Secure SIP: A Scalable Prevention Mechanism for DoS Attacks on SIP Based VoIP Systems

Gaston Ormazabal[1], Sarvesh Nagpal[2], Eilon Yardeni[2], and Henning Schulzrinne[2]

[1] Verizon Laboratories
gaston.s.ormazabal@verizon.com
[2] Department of Computer Science, Columbia University
{sn2259,ey2125,hgs}@cs.columbia.edu

Abstract. Traditional perimeter security solutions cannot cope with the complexity of VoIP protocols at carrier-class performance. We implemented a large-scale, rule-based SIP-aware application-layer-firewall capable of detecting and mitigating SIP-based Denial-of-Service (DoS) attacks at the signaling and media levels. The detection algorithms, implemented in a highly distributed hardware solution leveraged to obtain filtering rates in the order of hundreds of transactions per second, suggest carrier class performance. Firewall performs SIP traffic filtering against spoofing attacks; and request, response and out-of-state floods. The functionality and performance of the DoS prevention schemes were validated using a distributed test-bed and a custom-built, automated testing and analysis tool that generated high-volume signaling and media traffic, and performed fine grained measurements of filtering rates and load-induced delays of the system under test. The test-tool included SIP-based attack vectors of spoofed traffic, as-well-as floods of requests, responses and out-of-state message sequences. This paper also presents experimental results.

Keywords: SIP, DoS, DDoS, VoIP, Security, Signaling Attacks, Application Layer Firewall, Deep Packet Inspection, Distributed Computing, Scalability.

1 Introduction

Denial-of-Service (DoS) attacks are explicit attempts to disable a target thereby preventing legitimate users from making use of its services. DoS attacks continue to be the main threat facing network operators. As telephony services move to Internet Protocol (IP) networks and Voice over IP (VoIP) becomes more prevalent across the world, the Session Initiation Protocol (SIP) [1] infrastructure components, which form the core of VoIP deployments, will become targets in order to disrupt communications, gain free services, or simply to make a statement. Since DoS attacks are attempts to disable the functionality of the target, as opposed to gaining operational control, they are much more difficult to defend against than traditional invasive exploits, and are practically impossible to eliminate. We designed and demonstrated effective defenses against SIP-specific DoS attacks, with the capability to operate at carrier-class rates. We addressed all four aspects that an effective solution against DoS attacks should cover namely, definition, detection, mitigation, and validation.

H. Schulzrinne, R. State, and S. Niccolini (Eds.): IPTComm 2008, LNCS 5310, pp. 107–132, 2008.

Definition characterizes DoS attacks on the SIP infrastructure, examining the threat taxonomy to identify specific areas that require research focus. *Detection* distinguishes the attack traffic from valid traffic, whereas *mitigation* reduces the impact of DoS attacks on the target infrastructure. Detection and mitigation schemes work in tandem and aim to maintain adequate bandwidth and resources for legitimate traffic, throttle the malicious packets and streams, and perform continued analysis to enhance the detection and mitigation capabilities. V*alidation* of the defense scheme for correct operation, involves modeling the system behavior, building a testing setup capable of generating VoIP DoS attacks, quantifying their impact on protected and unprotected VoIP infrastructure, and measuring the effectiveness of the defense strategies.

This paper examines the SIP threat model and DoS taxonomy in Section 2. An overview of related work is presented in Section 3. This is followed by SIP-specific DoS solutions and filter design in Section 4. The system architecture and implementation aspects are addressed in Section 5. The benchmarking methodology and the **secureSIP** toolkit with the experimental results are covered in Section 6. Conclusions are presented in Section 7.

2 Problem Definition: The SIP Threat Model

This section examines the SIP threat model as the basis for formulating requirements for our detection and mitigation strategies. Since SIP is used on the public Internet, the threat model assumes an environment in which attackers can potentially read any packet on the network. Furthermore, the fact that SIP runs over UDP, provides opportunities for attacks like *spoofing*[1], hijacking, and message tampering. Attackers on the network may also be able to modify packets, perhaps at some compromised intermediary node. We note that the security of SIP signaling, however, is independent from protocols used to secure transmission of media. For example, SRTP (RFC 3711) [2] may be used for end-to-end encryption of the RTP encapsulated audio stream. This section is based on the VoIP Security Alliance (VOIPSA) threat taxonomy report [3] together with definitions in RFC 3261– SIP [1].

There are three basic types of DoS attacks that may occur over a VoIP network, namely, exploitation of implementation flaws, exploitation of application level syntactic vulnerabilities, and flooding of the SIP signaling channel or the RTP media channels. These attacks may target a VoIP component, such as a SIP proxy, or supporting servers, such as a DNS, or a DHCP server. A DoS attack against a supporting server affects the VoIP service in different ways. Attacks against a domain's DNS server result in denial of VoIP calls destined to users in that domain. Attacks against an authorization service, used by a SIP proxy to store address-of-record (AOR) to User Agent (UA) mappings, can result in denial of service to the UAs registering with this proxy. This document, however, focuses exclusively on attacks against SIP-based components. The following sub-sections describes the three basic types of attacks in the SIP-specific context.

[1] Usually referred to as IP spoofing, where an attacker fakes or falsifies the source IP address in a SIP message header.

DoS Due to Implementation Flaws
Attack occurs when a specific flaw in the implementation of a VoIP component is exploited by a carefully crafted packet sent to cause unexpected behavior. The attacked software component, in this case, has typically not been implemented robustly enough to handle these unexpected packets, and also suffers from inadequate software assurance testing or negligent patching. The malformed packet interacts with installed software and may cause excessive memory or disk consumption, extra CPU processing, a system reboot or system crash. The targeted vulnerability may originate in different levels of the network protocol stack, such as the TCP layer or the SIP layer, or in the underlying operating system or firmware [5] and [6]. Examples of implementation flaws attacks include:

***Malformed signaling*:** Unusually long or syntactically incorrect SIP message packets, referred to as "malformed", are sent to the UA degrading its performance, resulting in its inability to process normal setup and teardown messages for calls.

