidth of the attached link(s). "Flow-cookies" does not require per-flow state at the cookie box and is resistant to the first-packet flooding problem [11]. Flow-cookies is designed to handle the common case of relatively low-bandwidth attacks (e.g.several tens to a few hundred Mbps). Compared to standard IP-address based filtering, flow-cookies enjoys non-spoofable attack recognition, and guarantees that only "authenticated flows" can reach the web site. Compared to capability-based schemes, flow-cookies does not require a separate capability setup step, operates from a point-deployment, is completely backward compatible and does not require any change to clients' operating systems or browser applications. Unlike CDNs or other aggressive caching or content distribution strategies, flow-cookies fully supports dynamic content and does not require sensitive data to be replicated within the network. Therefore, it is a practical solution that can be immediately employed by public web sites today.

We have implemented flow-cookies in software and tested our implementation using live connections between various commodity client operating systems (WindowsXP, MacOSX, NetBSD, Linux2.4, Linux2.6) and multiple popular, public web sites. Our implementation is able to operate at gigabit speeds including per-packet IP filtering of millions of addresses. We also found our approach to be very effective against high volume SYN flooding attacks.

The rest of the paper is organized as follows. In the next section we present related work. In Section 3, we provide an overview of flow-cookies. In Section 4 we describe the flow cookie protocol in detail. Section 5 discusses the properties of flow-cookies and possible attacks that flow cookies cannot handle. Section 6 describes our implementation and evaluation of flow cookies. Finally, we conclude and present future work in Section 7.

2 Related Work

SYN-Cookies. SYN-cookies [13] are designed to prevent SYN floods from exhausting server connection state with half-open connections. They operate by using a cryptographically secure cookie in place of the ISN in the SYN|ACK from the server. If the subsequent ACK from the client contains a valid cookie, then connection state is allocated at the server. Flow-cookies generalizes SYN-cookies by requiring all client packets (except the initial SYN) to contain a valid cookie. In our scheme, SYN-cookies are used for connection establishment and "flow-cookies" are placed in the TCP timestamp field of all subsequent data packets (details in Section 3).

Filtering and Capability-based approaches. Two classes of solutions have been proposed to handle bandwidth exhaustion DDoS: filter-based and capability-based.

Filter-based approaches, such as Pushback [18] and AITF [12] require the identification and blocking of illegitimate traffic from within the network. In Pushback, a router attempts to identify attack traffic by determining that it is in a congested state, finding an “aggregate” that describes the attack traffic, and pushing the “aggregate” upstream to be blocked close to the source. In pushback routers must discern legitimate traffic from the illegitimate; In contrast, our solution lets the web server decide.

AITF is an optimization of Pushback that is based on the observation that a pure filter-based approach requires too much state in core routers, AITF decentralizes the state by pushing filtering rules as close to the source as possible. However, it does not consider how the filtering rules are determined and whether the attack recognition mechanism is resistant against spoofing.

Flow-cookies can be viewed as a simplified variant of network capabilities [27, 26], which strive to offer “complete path protection”. With capabilities, clients first receives a capability from the server which must accompany each subsequent data packet. Current capability proposals call for modifications to the IP layer or the addition of a shim layer and infrastructural changes to the network. In contrast, flow-cookies requires no modification to routers, nor to IP or TCP. Also, flow cookies aims for “partial path protection”, and trusts one end of the communication: the public web sites. The less-ambitious goals of flow cookies allow it to be simpler and easier to adopt.

Overlays. A few approaches based on overlay networks [19, 10] have been proposed for protecting online servers against DDoS: Protection is enforced by only allowing a small set of IP addresses (whose value is unknown to the attacker) or nodes to communicate with the server. These nodes belong to a secure overlay. To access the overlay, a client must validate itself at predetermined entry points. In contrast, flow cookies provides a webserver the ability to accept connections from unknown clients and then enforce immediate filtering if a client is deemed malicious.

Commercial solutions. Many large web sites use CDNs [1] to diffuse DDoS attacks. CDNs are mainly employed for protecting static content. They are seldom used for content involving user information–quite common to e-commerce sites, which are often victims of DDoS attacks.

DDoS solutions relying on the deployment of security appliances, such as [4, 5], perform a number of DDoS prevention functions such as offloading the three-way handshake with SYN cookies, enforcing per-flow rate limiting of millions of flows and performing stochastic anomaly detection of DoS attacks, while maintaining substantial state per flow.

