WebSOS: Protecting Web Servers From DDoS Attacks

Debra L. Cook*, William G. Morein*, Angelos D. Keromytis*, Vishal Misra* and Daniel Rubenstein†
* Department of Computer Science † Department of Electrical Engineering
Columbia University
New York City, NY, USA
{dcook,wgm2001,angelos,misra,danr}@cs.columbia.edu

***Abstract*— We present the WebSOS architecture, a mechanism for countering denial of service (DoS) attacks against web servers. WebSOS uses a combination of overlay networking, content-based routing, and aggressive packet filtering to guarantee access to a service that is targeted by a DoS attack. Our approach requires *no modifications* to servers or browsers, and makes use of the web proxy feature and TLS client authentication supported by modern browsers.**

We use a WebSOS prototype to conduct a preliminary performance evaluation both on the local area network and over the Internet using PlanetLab, a testbed for experimentation with network overlays. We determine the end-to-end latency imposed by the architecture to increase by a factor of 5 on average. We conclude that this overhead is reasonable in the context of a determined DoS attack.

Keywords: *Denial of Service, web proxies, overlay networks, packet filtering, consistent hashing, distributed hash tables.*

I. Introduction

The extremely widely-used World Wide Web environment provides a rich set of targets for motivated attackers. This has been demonstrated by the large number of vulnerabilities and exploits against web servers, browsers, and applications utilizing these (*e.g.,* CGI scripts). Traditional security considerations revolved around protecting the network connection's confidentiality and integrity, protecting the server from break-in, and protecting the client's private information from unintended disclosure. To that end, several protocols and mechanisms have been developed that address these issues individually [1], [2], [3], [4], [5]. One area that has been neglected thus far has been that of service availability in the presence of denial of service (DoS) attacks, and their distributed variants (DDoS). Such attacks can take many forms, depending on the resource the attacker is trying to exhaust. Of particular interest are *link congestion* attacks, whereby attackers identify "pinch" points in the communications substrate and flood them with large volumes of traffic. An example of an obvious attack point is the location (IP address) of the destination that is to be secured, or the routers in its immediate network vicinity; sending enough attack traffic will cause the links closest to the destination to be congested and drop all other traffic.

Previous approaches that address the general network DoS problem [6], [7], [8] are *reactive*: they monitor traffic at a target location, waiting for an attack to occur. After the attack is identified, typically via analysis of traffic patterns and packet headers, filters may be established in an attempt to block the offenders. The main two problems with this approach are the accuracy with which legitimate traffic can be distinguished from the DoS traffic, and the robustness of the mechanism for establishing filters deep enough in the network (away from the target) so that the effects of the attack are minimized.

We build on the *Secure Overlay Services (SOS)* architecture, originally introduced in [9]. Our intent is to allow legitimate users to access web servers under a congestion-based DDoS attack. We assume that there is a pre-determined subset of clients scattered throughout the wide-area network who require (and should have) access to these servers. These users prove their right to contact the web server through cryptographic means — *i.e.,* possession of a secret key. This works well in the web environment, where users are familiar with SSL [1] and, more importantly, practically all browsers provide the necessary functionality. We should stress that WebSOS does not solve the general DoS problem (*e.g.,* the "flash crowd" phenomenon). We are interested in classes of communication where both participants are known to each other *a priori*.

In WebSOS, the portion of the network immediately surrounding the web servers to be protected aggressively filters and blocks all incoming packets from hosts that are not approved, as shown in Figure 1. The small set of nodes that are approved at any particular time is kept secret so that attackers cannot try to impersonate them to pass through the filter. These nodes are picked from among those within a distributed set of nodes throughout the wide area network, that form a *secure overlay* dedicated to DoS protection services. Any transmissions that wish to traverse the overlay must first be validated at entry points of the overlay. Once inside the overlay, the traffic is tunneled securely for several hops along the overlay to the approved (and secret from attackers) locations, which can then forward the validated traffic through the filtering routers to the target. Thus, the two main principles behind our design are the elimination of communication pinch-points, which constitute attractive DoS targets, via a combination of filtering and overlay routing to obscure the identities of the sites whose traffic is permitted to pass through the filter, and the ability to recover from random or induced failures within the forwarding infrastructure or within the secure overlay nodes.