***Invalid call setup messages*:** A number of invalid call set up messages, such as a SIP **ACK** request when none is expected, are sent to cause the endpoint to crash, reboot, or exhaust all of its resources.

DoS Due to Exploitation of Application-level Vulnerabilities
Attack occurs when a feature of the VoIP protocol syntax is manipulated to cause a DoS attack. Examples of application level attacks against SIP-based components include:

***Registration hijacking*:** The SIP registration mechanism allows a UA to identify itself to a registrar as a device whose location is designated by an AOR. Attackers register their devices with other users' AORs, thereby directing all requests for the affected user to the attacker's device.

***Call hijacking*:** Once a dialog has been established, subsequent requests are sent to modify the state of the dialog or session. For example, the attacker injects a **302 Moved Temporarily** message in an active session, thereby hijacking the media session.

***Media sessions modification*:** The attacker spoofs **re-INVITE** messages, thereby modifying security attributes of a session, reducing Quality of Service (QoS), or redirecting media streams to another device for wiretapping.

***Session teardown*:** The attacker spoofs a **BYE** message and injects it into an active session, thereby tearing down the session.

***Amplification attacks*:** The attacker creates bogus requests containing a falsified source IP address, and a corresponding **Via** header field identifying a targeted host, as the originator of the request. Subsequently, the attacker sends this request to a large number of SIP network elements, thereby causing hapless SIP UAs or proxy servers to generate a DoS attack aimed at the target host, typically a server. Similarly, DoS can also be carried out on an individual by using falsified **Route** header field values in a request that identifies the target host, and then sending these messages to forking proxies that will amplify messages sent back to the target. **Record-Route** is used to similar effect when the attacker is certain that the SIP dialog initiated by a request will

result in numerous transactions originating in the backwards direction. An attacker can also register a large number of contacts designating the same host for a given AOR, in order to use the registrar and any associated proxy servers as amplifiers in a DoS attack. Attackers may also attempt to deplete a registrar's available memory and disk resources, by registering large numbers of bindings. Multicast may be also used to transmit SIP requests, greatly increasing the potential for DoS attacks.

Note that if the volume of an application-level DoS attack is sufficient to cause resource depletion, or excessive performance degradation, the attack is reclassified as a *flooding* DoS attack.

DoS Due to Flooding

Attack occurs when a large number of packets are sent to a target IP component; hence any Internet based service is vulnerable to DoS attacks. DoS attacks on services that run on IP represent the broader perspective. The attacker floods the network link by generating more packets than the recipient can handle, or overwhelms the target making it too busy processing packets from the attack and hence unable to process legitimate packets. Flood attacks for IP components include UDP **SYN** floods, ICMP **echo** packets, where the attacker generates a large number of packets directed to the targeted sources. When this attack is done using multiple distributed sources, such as ***botnets***[2], the result is a Distributed DoS (DDoS) [4]. Both the DoS and the DDoS problem for generic IP systems have received a great deal of attention over the years and several commercial products already exist that address this threat. The focus of this work, however, is on DoS, and its corresponding DDoS variety, specifically targeted to VoIP and VoIP-based components, for which currently no protection exists. Flooding DoS attacks to VoIP-based server components can be broadly classified into two categories:

***Signaling floods*:** The most prominent of this category of attacks involves sending a large number of SIP **INVITE** or **REGISTER** messages originating from one or multiple SIP UAs to cause excessive processing at a SIP proxy server - thus delaying or dropping legitimate session establishment messages. There is a computational expense associated with processing a SIP transaction at a proxy server. This expense is greater for stateful than for stateless proxy servers as stateful servers maintain client and server transaction state machines, while stateless do not. Stateful servers are therefore more susceptible to flooding than the stateless type. Floods of messages directed at SIP proxy servers may lock up proxy server resources and prevent desirable traffic from reaching its destination.

***Media floods*:** A range of ports known to be open for legitimate RTP streams are randomly flooded with meaningless and/or un-sequenced packets, over-claiming bandwidth and hindering the RTP QoS.

[2] Botnet describes a collection of software robots, or bots, running autonomously on groups of zombie computers controlled remotely. It also refers to a network of computers using distributed computing software.

3 Related Work

There has been previous effort to protect VoIP deployments from DoS threats. An early evaluation of firewalls for VoIP security was proposed in [7], but it lacked concrete architectural and implementation aspects. A mitigation strategy for flooding DoS attacks on media components using a dynamic pinhole filtering device that blocks all traffic not associated with a legitimate call was previously developed as part of an earlier phase of this research. We designed and built a scalable SIP-aware application layer firewall based on the principle of dynamic pinhole filtering for the RTP streams [8] and [9]. This was the first attempt to combine the SIP proxy with a commercial hardware based, fast packet processing application server, to achieve carrier-class performance and full SIP conformance.

Wu, Y. et al. [10] and Niccolini, S. et al. [11] have applied intrusion detection and prevention mechanisms to safeguard the SIP infrastructure, while the work described in [12] makes use of finite state machines to achieve similar goals. An interesting approach involving VoIP "honeypots" was proposed in [13]. Extensive work on detecting DoS attacks on IP telephony environments has been published in [14], [15], [16], [17] and [18]. Although promising, none of the architectures and algorithms proposed so far offer a comprehensive DoS mitigation strategy that scales up to the performance needs and complexity of carrier-class VoIP deployments, because they are based on software solutions. We are not aware of any specific performance measurements for any of these software based systems. Our solution leverages the CloudShield Technologies CS-2000 distributed hardware platform [18] that combines the processing speed of a distributed network processor platform with the full functionality of a SIP proxy.

4 SIP-specific DoS Solutions and Filter Design

We propose a novel approach that builds on our earlier SIP-aware firewall design, introducing two phases of VoIP traffic filtering, a dynamic pinhole filter (Filter I) for the media traffic, followed by SIP-specific filters (Filter II) for the signaling traffic. Figure 1 gives a high-level view of a SIP security system consisting of these two levels of filtering. Filter I provides the first line of defense by allowing only the signaled media to traverse the firewall, preventing any DoS attacks on the media processing end points. Additionally, it provides standard static filtering for traditional attacks, described as "other attack traffic" in Figure 1, by only allowing traffic on the standard SIP (5060) port. The SIP signaling channel, however, can itself contain SIP-based DoS and hence the motivation forOP Filter II. Filter II, which is comprised of a series of SIP-based filters provide the second line of defense by protecting the SIP signaling port (and thereby the SIP-proxy) from DoS attacks.