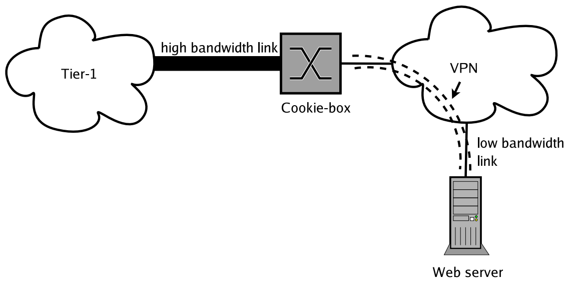

Figure 1: With flow-cookies, a webserver cooperates with a "cookie-box" connected to a high bandwidth link. The cookie-box provides high-bandwidth protection for the web-server by filtering all blacklisted IPs and only allowing packets that are part of legitimate flows to pass.

Flow-cookies takes a different approach: it allows the end-server to distinguish good vs bad traffic and push the filtering decisions to the network rather than rely on sophisticated in-network detection algorithms. Another distinction is that flow-cookies does not require per-flow state lowering the relative complexity and cost required for implementation.

3 Our Approach

Flow-cookies operates using a stand-alone device—which we call the flow-cookie box, or cookie-box for short—deployed by a third-party service provider. We show an idealized setup in Figure 1.[1] We assume that the cookie box is deployed in a data center with a very high speed connection to a tier-1 ISP. Note that the cookie box could be deployed at a location far removed from the web site. Furthermore, the box could be multihomed to several tier-1 ISPs.

All traffic to and from the protected web-server must traverse the cookie box. The cookie box maintains a private channel to each back-end web-server it is protecting. In practice this would likely be a relatively lower bandwidth IPSec tunnel (compared to the link between the cookie box and its ISP) or, for more severe threat environments, a lower bandwidth leased line (for co-located clients). The cookie box announces routes to its protected servers onto its upstream ISP(s) (To be specific, the third-party service may own a public IP-block and assign IP addresses to its customer web-servers from it). This ensures that incoming client traffic is routed through the box.

[1]While in the context of this paper we assume that flow-cookies is implemented in a stand-alone device, it is simple enough that it could be added as extra functionality to a line card on an access ISP router.

The cookie box and the web-servers cooperate to perform the following four tasks:

1. All incoming clients complete their 3-way handshake with the cookie box. The box uses SYN cookies for the handshake. SYN cookies does not require state maintenance at the cookie box and can be run at gigabit line speeds [4]. This step ensures that *SYN floods cannot traverse the link between the cookie box and the protected website.*

2. Once a client has completed the handshake, the cookie box hands off the connection request to the website using TCP-handoff [7].[2]

3. For outgoing packets from the web-server to its clients (this happens after the web-server has accepted the client connection request), the cookie box adds a secure "flow-cookie" (explained below). The flow-cookie is echoed back by the client, and checked by the cookie box. This is to ensure that *only packets belonging to flows accepted by the server traverse the link between the cookie box and the web server*.

4. The webserver understands the accepted usage policy of its local administration, and is already keeping per-flow state for each outstanding connection. Therefore, *the backend server is in the best position to determine if a client is misbehaving.* IPs (and associated ports) deemed malicious by the webserver (e.g. deviant flows) are passed to the cookie box, which filters current and future packets from the offending clients.

Flow-cookies. To be backward compatible, we exploit the *TCP timestamp* option, and place the *flow-cookie* from step #3 in the timestamp field of packets. The TCP timestamp option, proposed in RFC-1323 to measure RTTs, is supported by all the major host operating systems. According to the RFC, once the option is enabled by both ends, the sender places a timestamp in a packet, and subsequent packets from the receiver *echo* the timestamp.[3] On connection set up, the cookie box negotiates the timestamp option with the client.

Flow-cookies are valid for a limited period, are non-forgeable, and are computed on the basis of the IP and port of the client. The cookie-box verifies that all packets contain a legitimate cookie thus ensuring that only packets from clients accepted by the server are forwarded. All others are dropped. Therefore the cookie

[2]Instead of handing the connection directly to the website, the cookie box may hand it off to a participating layer-7 switch which can then load-balance across multiple servers, as is common today.

[3]Common operating systems enable timestamps by default, with the exceptions of Windows2000 and WindowsXP. As has been shown by [24], it is possible to trick Windows into echoing timestamps by including a timestamp option in the SYN|ACK packet.

box provides protection proportional to its line-speed for one or more backend servers which, themselves, have low speed connections.

Note that flow cookies require *no modification of clients*. The cookie box maintains *no state* during normal operation (however, the box may store IPs to filter upon a webserver's request). All per-connection state is maintained at the web server, as is usual. In effect, flow cookies simply provides a stateless, in-band mechanism for web-servers to signal acceptance of clients to the third-party service.

Unlike approaches common in the commercial world, flow cookies is not meant to be deployed as a "perimeter solution" at web-sites. In contrast, we call for traffic vetting at a remote location with high speed connection. This helps flow cookies *leverage the high bandwidth and filtering capacity of the the link(s) between the cookie box and its ISP(s)* to protect the link (or path) between a cookie box and the web site from SYN floods, connection floods, and bandwidth-hogging deviant flows.