We use web proxies enhanced with Chord-routing [10]

functionality as the overlay nodes. A proxy local to the web browser performs the authentication between the legitimate user and the overlay network, using TLS. Traffic inside the overlay is also protected via TLS. The implementation is fairly compact, and was used to evaluate the performance of the WebSOS overlay both in a local area scenario and using the PlanetLab testbed [11]. The results show that the end-to-end latency increases, on average, by a factor 5. We believe this is an acceptable alternative to providing no service. WebSOS can be used on an as-needed basis, without affecting performance when no DoS attack is taking place.

A. Paper Organization

The remainder of this paper is organized as follows. Section II gives an overview of Secure Overlay Services (SOS) discusses the specifics of the WebSOS architecture. Section III presents details of our prototype implementation, while Section IV contains a preliminary performance evaluation. Section V discusses related work in DoS detection, prevention, and mitigation. Section VI concludes the paper and discusses directions for future research.

II. The WebSOS Architecture

The goal of the WebSOS infrastructure is to distinguish between authorized and unauthorized traffic. The former is allowed to reach the destination, while the latter (or, more generally, unverified traffic) is dropped or rate-limited. Thus, at a very basic level, we need the functionality of a firewall "deep" enough in the network that the access link to the target is not congested. This imaginary firewall performs access control by using traditional protocols such as IPsec. Without WebSOS, knowledge of the target's IP address is sufficient to a moderately-provisioned attacker to saturate the target site.

Unfortunately, traditional firewalls themselves are susceptible to DoS attacks. One way to address this problem is to replicate the firewall functionality, in a manner similar to that described in [12]. To avoid the effects of a DoS attack against the firewall connectivity, we need to distribute these instances of the firewall across the network. In effect, we are "farming out" the expensive processing (such as cryptographic protocol handling) to a large number of nodes. However, firewalls depend on topological restrictions in the network to enforce access control policy. Since our distributed firewall has performed the access control step, it would seem obvious that all we need around the target is a router that is configured to only let through traffic forwarded to it by one of the firewalls.

However, a security system cannot depend upon the identity of these firewalls to remain secret. Thus, an attacker can launch a DoS attack with spoofed traffic purporting to originate from one of these firewalls. Notice that, given a sufficiently large group of such firewalls, we can select a very small number of these as the designated authorized forwarding stations: only traffic forwarded from these will be allowed through the filtering router, and we change this set periodically. We call these nodes *secret servlets.* All other nodes must forward traffic for the target to one of these nodes. Figure 1 gives a high-level overview of the WebSOS architecture.

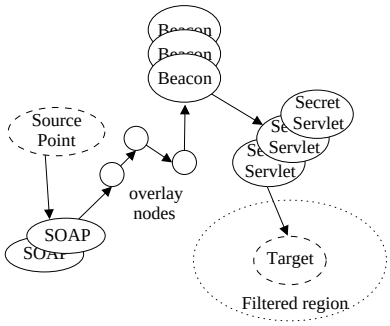

Fig. 1. **Basic SOS architecture.**

To route traffic internally, WebSOS uses Chord [10], which can be viewed as a routing service that can be implemented as a network overlay. Consistent hashing [13] is used to map an arbitrary identifier to a unique destination node that is an active member of the overlay. Each overlay node maintains a list that contains $O(\log N)$ identities of other active nodes in the overlay, where N is the number of overlay nodes. Given the destination identifier, each node knows how to choose a member in its list such that, from an arbitrarily chosen starting node, the destination node to which the identifier hashes is reached in $O(\log N)$ overlay hops. The Chord service is robust to changes in overlay membership: each node's list is adjusted to account for nodes leaving and joining the overlay such that the above stated properties continue to hold. Note that WebSOS can use any routing algorithm; we chose Chord because of our familiarity with it and its self-healing properties.

In WebSOS, the identifier to which the hash function is applied to is the IP address of the target (web server). Thus, Chord can be used to direct a packet from any node in the overlay to the node that the identifier is mapped to (or the node whose identifier is "closest" to that value), by applying the hash function to the target's IP address. This node to which Chord delivers the packet is not the target, nor is it necessarily the secret servlet. It is simply a unique node that will be eventually be reached using Chord, regardless of the starting point in the overlay. We refer to this node as the *beacon*, since it is to this node that packets destined for the target are first guided. When a packet is approved by a SOAP for forwarding over the overlay, the hash on the IP address of the target is used as the key. Chord therefore provides a robust and reliable while relatively unpredictable means of routing packets from an access point to one of several beacons.