This paper covers design, realization, and analysis of SIP-specific filters including a return routability filter, rate-limiting filter and state-validation filter. Together, these filters can protect the SIP infrastructure against known and currently achievable

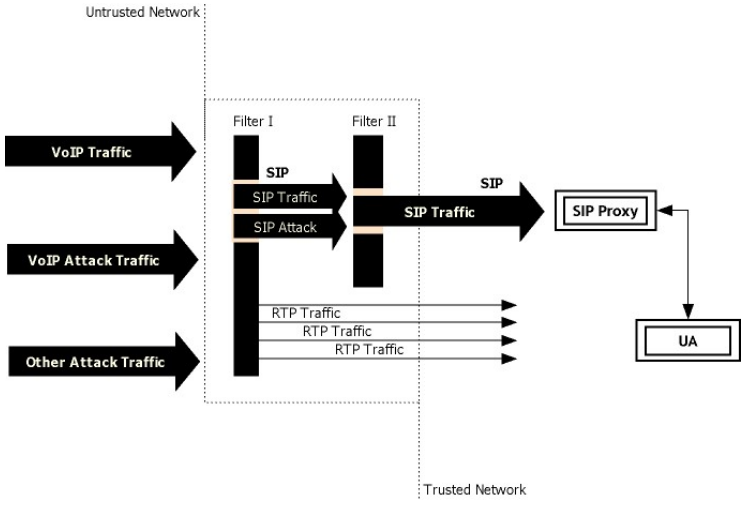

Fig. 1. Two-phase filtering (SIP and media)

spoofing attacks, flood-of-requests and flood-of-response attacks, and "out-of-state" signaling attacks. We built a scalable security system prototype based on the CS-2000 fast packet processing application server, combined with the Columbia SIP Proxy ***sipd***, developed as part of the Columbia InterNet Multimedia Architecture (CINEMA) [19], enabling an effective realization of the proposed SIP security architecture for carrier-class VoIP deployments.

The filters are realized in the deep-packet processing module (DPPM) of the SIP-aware firewall system deployed at a VoIP network perimeter. The DPPM includes very high speed silicon databases that use content addressable memory (CAM) technology for table look-up and keeping state. Additionally, the DPPM is equipped with a regular expression engine used for pattern matching logic in state validation. Some of the filters require the use of a firewall control protocol (FCP) to update state tables in the DPPM, while others result from packet logic manipulation directly on the DPPM, and directly updated on the CAM tables. The filters include a return routability check, and a series of filters based on SIP method manipulation mechanisms that can be used to cause flooding.

Return Routability Filter

The return routability filter is designed to detect and block spoofed incoming requests by using the SIP Digest Authentication[3] mechanism. The SIP protocol specifies that upon receiving a request, other than **CANCEL** and **ACK**, a proxy can challenge the request initiator to provide assurance of its identity. The challenge is sent in a **Proxy-Authorization** header field of a **407 Proxy Authentication Required** response, including a

[3] Digest Authentication provides message authentication and replay protection only, without message integrity or confidentiality.

freshly computed ***nonce***[4] value. The initiator then retries the request with the proper credentials, along with a pre-shared secret[5], in a **Proxy-Authorization** header field.

The proxy responds with the digest authentication challenge whenever it gets a new request, simultaneously instructing the firewall to create a filter rule using the FCP. This firewall filter will then block all further unauthenticated requests from the same IP address from getting to the proxy. If the request originator responds with the correct challenge response, the proxy removes the filter rule from the firewall. The filter is temporary, with a short expiration time on the order of seconds. This process can be viewed as *layer-7-controlled-layer-3- filtering*. An example call flow diagram of the return routability filter operation is shown in Figure 2. The corresponding detailed call flows are in Appendix A.

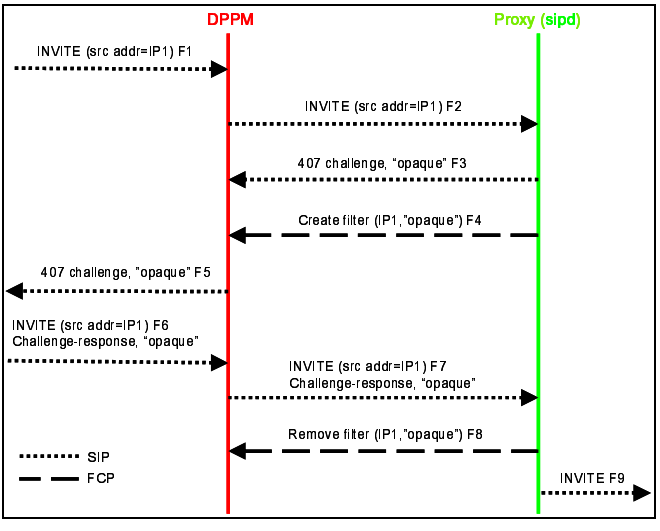

Fig. 2. The call flow for digest authentication

In the call flow described in Figure 2, the DPPM intercepted the first **INVITE** request (F1) with IP1 as the source IP address. The DPPM did not find a match in the filter table and hence forwarded the request to the proxy (F2). The proxy received the **INVITE** request and responded with a **407** message containing the challenge (F3), and also by sending an FCP message (F4) to create a temporary filter rule blocking further requests from IP1. The filter rule was based on the nonce that was part of the authentication challenge, and was expected to be included in the authentication response unchanged. This FCP message was processed by the DPPM and the filter was created. When the

[4] A nonce is a uniquely generated string used for one challenge only, computed using IP address, timestamp, username, password and realm, and has a lifetime of 60 seconds.