4 Protocol Details

In this section, we present additional details of our approach starting with a description of the actions taken by the cookie box. A cookie box must maintain the following state:

- S_r: A secret known only to itself (128 - 1024 bits).
- C_r: A counter that is incremented every n seconds (32 bits).
- $IPblacklist$: Table of IP addresses deemed to belong to known attackers and being actively blocked or rate limited on request of the server.
- $Flowblacklist$: A more dynamic table of <source IP, source Port> pairs that are being blocked temporarily with the associated values of C_r and C_r *- 1*. Each increment of C_r flushes the values associated with the old counter value.

The steps taken by the flow-cookie box are described below and shown in Figure 2.

- The cookie box intercepts all SYN packets destined to the webserver. If the SYN packet's source IP is in the $IPblacklist$, it is dropped (i.e., the IP blacklist is only looked up for SYN packets). Otherwise, the box responds with a SYN cookie [13] in which the source address is forged to be that of the webserver. The cookie is computed using a keyed message authentication code, such as UMAC-32 or

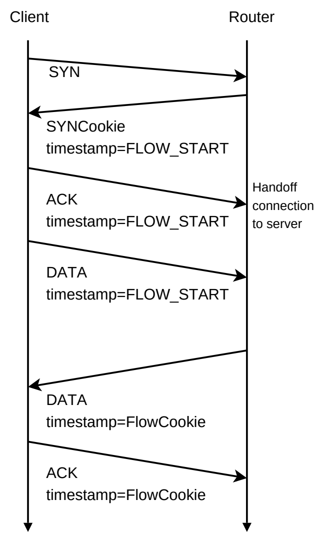

Figure 2: Exchange of packets between a client and flow-cookie box.

HMAC, over the concatenation of C_r and the connection 4-tuple:
cookie = MAC($S_r, C_r|src_{ip}|src_{port}|dst_{ip}|dst_{port}$)

- In addition to the SYN cookie, the cookie box includes a TCP timestamp option in the outgoing SYN|ACK with the timestamp set to the constant value *FLOW_START*, to indicate the possible start of a new TCP flow. This value is unique to SYN|ACK packets.

- For all packets to the web server that carry an ACK flag and have the timestamp *FLOW_START*, the cookie box checks that the ACK'ed sequence number is a valid SYN cookie. If yes, and the source IP address is not blacklisted, the connection is handed off to the web site using a TCP handoff scheme such as [7]. Note that with a starting slow start window size of 2 MSS, the cookie box might have to check 3 packets (ACK of SYN, and first two data packets) for a valid SYN cookie. All valid packets are forwarded on to the server.

- For each data packet sent by the server, the cookie box inserts a "flow-cookie" in the TCP timestamp field. The flow-cookie is calculated in the same manner as the SYN cookie.

- For all packets to the web server that carry an ACK flag but does not contain *FLOW_START* in the timestamp field, the cookie box checks that the returned timestamp field has a valid flow-cookie. By design, the maximum lifetime of a cookie is 2 times the increment period of C_r. If the packet does not have a valid flow-cookie, it is dropped. (Note that handling websites that use timestamps for RTT calculation requires a small modification to this step, discussed in Section 4.1.)

- If the webserver does not want to receive packets from a particular flow, it can do two things: (1) push filters to the cookie box's *flow blacklist*; in this case, the cookie box maintains the source IP and port in a revocation list with an associated time out and filters packets accordingly. (2) inform the cookie box to stop issuing fresh capabilities for the client. This can be done simply by closing the connection in which case the client will no longer receive valid cookies for the flow. The first approach can be employed to filter high-bandwidth malicious flows immediately. The second approach can be used in less critical situations or to shut off low priority clients when under overload.

- If a server determines that an attacker is behind a given IP address (or address block), it can request the cookie box to add the IP (address block) to the *IP blacklist*.

The cookie-box's per-packet forwarding checks from client to server is shown in pseudocode in Figure 3. And the forwarding checks from server to client is included in Figure 4. In the next section we discuss practical issues regarding the timestamp, RST packets and persistent connections.

4.1 Design Considerations

Patching timestamps. RFC-1323 does not specify how the timestamp value is to be encoded in the option field. To prevent disruption, the cookie box and the webserver must explicitly agree on a format. Also, care needs to be taken if the web site wishes to use timestamps to measure RTTs. The simplest approach to avoid undesirable interactions with flow cookies is for the third-party service to dissuade webservers from using timestamp values in RTT calculations. This requires a relatively insignificant modification to the TCP stack at the web server. Also, all stacks are able to do RTT calculations without the aid of timestamps.