The secret servlet can determine the beacon's identity by hashing on the target identifier (which the secret servlet knows), and then use Chord to route traffic to it. Hence, the secret servlet can inform the beacon of the secret servlet's identity. The servlets send these advertisements periodically. Should the servlet for a target change, the beacon will find out as soon as the new servlet sends an advertisement. If

the old beacon for a target drops out of the overlay, Chord will route the advertisements to a node closest to the hash of the target's identifier. Such a node will know that it is the new beacon because Chord will not be able to further forward the advertisement. By providing only the beacon with the identity of the secret servlet, the packet can be delivered from any overlay entry point (we call these Secure Overlay Access Points, or SOAPs) to the target by traveling across the overlay to the beacon, then from the beacon to the secret servlet, and finally from the secret servlet (through the filter) to the target. This allows the overlay to scale for arbitrarily large numbers of overlay nodes and target sites.

A. WebSOS Overlay Nodes

We use special-purpose web proxies as the overlay nodes. These proxies implement the following functionality:

1) Receive SSL connections from other proxies; both sides are mutually authenticated using public key certificates issued by the administrator of the WebSOS overlay.
2) Proxy HTTP (or HTTPS) requests received over these SSL connections to other WebSOS nodes, again over SSL.
3) Accept SSL connections from browsers, verify the client certificate received over the SSL exchange, and proxy the HTTP or HTTPS request(s) to other overlay nodes, per item *(2)*.
4) Implement the Chord routing algorithm, as discussed above. The routing table thus constructed is used to determine the next proxy to forward an HTTP/HTTPS request to, as per above.
5) Implement beacon functionality. This means that every node must maintain a table associating web server IP addresses to secret servlet IP addresses. If a proxy request for a web server in this table is received, the connection is proxied to the associated secret servlet.
6) Implement secret servlet functionality. If a request is received for a web server for which the node is a secret servlet, the connection is forwarded directly to the web server. Furthermore, allow authorized web servers to notify a WebSOS node that it is a secret servlet for that server. The node must verify the certificate presented by the server in the SSL exchange, and then notify the beacon for that target that it is the new servlet.

Notice that if the browser is doing end-to-end SSL (to the web server), the encrypted link will be super-encrypted with SSL on the link between the browser and the SOAP, and between each overlay node. We use SSL in the intermediate links to prevent an attacker from inserting traffic in the overlay through IP spoofing. We also use SSL as an admission-control mechanism to the overlay; SOAPs perform the admission control function. No special functionality is required by the proxies to perform these tasks; the user browser simply has to be supplied with the appropriate public key certificate(s) from the WebSOS administrator.

Traffic in the reverse direction (from web server to client) could also traverse the overlay, by reversing the roles of user and target. In that case, the path taken by requests and responses would be different. Alternatively, traffic from the target to the user could be sent directly (without using the overlay); this is usually not a problem, since most communication channels are full-duplex and, in the event of a DDoS attack, only the downstream portion (to the target) is congested. An additional benefit of this asymmetric approach is reduced latency, especially considering that most client/server traffic (especially in the web environment) is highly asymmetric (*i.e.,* clients receive a lot more information than they transmit). This was possible because routing decisions in SOS are made on a per-packet basis.

In WebSOS, routing decisions are made on a per-connection basis. Any subsequent requests over the same connection (when using HTTP 1.1) and any responses from the web server can take the reverse path through the overlay. While this makes the implementation simpler, it also introduces increased latency, as the bulk of the traffic will also traverse the overlay. We shall see in the next section how to avoid this problem.

B. Sequence of Operations

The sequence of operations in the WebSOS architecture consists of the following steps.