[5] SIP allows the use of "null authentication", where a proxy can authenticate an "anonymous" username with no password. The return routability filter was designed based on null authentication, which is a necessary and sufficient condition to establish return routability to the request initiator, avoiding the extra overhead inducing password management process.

new SIP request arrived, the DPPM intercepted it (F6) and tried to match the source IP address with the IP address in the filter table. If there was no match then the request was blocked. Otherwise, if the nonce values were equal, the request was forwarded to the proxy (F7) and the proxy successfully authenticated the **INVITE** request and sent an FCP message (F8) to remove the filter from DPPM. By configuring the proxy not to keep any state until the return routability was verified by the firewall, the possibility of proxy overloading with potentially spoofed request floods could be eliminated.

SIP Method-based Filters

Method-based filters were designed to mitigate attacks that exploit protocol vulnerabilities to cause flood DoS. The design focused on rate-limiting SIP requests and response floods, and also using state validation mechanisms to achieve this.

SIP is a request/response protocol. A request and its associated responses constitute a SIP transaction, which follows the same signaling path through a set of SIP servers. A SIP call, as presented in Figure 3, can be broken down to four levels of granularity. A call is composed of one or more dialogs, while a dialog contains one or more transactions. A transaction can be a client transaction or a server transaction; and each of the client/server transactions can be divided into **INVITE** and non-**INVITE** types.

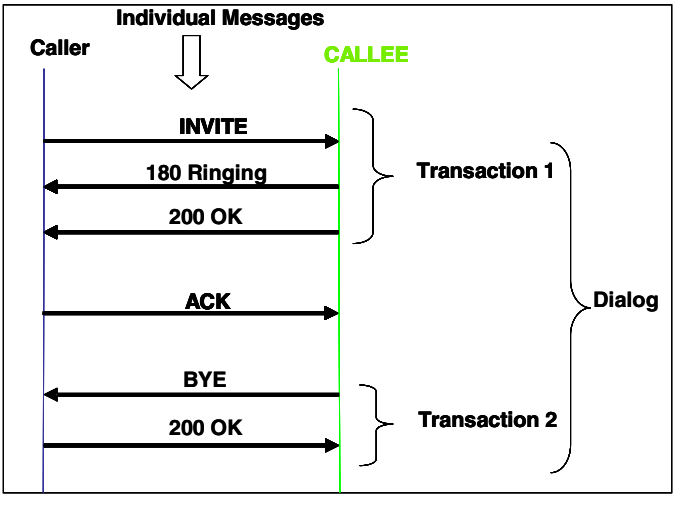

Fig. 3. Levels of granularity in a SIP session

A SIP dialog is identified by a combination of the **Call-ID**[6], **From** tag and **To** tag. A SIP transaction is identified by the **branch**[7] parameter of the **Via** header and the **Method** name in the **CSeq** field. These fields can be used to construct respective ***dialog ID*** and ***transaction ID*** identifiers. Both of these identifiers are used to maintain the corresponding state

[6] Call-ID is a globally unique identifier for a call, generated by the combination of a random string and the phone's host name or IP address.

[7] The branch parameter of the Via header is a unique value across space and time that is created by a UA for a particular and specific request.

information. Rate-limiting can be applied either at the dialog level or at the transaction level; however, for every SIP method except for **BYE** and **CANCEL**, the dialog level does not provide sufficiently precise parameters to perform meaningful thresholding. For example, it may be hard to distinguish a legitimate **INVITE** from a spurious one(s) if they have different transaction IDs. Hence, for every other method, transaction level is the most effective way to narrow down to more specific parameter thresholds for filtering.

Dialog based attacks include **CANCEL** and **BYE** attacks, that can only happen at the dialog level, as both are dialog terminating requests. In a **CANCEL** attack, a spurious **CANCEL** request is sent before the final response of a dialog/transaction, thereby terminating the dialog prematurely, hence causing DoS. **BYE** attacks involve *sniffing*[8] session parameters (such as Call-ID), and generating illegitimate **BYE** requests to terminate an on-going session without knowledge of any of the involved end-clients. To keep track of **BYE** messages, **record-routing** has to be enabled at the proxy. Alternatively, the firewall at the perimeter may be used to identify unsolicited **BYE** messages. In addition to **BYE** message filtering based on dialog ID, a table of all participating URIs must be maintained to verify whether **contact header** field of the **BYE** message corresponds to one of the participating URIs. The **REFER** attack, similar to a man-in-the-middle attack, involves an eavesdropper manipulating the **Referred-By** header to cause DoS. **REFER** attacks can be mitigated by deploying **S/MIME** to detect possible manipulation of the **Referred-By** header data, but are not covered in our current filter design.

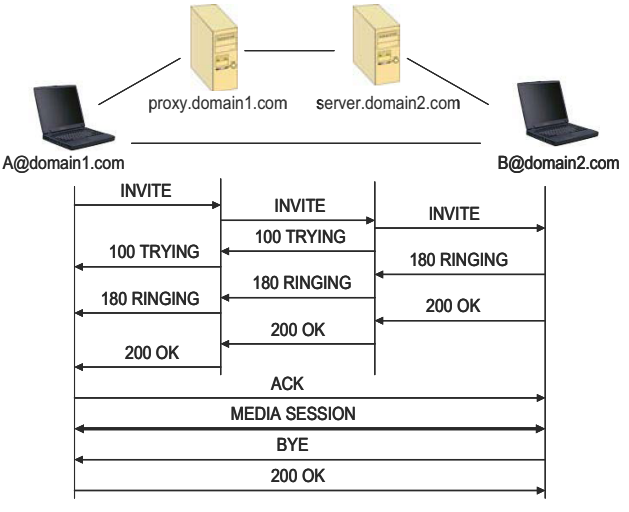

Fig. 4. SIP trapezoid

Transaction-based attacks on a proxy include floods of **INVITE** requests containing same transaction ID, thus causing processing overload. Furthermore, a re-**INVITE** attack can change on-going session parameters by issuing or resending **INVITE** or **UPDATE**

[8] Sniffing the Call-ID in a SIP message is easy to accomplish given the *clear-text* nature of the protocol in its basic form, i.e. non-encrypted.

requests with different parameters. Transaction-based rate-limiting filters detect and mitigate floods of **INVITE** requests with the same transaction ID, and all of their associated responses, to stop them at the perimeter.