An alternate method that does not require any modification of the webserver is to store a portion of the timestamp: Using this approach, the cookie box places the cookie in the most-significant 24 bits of the timestamp, reducing cookie size from 32 to 24 bits and preserving the low-order bits of the timestamp. When the client echoes the cookie, the cookie-box uses the last-seen timestamp (T_s) from the server to replace the upper 24 bits of the timestamp field thus patching-up the original timestamp. We note that as long as the RTT between the web site and the client is less than what can be represented by 8 bits in the

```
Sr: secret
Cr: current counter value
Ts: last server timestamp

1 if packet is SYN:
2   if scrIP in IPBlacklist:
3      drop packet and exit
4   syn_cookie = UMAC(Sr, Cr | 4-tuple)
5   send back SYN_ACK with ISN = syn_cookie and
    timestamp = FLOW_START
6 else:
7   cookie = UMAC(Sr, Cr | 4-tuple)
8   if timestamp == FLOW_START:
9      if ACK sequence != cookie:
10        drop packet and exit
11  else:
12     if timestmp[8:32] != cookie:
13       drop packet and exit
14     else:
15       if <srcIP and srcPort> in FlowBlacklist:
16           drop packet and exit
17  set timestmp[8:32] = Ts[8:32]
18  forward the packet
```

Figure 3: Per-packet logic at the cookie box for packets destined to the protected web-server.

```
1 Ts = timestamp value
2 cookie = HMAC_UMAC(Sr, Cr | 4-tuple)
3 timestamp[8:32] = cookie
```

Figure 4: Per-packet logic at the cookie box for packets sent from the protected web-server.

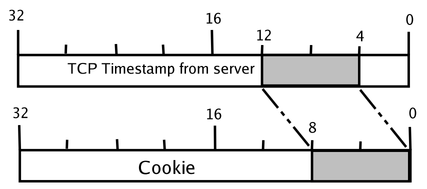

Figure 5: Preserving bits 4 through 12 from the original timestamp to support clients with high RTT to the server.

timestamp (i.e. 256ms if a millisecond resolution timer is being used), the original timestamp used by the web site is restored.

If the RTT is greater (or the server has a higher precision timer), the cookie-box can use higher-order bits of the timestamp (we use bits 4 through 12), effectively reducing the RTT timer granularity (see Figure 5). In Section 6 we show that reducing the fidelity of the timestamp by truncating the least significant 4 bits does not produce any measurable effect on throughput.

RST Packets. Flow-cookies relies on all packets to have a valid cookie in the TCP timestamp field. Unfortunately, the timestamp field is only valid if the packet's ACK bit is set. *All* TCP packets between a server and its client (in either direction) have their ACK bit set, *except RST packets*. RST packets are sent when clients continue to receive packets from the server, even after closing a connection. In our proposal, we rate limit TCP RST packets to the web server to a small percentage of the web site's link bandwidth L_b. Note that an attacker could fill the queue with illegitimate RSTs starving out the legitimate ones. However, the only adverse effect is that under attack conditions the web site sends out marginally more data on connections that could not complete a proper shutdown.[4]

Persistent Connections. Flow-cookies are issued for every packet from the server to its client. As long as a client can elicit a steady stream of packets from the server, it is assured of a valid capability. However, TCP connections such as persistent web sessions may remain idle for prolonged periods, in which case the client cookie may go stale (in our implementation, cookies time out after 30 seconds). To address this problem, the webserver can issue HTTP keep-alives (say every 15s) to updating the client flow-cookies. Keep-alives are small and will not consume significant bandwidth relative to the content at a busy Internet site.

Service Classes. In addition to enforcing the binary decision of accepting or denying a packet, flow-cookies can aid in differentiating flows into multiple classes. This is especially important under stress or attack conditions. The main idea is to encode the service class into the cookie (e.g as a 3bit value) which the cookie-box then uses to place incoming packets in the appropriate service queue. Doing so requires a change to the cookie computation so that instead of using a keyed hash, the cookie-box encrypts to generate the cookie and decrypts to retrieve the service class. We plan to continue to explore this area as future work.

[4]We note that outgoing RSTs are commonly blocked by firewalls today to protect against RST and inverse port scans.

5 Discussion

In this section, we discuss protection properties of the flow cookie mechanism. In addition, we discuss alternate avenues for attacking this mechanism, and possible workarounds.

5.1 Properties of Our Approach

As with most protection schemes, flow-cookies merely raise the bar of difficulty for an attacker to exhaust the bandwidth of a web site. Below, we outline the protection properties of flow-cookies.

Non-spoofability. Since the webserver only receives packets from established flows, flow cookies offers "non-spoofability": an attack IP has to be "real" for it to consume state at the cookie box or the webserver. An attacker with a spoofed source on the SYN cannot complete the three-way handshake. Similarly, an attacker who has established a valid connection cannot forge a valid cookie for an arbitrary source address. As a result, *no attacker (unless directly on-route) can trick the server or the cookie box into allocating state for IP addresses other than its own*.