- A site (target) installs a filter on a router in its immediate vicinity and then selects a number of WebSOS nodes to act as servlets. Routers at the perimeter of the site are instructed to only allow traffic from these servlets to reach the internal of the site's network. These routers are powerful enough to do filtering (using only a small number of rules) on incoming traffic without adversely impacting their performance. In order to make guessing the identity of a secret servlet for a particular target harder for the attacker, the filtering mechanism should use packet fields with potentially high entropy. For example, only GRE [14] packets form a particular source (the secret servlet) containing a specific 32-bit value in the GRE Key field [15]. An attacker trying to slip attack traffic through the filter must guess both the current servlet's IP address and the correct 32-bit key.
- When an SOS node is informed that it will act as a secret servlet for a site (and after verifying the authenticity of the request, by verifying the certificate received during the SSL exchange), it will compute the key k for each of a number of well-known consistent hash functions, based on the target site's network address. Each of these keys will identify a number of overlay nodes that will act as beacons for that web server. Any traffic to the target received from the overlay will be directly forwarded to these nodes, and from there to the secret servlets.
- Having identified the beacons, the servlets or the target will contact them, notifying the beacons of the servlets' existence and association with a particular target. Beacons, after verifying the validity of the request, will store the necessary information to forward traffic for that target to the appropriate servlet.

- A source that wants to communicate with the target contacts a SOAP. After authenticating and authorizing the request, the SOAP securely proxies all traffic from the source to the target via one of the beacons. The SOAP (and all subsequent hops on the overlay) can proxy the HTTP request to an appropriate beacon in a distributed fashion using Chord, by applying the appropriate hash function(s) to the target's IP address to identify the next hop on the overlay.
- Finally, the beacon proxies the request to a secret servlet that then proxies the traffic (through the filtering router) to the target.

This scheme is robust against DoS attacks because if an access point is attacked, the confirmed source point can simply choose an alternate access point to enter the overlay. Any overlay node can provide all different required functionalities (SOAP, Chord routing, beacon, secret servlet). If a node within the overlay is attacked, the node simply exits the overlay and the Chord service self-heals, providing new paths over the re-formed overlay to (potentially new sets of) beacons. Furthermore, no node is more important or sensitive than others — even beacons can be attacked and are allowed to fail. Finally, if a secret servlet's identity is discovered and the servlet is targeted as an attack point, or attacks arrive at the target with the source IP address of some secret servlet, the target can choose an alternate set of secret servlets.

III. Implementation

We have an initial implementation of WebSOS, consisting of three main modules. The components are a communications module, a SOS routing module, and an overlay routing module running on each node in the SOS overlay.

The communications module is responsible for forwarding HTTP requests and responses among the nodes in the SOS overlay. When a new proxy request (in the form of a new TCP connection) is received, the communications module calls the SOS routing module with the target's destination address to obtain the address of the next hop in the overlay. It then opens a new TCP connection to that node and relays the received HTTP request. Any traffic received in the opposite direction (*i.e.,* the HTTP responses and web content) are relayed back to the source. Authentication of the requesting node by the SOAP and internal nodes is accomplished through SSL. Authorized users and SOS overlay nodes are issued X.509[16] certificates signed by the WebSOS certificate authority.

Thus, when a request is issued by the browser, it is tunnelled through a series of SSL-encrypted links to the target, allowing the entire transmission between the requester and target to be encrypted. These SSL connections between WebSOS nodes are dynamically established, as new requests are routed. One problem we ran into while developing the WebSOS prototype is that web browsers do not provide support for the actual proxy request to be encrypted. To solve this problem, we wrote a port forwarder that runs on the user's system, accepts plaintext proxy requests locally, and forwards them using SSL to the access point node. In the current implementation, this forwarder is a stand-alone program. Thus, to use WebSOS, an authorized user simply has to access any SOAP, download the proxy code, and set the browser's proxy settings to the localhost. The downloaded proxy itself is not considered part of the WebSOS overlay and is not trusted to perform any access control decisions; it is simply a "helper" application.

TABLE I

Latency, measured in seconds, when contacting various SSL-enabled web servers directly and with different numbers of (intermediate) overlay nodes in the same LAN.