The SIP trapezoid, as specified in RFC 3261 and shown in Figure 4, is introduced to describe the method-based rate-limiting filters in more detail. The transactions depicted in the trapezoid are shown in Figure 5 in their client/server relationships. In reference to the interaction between the User Agent Client (UAC) and an outbound proxy, the request is an **INVITE**, and the associated responses are comprised of **100 Trying**, **180 Ringing** and **200 OK**. From the proxy's perspective, this is an **INVITE** server transaction, with the **200 OK** ending the transaction and taking the proxy to **Terminate** state. Accordingly, the messages at proxy are rate-limited to one **INVITE** per transaction (incoming); a finite number of **100 Trying** per transaction (outgoing); a finite number of **180 Ringing** per transaction (outgoing); and one **200 OK** per transaction (outgoing).

The finite number of allowed **100 Trying** and **180 Ringing** messages is flexible and should be decided by different network parameters depending on the complexity of the routed network. To allow for retransmissions, the threshold for **INVITE** and **200 OK** messages may also be raised from one message to a higher finite number that can be experimentally determined from the network configuration under test. Arbitrary messages that do not conform to the above sequence may leave the proxy in an unwanted state. A similar rate-limiting analysis can be applied to the transactions between the outbound proxy and inbound proxy, and User Agent Server (UAS) and inbound proxy. The number of **INVITE**s from a particular UAC is also limited to a single call at a time, or to some particular value based on the size of n-way conferences allowed. For example, if an **INVITE** message is from a particular UAC's IP address already in the CAM table, with its state label being intermediate (**in-progress**), then the new **INVITE** will be rejected. In order to avoid state exhaustion at the proxy, no state will be kept during any of these steps.

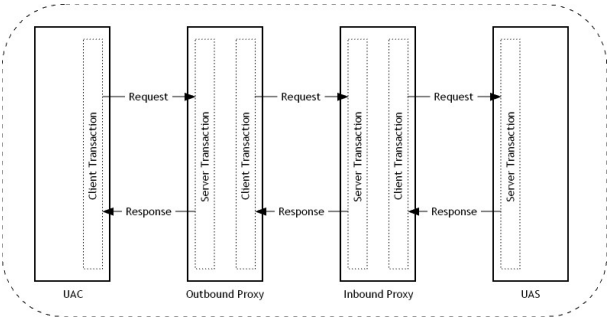

Fig. 5. SIP client-server interaction through inbound and outbound proxy

State Validation Filters

Extending the analysis in the previous section to the transactions between the UAC and outbound proxy, it is not only the rate but also the order of arrival of messages that may leave the proxy in an unexpected state. The schematics in Figures 6 and 7 describe the state machines for **INVITE** client and server transactions, respectively. A detailed description of **INVITE** and **non-INVITE** client/server transactions can be found in [1].

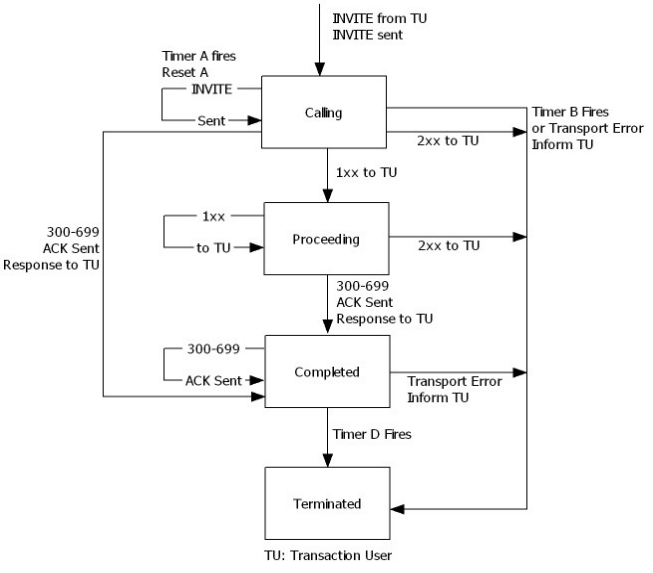

Fig. 6. INVITE client transaction

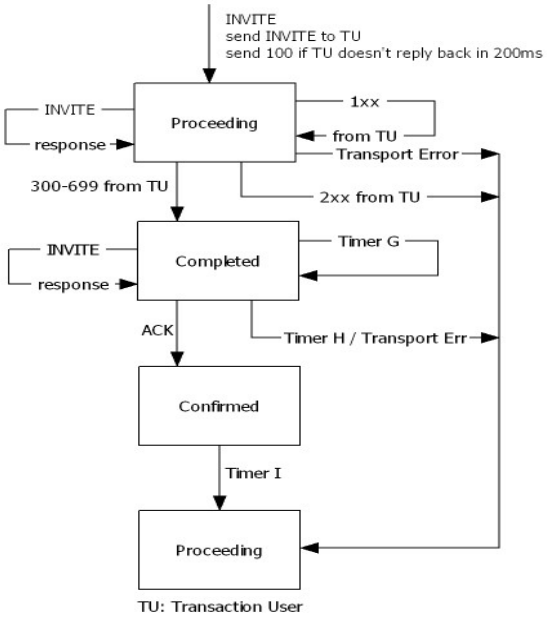

Fig. 7. INVITE server transaction

Using the SIP state machine protocol, it is possible to define the set of expected messages and hence discard the sequences considered out-of-state. The firewall filter will have state tables that point to the current state of a transaction from *{**Proceeding**,*

Completed, ***Confirmed***, ***Terminated****}*, and a set of rules governing the transitions. The table structure has the format *{Transaction ID, Timestamp, State, Acceptable message codes, Next state}*. This table is applicable for both rate-limiting as well as state-validation types of method-based filters.