Proportional state. In practice, the cookie box must store the IP blacklist, the flow blacklist, and the most recent timestamp per protected web site. Assuming that the web server being protected functions properly, the size of the two lists are proportional to the *number of attacking hosts/flows*, as opposed to aggregate number of flows.

Ease of deployment. By design, flow-cookies leverages a third-party service to offer bandwidth as a resource for protecting against DDoS. However, flow-cookies does not require the third-party to understand the semantics of the traffic to the servers, nor is the flow-cookie box susceptible to flow-state exhaustion due to legitimate flows. Further, the algorithmic simplicity and state-space requirements are such that flow-cookies can be implemented at relatively low cost.

Protection Bandwidth Guarantee. Flow cookies must make it difficult for the attacker to craft an attack packet that is outside an established flow, yet somehow allowed past the cookie box. Note that given an n bit cookie, for every 2^n packets generated by the attacker, one packet will get through, on average. Therefore, in order to maintain the protection bandwidth provided by the ingress link, the cookie size must be at least $\log_2(\frac{C_b}{L_b})$, where C_b is the bandwidth of link between the cookie box and its ISP (or the aggregate bandwidth in the case of multiple ISPs) and L_b is the bandwidth of the tunnel or link between the cookie box and the webserver. We believe our choice of 24 bits (with 8 bits reserved for time-stamp patching) is sufficient for almost all cases.

Fault Tolerance. Failure of the cookie-box must not cut traffic to or from the webserver. To ensure fault tolerance, the third-party provider could replicate the cookie-box *in situ*. Alternately, the provider could deploy cookie-boxes at multiple geographic locations. Clients of the protected web server could be randomly directed to one of the replicas using DNS (similar to existing third-party DNS mechanisms [2]) or anycast. The webserver site could similarly buy an additional Internet connection to use as a fail-over mechanism (special care must be taken to manage address allocation and route announcements).

5.2 Web Server Actions

In our approach, the web servers are fully responsible for discovering misbehaving clients and pushing filtering requests the cookie-box. Because the web-server must maintain per-connection state it is in the best position to determine if the TCP congestion protocol is not being followed or if a valid cookie is being used for flooding. In addition, the web-servers can use knowledge of higher-layer semantics to determine if a client is attempting to exhaust downloading bandwidth or CPU cycles. Filtering requests are pushed over the established private connection between the server and the cookie box.

5.3 Other Attack Venues

Download Bandwidth Exhaustion. Flow-cookies does not protect the outgoing link of the server from bandwidth exhaustion attacks. This can happen in “e-protests” where many clients plan and simultaneously request a large data item. We believe defenses for such attacks are most appropriately employed on the server where application specific knowledge can be used, for example, to compress specific objects.

State Exhaustion Attack on Filter Lists. An attacker may attempt to exhaust the memory used to store flow blacklists at the cookie-box by generating a large number of malicious flows from one or more IP addresses. However, the flow blacklist entries are established by the protected web server, which is assumed trustworthy and entries are only placed on legitimate flows and IP addresses. Our naive implementation discussed in Section 6 is able to handle millions of entries at gigabit speeds—orders of magnitude higher than the largest botnets known today.

Furthermore, the third-party provider could place a bound on the amount of filtering state that a protected web-server can consume. Beyond this, the webserver will either have to pay for extra room, or push aggregated filters instead of individual IPs; doing the latter has the obvious downside of preventing access to legitimate clients that share a common prefix with attackers.

Coordinated Bot Attacks. Perhaps the most effective attack against flow-cookies is for a large bot-network to launch a sustained attack by requesting and sharing individual cookies. For example, a single bot first obtains a flow-cookie with a 10s lifetime. It immediately communicates the cookie to the other bots of a large network. Each host in the bot network then spoofs its source address to that of the compromised client, and sends the webserver a burst of data, using the flow-cookie to get past the cookie-box. The webserver can stop the attack by immediately requesting a filter on the compromised IP. However, the bot-network could cycle through all of the IPs of nodes in the bot-net and continue to inflict damage. But we note that if it takes the webserver 50ms to detect and blacklist the source of a flood, then in 5s a 100 node bot network would have been exhausted; in 500 seconds, a 1000 node network etc. Larger botnets of tens or hundreds of thousands of hosts can anyway easily render the cookie-box imeffective by flooding the link to its ISP.

Attacks on DNS and Other non-TCP Traffic. We assume that all non-TCP traffic is either explicitly filtered, or placed in a different queue from TCP traffic, and rate limited in aggregate. However, there may be some non-TCP traffic that is critical to operation of the server, such as DNS traffic; this could be another avenue for attack. Though important, flow-cookies does not directly address these forms of attack. For the purposes of this paper we assume that separate protection mechanisms are used, such as DNS offloading to third-party services [2].