Server/Nodes	Direct	1	4	7	10
Yahoo!	1.39	2.06	2.37	2.79	3.33
Verisign	3.43	4.22	5.95	6.41	9.01
CU BB	0.64	0.86	1.06	1.16	1.21
CU BB (2nd)	0.14	0.17	0.19	0.20	0.25

The *SOS routing module* receives requests from the communications module and responds with the IP address of the next node in the SOS overlay to which the request should be forwarded. The module first checks whether the current node serves a specific purpose (*i.e.*, whether is it a beacon or secret servlet for that target). If the node serves no such purpose, the module calls the overlay routing module to determine the next hop in the SOS overlay and passes the reply to the communications module.

The overlay routing module is a general routing algorithm for overlay networks. A Chord implementation was used in our initial tests, but can be replaced with any other routing algorithm, *e.g.,* CAN [17]. It receives queries containing a destination IP address (the web server's) and responds with the IP address of the next node in the overlay to which the request should be forwarded. For maintenance of its own routing table, the Chord implementation also communicates with other overlay nodes to determine their status.

IV. Experimental Evaluation

In order to quantify the overhead imposed by the WebSOS architecture, we created a simple topology running on the local network (100 Mbit fully-switched Ethernet). For our local-area network overlay, we used 10 commodity PCs running Linux Redhat 7.3. We measured the time-to-completion of https requests. That is, we measured the elapsed time starting when the browser initiates the TCP connection to the destination or the first proxy, to the time all data from the remote web server have been received. We ran this test by contacting 3 different SSL-enabled sites: *login.yahoo.com*, *www.verisign.com*, and our Columbia University's course bulletin-board web service. For each of these sites we measured the time-to-completion for a different number of overlay nodes between the browser and the target (remote web server).

The browser was located in a separate network. The reason for this configuration was to introduce some latency in the first-hop connection (from the browser to the SOAP), thus simulating (albeit using a real network) an environment where the browsers have slower access links to the SOAPs, relative to the links connecting the overlay nodes themselves (which

TABLE II

Latency, measured in seconds, when contacting various SSL-enabled web servers directly and with different numbers of (intermediate) overlay nodes using PlanetLab.

Server/Nodes	Direct	1	4	7	10
Yahoo!	1.39	3.15	5.53	10.65	14.36
Verisign	3.43	5.12	7.95	14.95	22.82
CU BB	0.64	1.01	1.45	3.14	5.07
CU BB (2nd)	0.14	0.23	0.28	0.57	0.72

may be co-located with core routers). By locating all the overlay nodes in the same location, we effectively measure the aggregate overhead of the WebSOS nodes in the optimal performance case.

Table I shows the results for the case of 0 (browser contacts remote server directly), 1, 4, 7, and 10 traffic-handling overlay nodes. The times reported are in seconds, and are averaged over several HTTPS GET requests of the same page, which was not locally cached. For each GET request, a new TCP connection was initiated by the browser. The row labeled "CU BB (2nd)" shows the time-to-completion of an HTTPS GET request that uses an already-established connection through the overlay to the web server, using the HTTP 1.1 protocol.

As the table shows, WebSOS increases the end-to-end latency between the browser and the server by a factor of 2 to 3. This increase can be primarily attributed to the network-stack and proxy processing overhead at each hop.

Furthermore, there is an SSL-processing overhead for the inter-overlay communications. A minor additional cryptographic overhead, relative to the direct access case, is the certificate validation that the SOAPs have to perform, to determine the client's authority to use the overlay, and the SSL connection between the proxy running on the user's machine and the SOAP. As shown in [18], such overheads are typically dominated by the end-to-end communication overheads. Use of cryptographic accelerators can further improve performance in that area. One further optimization is to maintain persistent SSL connections between the overlay nodes. However, this will make the task of the communication module harder, as it will have to parse HTTP requests and responses arriving over the same connection in order to make routing decisions.

Table II shows the same experiment using PlanetLab [11], a wide-area overlay network testbed. The PlanetLab nodes are distributed in academic institutions across the country, and are connected over the Internet. We deployed our WebSOS proxies PlanetLab and ran the exact same tests. Naturally, the direct-contact case remains the same. We see that the time-to-completion in this scenario increases by a factor of 2 to 10, depending on the number of nodes in the overlay. In each case, the increase in latency over the local-Ethernet configuration can be directly attributed to the delay in the links between the WebSOS nodes. While the PlanetLab configuration allowed us to conduct a much more realistic performance evaluation, it also represents a worst-case deployment scenario for WebSOS: typically, we would expect WebSOS to be offered as a service by an ISP, with the (majority of) WebSOS nodes located near the core of the network. Using PlanetLab, the nodes are distributed in (admittedly well-connected) end-sites. We would expect that a more commercial-oriented deployment of WebSOS would result in a corresponding decrease in the inter-overlay delay. On the other hand, it is easier to envision end-site deployment of WebSOS, as it does not require ISP participation.