We have also implemented similar table-driven rate-limiting rules to filter non-standard **1xx** (except **100** and **180**), non-standard **2xx** (except **200)**, and **300-699** responses to a finite number per second, depending on network parameters. This will eliminate specific handling for each of the messages in the range. The non-standard messages are logged in a table having the structure *{Transaction ID, Timestamp, Non-standard message code}*.

Rate-limiting is also performed on **INVITE** requests coming from a single source IP and identical **From** URI, in case of outbound proxy, and **INVITE** requests coming to a single destination IP, and **To URI,** in case of inbound proxy. The timestamp differences between a new **INVITE** and an identical **INVITE** in the above table should be within one second, or else the request is rejected. This is defined in the firewall filter table as *{Source/Destination IP, Timestamp,* ***From/To*** *URI}*.

Lastly, filtering at the dialog level helps the identification of spurious **BYE** messages by using the dialog ID of a message, and rejecting BYE messages that are not part of an existing Dialog. This filtering requires a simple table structure *{Dialog ID, Timestamp}*.

5 System Architecture and Implementation

We deployed an architecture in which the SIP proxy (***sipd***) uses the wire-speed packet processing and CAM capabilities of the CS-2000 server DPPM to boost overall packet-processing capacity. In this section, we describe the architecture and the implementation components as integral modules of the underlying framework.

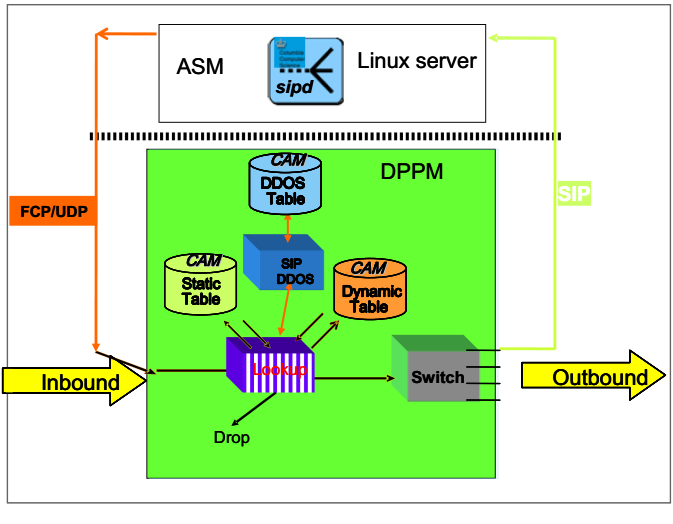

Fig. 8. Architecture components of CloudShield CS-2000

Components required for implementing the architecture shown in Figure 8 include a SIP proxy, data plane execution in the DPPM, and firewall control. The SIP proxy supports "null authentication" and a new FCP message[9] to create/remove a filter from the DPPM using *{IP, nonce}*. The CS-2000 data plane execution modules run filters as applications on DPPM. Filters intercept network traffic and monitor, process, and drop packets using static filtering of pre-defined ports (e.g., SIP, ssh, port 6252), dynamic filtering of legitimately opened ports (e.g., RTP) and a switch layer function performing switching between the input ports. The data plane also includes a return routability filter table, with table entries containing *{IP address, string (nonce)*, *state-label, timeout value}*. Additionally, the data plane features a counter that maintains a count of requests/second for comparison with a pre-determined threshold to detect request floods. When the threshold is crossed, the DPPM starts applying the rate-limiting policy. The DPPM tries to match SIP requests with filters in the state table by matching on dialog ID and transaction ID.

The SIP proxy server runs within the CS-2000 application server module (ASM). The proxy server interacts with the DPPM using the Firewall Control Protocol (FCP) for the return routability filter. The Firewall Control Module, in the SIP proxy, talks with the DPPM, intercepts SIP call setup messages, gets nonce from the **407 Proxy Authentication Required** header, gets RTP ports from the SDP payload and maintains call state, pushes filter for SIP UA (nonce) being challenged, and pushes dynamic table updates to the data plane. FCP can be used by multiple SIP proxies that control one or more CloudShield firewalls. FCP supports the new return routability create/remove filter messages, using the same FCP message format described in [8], with the addition of a random string option to accommodate the nonce.

SIP messages are related using message lookup tables, leveraging the DPPM built-in CAM databases for very low latency lookups. Aged lookup tables are implemented to track call, dialog and transaction relationships using the {dialog ID table, transaction ID table} tuple. Messages are identified by type (request or response) and code (request method or response status code). The "error status message" rate limiter performs error message limiting within the context of a valid transaction. The error rate limiters are implemented as high-speed counters in SRAM, with granularity of one second.

Return Routability Filter

The rate at which the SIP proxy can handle incoming SIP requests is mainly bounded by CPU power. When digest authentication was enabled, this rate decreased, as for every incoming SIP request the proxy had to both process a new challenge and validate the provided authorization data. This process has been thoroughly analyzed in [20] and experimentally verified in our test-bed, as detailed in Section 6 below. An attack flood of spoofed **INVITE** messages can then overload the proxy as the authentication of each spoofed request is attempted. The CS-2000 detected the SIP request floods, and a rate-limiting policy was applied in order to reduce the load from the proxy. The type of rate-limiting policy has a direct impact on the number of false-negatives ("bad" requests that were not blocked) and/or false-positives ("good" requests that were filtered). In the rate-limiting policy suggested in this work, the

[9] A detailed description of the FCP protocol can be found in reference [8].

firewall established a temporary filter, based on IP address and nonce, whenever a new request needs to be authenticated. The filter was used to block any further unauthenticated request attempts coming from the same source, from getting to the proxy. When the proxy got the request, it responded with the digest authentication challenge, and simultaneously issued an FCP message to create the filter in the DPPM. If the request originator successfully responded with the correct challenge response, the proxy removed the filter from the firewall. The filter was also temporary in the sense that it expired after some short period of time on the order of seconds. The filter can be based on the **From** URI or the source IP address.