6 Implementation and Testing

We implemented our flow-cookie approach within an existing software router. While flow-cookies was designed to be simple and amenable to hardware implementation, this remains part of our future work.

In this section, we present results from testing our implementation on clients using popular web-browsers on commodity, unmodified operating systems against public servers. We also include results from micro-benchmark tests designed to explore the performance characteristics of flow-cookies and any deleterious affects caused by the loss in timer resolution of the TCP timestamp. In addition to live tests on the Internet, we test our implementation in software using simulated traffic loads to show that even on a standard GPU, flow-cookies can handle gigabit line speeds. Finally, we explore the effectiveness of our approach under simulated attack conditions.

6.1 Implementation Details

Our implementation of flow-cookies was done in a simple software-router using a user-space network development environment [14]. The development environment provides hooks into the kernel for sending and receiving packets while allowing low-level network functionality to be implemented outside of the kernel. In addition to flow-cookies, our router supports ARP, a subset of ICMP, IP-prefix forwarding and IP TTL handling.

The implementation was done in C using HMAC-SHA1() from OpenSSL to compute the cookie. For the HMAC we use a 64-bit secret and increment the router's timer (C_r) every 30 seconds. Incorporating flow-cookies into the router required an additional 320 lines of code. Also, as discussed in Section 4.1, to repair the server timestamp value in place of the returned cookie, the implementation saves bits [4:12] of the original timestamp in the cookie (we discuss the impact of this in Section 6.3) We did not implement SYN cookies and the TCP handoff because it would prevent us from testing against existing, public sites over which we have no administrative control.

The IP Blacklist is implemented as a Patricia-Trie whose minimum node size is a 4-bit nibble. We chose to use a trie because it is space and time efficient for IP lookup [20] and amenable to hardware [21]. The source IP of all packets are checked against the trie and discarded if a match is returned.

6.2 Compatibility Testing with Existing Servers

In our first test, we validated that flow-cookies is indeed compatible with existing operating system and clients by testing against 10 public web-servers. To do this, we added a module to our implementation that would "bounce" packets sent from the client after they pass through the router to the server by rewriting the source and destinations addresses of the IP header. Packets returned from the web-servers where similarly bounced back by the module to the client. All compatibility tests were performed over a relatively low-bandwidth network (100Mbps).

The target websites included popular education, entertainment, news and hobby sites. For each of the web-servers we tested various client OSes: Linux, MacOSX, NetBSD and WindowsXP (in the last case, with timestamps turned on). Our test included recursive queries of static content as well as GETs and POSTs when supported by the site. In all cases, we found that flow-cookies was *totally compatible* with various client OSes and the servers we tested against.

In addition, we measured the the average time it took to *wget* files on the main page of each web site

site	RTT to site	router	bytes recd	files	avg. time (s)	var. (s)
1	77ms	norm	2,860k	119	37.78	0.515
		flow	2,860k	119	44.32	1.05
2	49ms	norm	704k	58	19.06	1.61
		flow	704k	58	18.42	.50
3	2.8ms	norm	155,727k	588	75.28	2.53
		flow	155,727k	588	79.12	1.19
4	97ms	norm	2,303k	255	81.74	6.83
		flow	2,303k	255	82.62	5.50
5	198ms	norm	4,576k	230	191.93	12.51
		flow	4,576k	230	179.40	8.10

Table 1: Comparison of router with (flow) and without (norm) flow cookies downloading multiple files from 5 websites.

using both the flow-cookies enabled router and an unmodified version. Some of the results of our tests are showing in Table 1. We note that there is no observable effect on either the download times or in the variance thereof.

6.3 Effects of RTT fidelity on total throughput

Our implementation lowered the resolution of the TCP timestamp from 1ms to 16ms increments by truncating the lower 4 bits. As shown in [28], RTTs over the Internet path can vary by several milliseconds. Hence, the loss of granularity of RTT for $<$ 16ms variations should not lead to drastically different client or server behavior. This is partly supported by the results of our throughput tests above.

However, because the RTT is used to calibrate the TCP retransmission timeout value, the negative affects of poor RTT calculation are most likely to arise during conditions of high loss. We explore this case by re-running multiple HTTP sessions on Internet paths with different underlying RTTs using flow-cookies. We forced a loss rate of 9%; the losses are distributed uniformly at random.

The results for two Internet paths are show in Figure 6. We see that a reduction in granularity by up to 16ms (4 bits) does not have a significant impact on the throughput during loss conditions. However, a further loss in granularity significantly reduces the effective throughput.