Note the latency increases with the $\log$ of the number of nodes in the overlay, since that is how many nodes, in the worst case, a connection will be relayed through before it reaches the beacon. However, the difficulty of shutting down the network for the attacker increases linearly with the number of nodes. This is directly analogous to modern cryptography: extending the length of a cryptographic key increases the work factor for legitimate users linearly, but makes a brute force attack exponentially more difficult.

Finally, while the additional overhead imposed by WebSOS can be significant, we have to consider the alternative: no web service while a DoS attack against the server is occurring. While an increase in end-to-end latency by a factor of 5 (or even 10, in the worst case) is considerable, we believe it is more than acceptable in certain environments and in the presence of a determined attack.

V. Related Work

The need to protect against or mitigate the effects of DoS attacks has been recognized by both the commercial and research world. Some work has been done toward achieving these goals, *e.g.,* [6], [7], [8]. However, these mechanisms focus on detecting the source of DoS attacks in progress and then countering them, typically by "pushing" some filtering rules on routers as far away from the target of the attack (and close to the sources) as possible. Thus, they fall into this class of approaches that are reactive. The motivation behind such approaches has been twofold: first, it is conceptually simple to introduce a protocol that will be used by a relatively small subset of the nodes on the Internet (*i.e.,* ISP routers), as opposed to requiring the introduction of new protocols that must be deployed and used by end-systems. Second, these mechanisms are fairly transparent to protocols, applications, and legitimate users. Unfortunately, these reactive approaches by themselves are not always adequate solutions.

Methods that filter traffic by looking for known attack patterns or statistical anomalies in traffic patterns can be defeated by changing the attack pattern and masking the anomalies that are sought by the filter. Furthermore, statistical approaches will likely filter out valid traffic as well. Since the Internet spans multiple administrative domains and (legal) jurisdictions, it is often very difficult, if not outright impossible, to shut down an attack by contacting the administrator or the authorities closest to the source. In any case, such action cannot be realistically delivered in a timely fashion (often taking several hours). Even if this were possible, it is often the case that the source of the attack is not the real culprit but simply a node that has been

remotely subverted by a cracker. The attacker can just start using another compromised node.

Using a "pushback"-like mechanism such as that described in [6] to counter a DoS attack makes close cooperation among different service providers necessary: since most attacks use random source IP addresses (and since ingress filtering is not widely used), the only reliable packet field that can be used for filtering is the destination IP address (of the target). If filters can only be pushed "halfway" through the network between the target and the sources of the attack, the target runs the risk of voluntarily cutting off or adversely impacting (*e.g.*, by rate-limiting) its communications with the rest of the Internet. The accuracy of such filtering mechanisms improves dramatically as the filters are "pushed" closer to the actual source(s) of the attack. Thus, it will be necessary for providers to allow other providers, or even end-network administrators, to install filters on their routers. Apart from the very realistic possibility of abuse, it is questionable whether such collaboration can be achieved to the degree necessary.

[19] proposes using Class-Based Queueing on a web load-balancer to identify misbehaving IP addresses and place them in lower-priority queues. However, most DDoS attacks use spoofed IP addresses that vary over time, thus defeating classification. Even if the same address is used, the amount of state that the load-balancer needs to keep may be prohibitive. Furthermore, many of the DDoS attacks simply cause congestion to the web server's access link. To combat that, the load-balancer would have to be placed closer to the network core. Not only would this further compound the state-explosion problem, but such detailed filtering and especially state-management on a per-source-IP address bassis can have performance implications at such high speeds.

VI. Conclusions

We presented WebSOS, an architecture that allows legitimate users to access a web server in the presence of a DoS attack. We use a combination of cryptographic protocols for authentication, packet filtering, overlay networks, and consistent hashing to provide service to authorized users trying to contact a web server under attack. Our architecture requires no changes to web servers, browsers, or existing protocols.