The detailed design of return routability filters involved the interception of incoming **INVITE** requests at DPPM, and extraction of source IP addresses from the requests. If no corresponding entry for the source IP address was found in the filter's CAM table, the incoming request was forwarded to the proxy. This rule ensured that the first packet from a UA always reached the proxy regardless of filters deployed. After receiving the **INVITE** request, the proxy responded back to the UA with a **407 Proxy Authentication Required** challenge, and also simultaneously sent an FCP message, containing source IP and nonce value, to the DPPM to create a filter table entry. All subsequent **INVITE** requests coming from the same UA were intercepted by DPPM, as before, but at this stage, a corresponding filter entry for this source IP was found to already exist. At that point, if the incoming request contained the same nonce value as previously stored in CAM table filter entry, the request was forwarded to the proxy, and CAM tables were updated to allow all incoming packets from this source IP (white-list), for a short interval of time. In the event of no match, however, the request was dropped right at the perimeter. White-lists are dynamic and the lifetime of each entry was automatically extended with every packet containing the correct nonce. In our experiments, we used thirty seconds for the white-list auto expiry default.

Rate-limiting Filters

Rate-limiting filters required the extraction of the dialog ID (DLGID) and transaction ID (TXNID) from every received SIP request, and their storage in different and subordinate CAM tables. Since dialog ID and transaction ID are variable length fields, a CRC-32 bit hash algorithm was applied in order to generate a fixed length index in the CAM tables, to enable state keeping. DLGID was the 32 bit integer calculated by Hash {**From IP**, **To IP**, **Call ID**} and for every DLGID entry in CAM database, there was a subordinate table for associated TXNIDs. TXNID was the 32 bit integer calculated by Hash {**Top Via: BranchID**, **CSeq Command Value**}. If a TXNID was not found to be duplicated, normal call processing execution continued. If TXNID was found to be duplicated, then the packet was dropped before it reached the proxy. Ideally only one SIP request message should have been allowed per TXNID, however, because of network conditions, the same request may need to be retransmitted multiple times. To allow for this, a window of finite retransmissions before packet drop was implemented and the system trained to find the optimum window length for a given network configuration. The purpose of this window was for optimization to prevent "false positives". A CAM table entry was maintained for each authenticated **INVITE**, and state was incremented for each client so that a filter was put up to accept messages corresponding to only the next allowed state, or any termination message. A timeout filter was also used to terminate a session after a predetermined interval.

Upon receiving a new subsequent status message, if the status message record is valid then the request was accepted, if bogus, the packet was dropped. Additionally, the rate of requests per transaction per second was also checked not to exceed a selected finite number (6), after which packet was dropped. The rate at which messages are received in any state from the session/UA, were limited to a predefined rate, and handled within the state a session/UA is in. Arbitrary error messages at high rates were also blocked if the rate crosses a pre-determined threshold.

SIP Transaction State Validation

This filter validated the state of each SIP transaction for each message received and complemented the other filtering mechanisms. The use of the CS-2000 regular expressions engine allowed validation of every arriving message as "in-state" or "out-of-state" in one CPU cycle, resulting in high scalability and performance. Messages that resulted in invalid states were dropped and the transaction state was always maintained in a legitimate state. The DPPM made an entry for the first transaction request, and logged all subsequent status messages in a buffer, on a per transaction basis. Each received packet was added to the status messages table for the original transaction. If the received status message fit a valid state pattern, it was accepted while if it was an invalid pattern, the message was dropped.

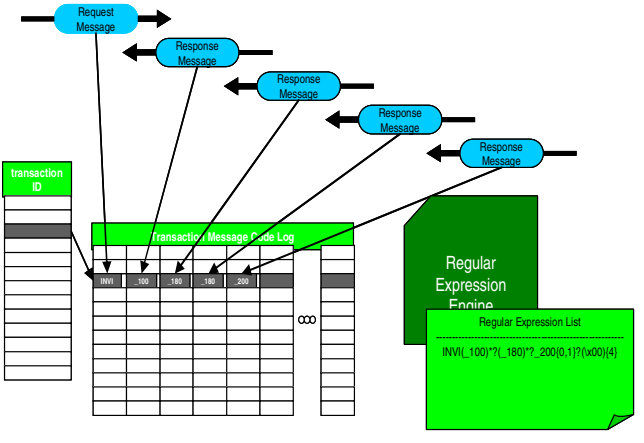

Fig. 9. Regular expressions for request-response transaction

The implementation of the rate-limiting filters, in more detail, involved extraction of the dialog ID and transaction ID from an incoming packet, and comparison with the dialog ID table and subordinate transaction ID table stored in the CAM databases. If a corresponding entry already existed, the message type was entered in a transaction message code log, as shown in Figure 9. The string formed by the sequence of messages {**INVI_100_180_180_200**}, in the example in Figure 9, was matched with the rules list {**INVI(_100)*?(_180)*?_200{0,1}?(\x00){4}**} that codify the SIP state machine pre-stored regular expression rules. The use of wild cards in regular expression syntax afforded validation of all permutations of allowed states in a single operation. If a match was found, the new arriving message was inferred to adhere to the state validation

rules, and allowed to go through to the proxy; otherwise it was discarded, and also removed from the transaction message log, e.g., in the sequence {**INVITE, 100, 180, 200, 180, 200**}; the filters will only allow the sequence {**INVITE, 100, 180, 200**}, while the last {**180, 200**} messages are removed, as the second **180** was already out of state.

6 Benchmarking Methodology

The primary aim of the benchmarking methodology was the verification of correct filter functionality in effectively preventing DoS attacks, and their performance and scalability at carrier-class traffic rates. The security system was verified for its functional accuracy, by developing a novel benchmarking toolkit that provided an extensible and automated interface for testing and analysis based on distributed computing. The test tool generated high-volume SIP sessions, including SIP-based attack vectors of spoofed traffic, as well as floods of SIP requests, SIP responses and out-of-state message sequences. The analysis module presents the data in easy-to-read table form results.

Prior to determining the filters effectiveness, the baseline capacity of the proxy server, for our specific hardware configuration, had to be first established by launching signaling traffic to find the maximum server call handling rate, for a given set of concurrent calls. As described in [21], the call rate handling capacity is directly related to the processing power of the computer hosting the proxy server, and the number of concurrent calls is dependent on the machine's available memory. We evaluated performance of the system with 100,000 concurrent calls of legitimate traffic, as a reference number for comparing performance under different experimental call rates configurations[10].