We reiterate that "patching up" the timestamp is done merely as a convenience so that the server does not require modification to the kernel. For servers that do rely on high-precision RTTs, we would recommend that the timestamp not be used for RTT calculation under attack conditions when flow-cookies is being used

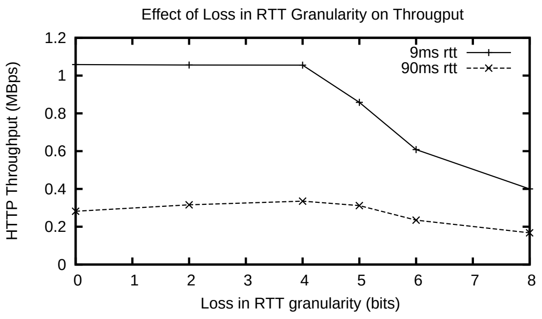

Figure 6: The effect of loss in RTT fidelity on total throughput over a lossy link. We test from two sources, one with a 9ms RTT from the server and another with 90ms. The low throughput is due to high packet loss.

	network throughput	software
unmodified	84.84/s	8.3Gbs
flow-cookies	84.16/s	2.38Gbs
large blacklist	82.56/s	2.13Gbs

Table 2: Comparison of network and router throughput with (flow-cookies) and without (unmodified) and with 1 million randomly generated IP entries in the IP blacklist (large blacklist).

for defense.

6.4 Performance Microbenchmarks

Table 2 shows network throughput of the router before and after the flow-cookie modifications, on a 1.2GHz Pentium III running Redhat Linux 9.0. These tests are limited by the speed of our network testbed (100Mbps). The numbers are obtained for multiple downloads of a 512MB file through the router. We note that our flow-cookie enabled router was able to perform at the same speed as the unmodified router. We also profiled the CPU and found that the per-packet processing overhead was dominated by the hash computation.

In order to test the throughput performance of the flow-cookie router under stress conditions, we modified the router to read traces directly from a file and run them through the router forwarding path without injecting them back on the network. Under these idealized conditions, our implementation of flow-cookies was able to process packets at a speed of *2.38Gbps*. Note that the forwarding path used was identical to that in the full router, that is, it performed all the header parsing, decision logic, flow cookie computation and TCP checksum update on modification of the timestamp.

We profiled the execution of the flow-cookies during the software test-case and, again, a significant percentage (91%) of the time was spent in flow cookie computation. In a hardware implementation, the

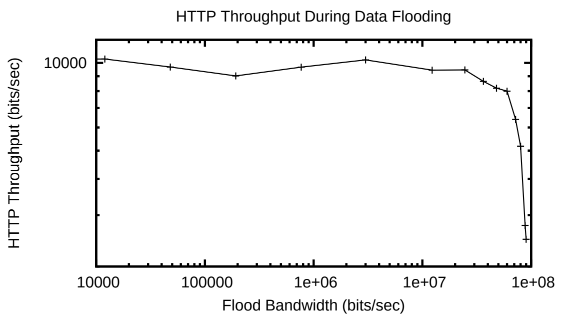

Figure 7: Affect of SYN flooding attack on legitimate, low-bandwidth HTTP flows.

hash function will be offloaded to an application specific chip as is done in crypto-accelerators today. Flow-cookies uses the same cookie generation computation as SYN cookies which today support multi-gigabit line speeds [4]. We also note that switches today, such as Cisco's catalyst 6K series, can already perform encryption at 10Gbps.

6.5 Blacklist Lookup Performance

Next, we test the impact of per-packet lookup in the IP blacklist on the router's throughput performance. We note that our blacklist implementation is able to handle over a million insertions per second and nearly two million lookups per second when filled with a million entries (i.e., attacker IP addresses) generated uniformly at random. Even given our naive software implementation which checks the blacklist serially before the cookie computation, the lookup has little affect in the network and software tests (see Table 2, row 3). Note that in a hardware implementation, the lookup would in fact be done in parallel with the cookie computation.

6.6 Flood Protection

We explored the protection capacity of our flow cookie implementation for a low-bandwidth back-end server during a high-bandwidth SYN flooding attack. In these experiments, we ran multiple legitimate HTTP downloads to the back-end server. These downloads were rate-limited so that aggregate bandwidth did not exceed 10Mbps. Since we did not implement SYN cookies, in order to generate flood conditions we ran multiple instances of hping2, generating valid TCP packets with random timestamps at rates ranging from 12Kbps to 88Mbps.

The average throughput of HTTP flows during flooding is shown in Figure 7. We see that that until the flood nears 65Mbps, the aggregate throughput of legitimate flows is virtually unaffected. At this stage, the router sees 75Mbps of traffic in aggregate, which is close to its capacity (85Mbps, Table 2). At saturation, throughput drops drastically due to heavy packet loss at the router. These results show that our implementation can indeed offer back-end servers the protection bandwidth of the higher-bandwidth link between the cookie box and its ISP for SYN flood attacks.