We evaluated the performance of WebSOS, using our prototype implementation, over both a local area network and over the Internet using PlanetLab. Our measurements show that in a realistic but worst-case deployment scenario the end-to-end communication latency between browser and server increases on the average by a factor of 5, with a worst case of 10. We also discussed various potential optimizations for improving performance. We believe that even at its current level, the overhead imposed is acceptable in many critical environments and applications.

Acknowledgements

This work is supported in part by DARPA contract No. F30602-02-2-0125 (FTN program) and by the National Science Foundation under grant No. ANI-0117738 and CAREER Award No. ANI-0133829, with additional support from Cisco and Intel Corporations. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Alexander Konstantinou's NetCallback was used as a basis for the forwarding code. Abhinav Kamra wrote the CHORD implementation.

References

[1] T. Dierks and C. Allen, "The TLS protocol version 1.0," RFC 2246, January 1999. [Online]. Available: ftp://ftp.isi.edu/in-notes/rfc2246.txt

[2] R. S. Sandhu and J. S. Park, "Decentralized user-role assignment for web-based intranets," in *ACM Workshop on Role-Based Access Control*, 1998, pp. 1–12. [Online]. Available: citeseer.nj.nec.com/sandhu98decentralized.html

[3] L. Gong and R. Schemers, "Implementing Protection Domains in the Java Development Kit 1.2," in *Proceedings of Network and Distributed System Security Symposium (NDSS)*, March 1998, pp. 125–134.

[4] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing for anonymous and private internet connections," *Communications of the ACM (USA)*, vol. 42, no. 2, pp. 39–41, 1999. [Online]. Available: citeseer.nj.nec.com/goldschlag99onion.html

[5] L. Cranor, M. Langheinrich, M. Massimo, M. Presler-Marshall, and J. Reagle, "The Platform for Privacy Preferences 1.0 (P3P1.0) Specifi cation," World Wide Web Consortium (W3C), Tech. Rep., 2002.

[6] J. Ioannidis and S. M. Bellovin, "Implementing Pushback: Router-Based Defense Against DDoS Attacks," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, February 2002.

[7] D. Dean, M. Franklin, and A. Stubblefi eld, "An Algebraic Approach to IP Traceback," in *Proc. of the Network and Dsitributed System Security Symposium (NDSS)*, February 2001, pp. 3–12.

[8] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network Support for IP Traceback," *ACM/IEEE Transactions on Networking*, vol. 9, no. 3, pp. 226–237, June 2001.

[9] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure Overlay Services," in *Proceedings of ACM SIGCOMM*, August 2002, pp. 61–72.

[10] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM*, San Diego, CA, August 2001.

[11] L. Peterson, D. Culler, T. Anderson, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," in *Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I)*, October 2002. [Online]. Available: citeseer.nj.nec.com/peterson02blueprint.html

[12] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith, "Implementing a Distributed Firewall," in *Proceedings of Computer and Communications Security (CCS)*, November 2000, pp. 190–199.

[13] D. Karger, E. Lehman, F. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relievig Hot Spots on the World Wide Web," in *Proceedings of ACM Symposium on Theory of Computing (STOC)*, May 1997, pp. 654–663. [Online]. Available: citeseer.nj.nec.com/karger97consistent.html

[14] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, "Generic Routing Encapsulation (GRE)," RFC 2784, March 2000. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2784.txt

[15] G. Dommety, "Key and Sequence Number Extensions to GRE," RFC 2890, September 2000. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2890.txt

[16] CCITT, *X.509: The Directory Authentication Framework*, International Telecommunications Union, Geneva, 1989.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proceedings of ACM SIGCOMM*, San Diego, CA, August 2001.

[18] S. Miltchev, S. Ioannidis, and A. Keromytis, "A Study of the Relative Costs of Network Security Protocols," in *Proceedings of USENIX Annual Technical Conference (Freenix track)*, June 2002, pp. 41–48.

[19] F. Kargl, J. Maier, and M. Weber, "Protecting web servers from distributed denial of service attacks," in *Proceedings of the World Wide Web Conference*, 2001, pp. 514–524. [Online]. Available: citeseer.nj.nec.com/444367.html