7 Conclusions and Future Work

Bandwidth exhaustion DDoS attacks are a very-real and serious threat on the Internet. "Flow-cookies" is an evolutionary approach to preventing bandwidth exhaustion attacks that does not require modification of client end-hosts. It provides a web site with the effective bandwidth of a high-speed link for processing and discarding attack packets, without requiring the network to store per-flow state. Flow-cookies does not require modification to clients, is resistant to source spoofing, and is simple to deploy. The state required in the network to filter attacks is proportional to the number of real attackers and is small enough that fast packet lookup implementations in SRAM and available TCAMs can handle multiples of the largest bot-nets known to exist today.

A main part of our future work is to implement flow-cookies in hardware using a network programmable FPGA platform such as NetFPGA-2 [15]. We plan to deploy a live implementation of flow-cookies in hardware at 1Gbps speeds or higher in order to further study deployment, compatibility and performance issues.

References

[1] Akamai home page. http://www.akamai.com.

[2] Akami first point. http://www.akamai.com/en/html/about/press/press129.html.

[3] Internet attack targets doubleclick. http://www.washingtonpost.com/wp-dyn/articles/A18735-2004Jul27.html.

[4] Netscaler syn flood protection. http://www.netscaler.com/docs/library/NetScalerSYNDefense.pdf.

[5] Prolexic home page. http://prolexic,com.

[6] Rings of steel combat net attacks. http://news.bbc.co.uk/1/hi/technology/4169223.stm.

[7] Tcpha home page. http://dragon.linux-vs.org/ dragonfly/.

[8] Denial of service hackers take on new targets. http://archives.cnn.com/2000/TECH/computing/02/09/denial.of.service.03/-index.html, 2000.

[9] Us credit card firm fights ddos attack. http://www.theregister.com/2004/09/23/authorize_ddos_attack/, 2004.

[10] D. Andersen. Mayday: Distributed filtering for internet services. In *USITS, Seattle, WA, 2003.*, 2003.

[11] K. Argyraki and D. Cheriton. Network capabilities: The good, the bad and the ugly. In *ACM HotNets IV*, 2005.

[12] K. Argyraki and D. R. Cheriton. Active internet traffic filtering: Real-time response to denial-of-service attacks. In *USENIX Annual Technical Conference*, 2005.

[13] D. Bernstein. Syn cookies. http://cr.yp.to/syncookies.html, 1996.

[14] M. Casado and N. McKeown. he virtual network system. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, 2005.

[15] M. Casado, G. Watson, and N. McKown. Reconfigurable networking harfware: A classroom tool. In *Hot Interconnects*, 2005.

[16] G. V. David Moore and S. Savage. Inferring internet denial of service activity. In *Proceedings of the 2001 USENIX Security Symposium*, 2001.

[17] M. J. Freedman, E. Freudenthal, and D. Mazires. Democratizing content publication with coral. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI '04)*, San Francisco, California, March 2004.

[18] J. Ioannidis and S. M. Bellovin. Implementing pushback: Router-based defense against DDoS. In *Proceedings of Network and Distributed System Security Symposium*, 2002.

[19] A. Keromytis, V. Misra, and D. Rubenstein. Sos: Secure overlay services. In *Proceedings ofACM SIGCOMM'02*, 2002.

[20] S. Nilsson and G. Karlsson. Fast address look-up for internet routers, 1998.

[21] H. Song and J. W. Lockwood. Efficient packet classification for network intrusion detection using fpga. In *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pages 238–245, New York, NY, USA, 2005. ACM Press.

[22] M. J. Srikanth Kandula, Dina Katabi and A. Burger. Botz-4-sale: Surviving ddos attacks that mimic flash crowds. In *Usenix NSDI*, 2005.

[23] A. Stavrou, D. L. Cook, W. G. Morein, A. D. Keromytis, V. Misra, and D. Rubenstein. WebSOS: an overlay-based system for protecting web servers from denial of service attacks. 48(5):781–807, Aug. 2005.

[24] A. B. Tadayoshi Kohno and K. Claffy. Remote physical device fingerprinting. In *IEEE Symposium on Security and Privacy*, 2005.

[25] M. Welsh and D. Culler. Adaptive overload control for busy internet servers. In *USITS*, 2003.

[26] D. W. X. Yang and T. Anderson. A dos-limiting network architecture. In *Proc. ACM SIGCOMM*, 2005.

[27] A. Yaar, A. Perrig, and D. Song. Siff: A stateless internet flow filter to mitigate ddos flooding attacks. In *In Proceedings of the IEEE Security and Privacy Symposium*, 2004.

[28] Z.-L. Z. Zhenhai Duan, Kuai Xu. Understanding delay variations on the internet paths. http://www.cs.fsu.edu/ duan/publications/pathdelay.pdf.