For our experiments, two proxy setup configurations were used: one without digest authentication, and one with digest authentication enabled. Digest authentication is necessary to distinguish spoofed requests from normal traffic; hence it was also required by the filters design. Our measurements of the difference in baseline capacity of these two setups are in accordance with expected results, and validate the previously reported numbers by Salsano, S., et al. [20]. Since the filters, as designed, rely on digest authentication, we used the maximum performance from this setup as the baseline reference for comparisons against measurements carried out with filters turned on. We begin by describing our test-bed architecture, hardware configuration, attack generation tools and mechanisms, followed by an analysis of the experimental results.

Test-bed Architecture

The generation, measurements and analysis of the SIP DoS attacks were performed in a controlled VoIP test-bed, consisting of hardware and software components used to generate high-volume loads. The test-bed was comprised of an array of seventeen Sun Fire X2100 servers, equipped with an AMD Opteron 2-GHz processor and 2GB of RAM, running Ubuntu Server OS. The test-bed also included the proxy server ***sipd***, resident on the application server module (ASM) of the CloudShield firewall, which consisted of a dual Pentium-III 1-GHz based CPU with 1GB of RAM, running Linux 2.6.17-10.

[10] The number 100,000 was arrived at from performing various experiments to be sufficient to obtain a statistically significant sample.

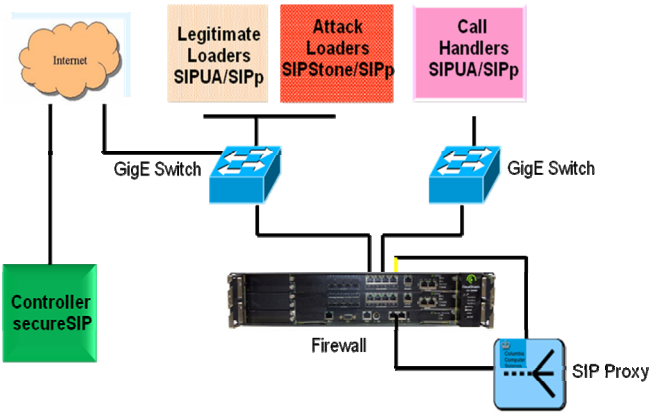

Fig. 10. Test-bed architecture

The setup was equipped with the SIPstone [22] and SIPp [23] suites of SIP traffic generation and benchmarking tools, configured in "loader" and "handler" modes[11]. SIPp is a robust, easily configurable open-source tool, with customizable XML-based scenarios for traffic generation and handling. SIPp uses multiple threads to generate higher call rates per loader-handler pair, as compared to other user agents. In the Sun Fire X2100 server cluster configuration used in these experiments, each loader/ handler pair can generate a maximum of 300 calls per second (CPS). SIPstone is a Columbia-developed signaling test-suite with enhancements for null-digest authentication, and generation of spoofed requests, both capabilities required for these experiments. Each Sun fire X2100 server equipped with SIPstone can generate 1200 spoofed requests/sec in standalone mode.

The seventeen machines in the setup were loaded with both of these test tools to enable a dynamic configuration, and were connected to the CloudShield firewall using GigE switches as shown in Figure 10. One of these machines was configured to host the test-bed controller running **secureSIP**, a web-based control software described in the next section, and the remaining sixteen machines were dynamically configured as traffic generators in loader/handler mode or in individual attack generator mode (e.g., spoofers). Within this distributed setup, network traffic was also captured in real-time using **wireshark**[12] and analyzed [24].

Controller - *secureSIP*

The measurements and validation procedures are controlled by **secureSIP**, a web-based control software using distributed computing processes that provides the tester a user interface to launch, terminate, manage, measure, analyze and store the outcomes of the benchmarking tests as shown in Figure 11.

---

[11] Loaders are used to generate calls, behaving as callers, while handlers receive these calls, thus behaving as callees.

[12] A network protocol analyzer that allows packet capture from live networks, as well as reading packets from saved "capture" files.

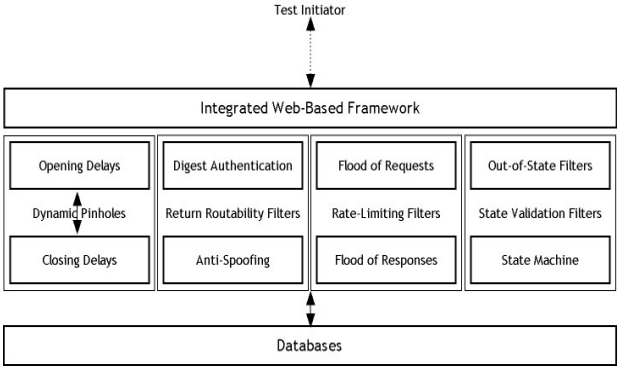

Fig. 11. Architecture of secureSIP controller

Each of the remaining sixteen machines was loaded with **secureSIP** clients, which communicated with the **secureSIP** controller on a predefined channel over UDP (port 6252), to perform the required actions. The **secureSIP** clients used a combination of SIPp, SIPstone and SIPUA (used only for registration [19]). Clients support digest authentication, and were capable of generating spoofed messages, floods of requests/responses, and out-of-state messages to verify performance of return routability, rate-limiting and state validation filters respectively. Each client updated traffic statistics in real time to a central relational database server using MySQL [25]. The data consolidation at one central server facilitated easy correlation, real-time performance analysis, exporting results to spreadsheets and drawing charts to visualize patterns from historical data.

Performance Bench-marking

The first step was the establishment of the test-bed setup baseline, without any security enhancements or attack traffic, defined as the base capacity of the proxy server. These measurements were performed first with digest authentication turned-off, and subsequently enabled. SIPp was used to generate legitimate traffic. After obtaining proxy baseline numbers with digest authentication enabled, attack traffic was introduced into the network and performance of the setup without filters was evaluated. Subsequently, filters were turned on to evaluate the portion of good and attack traffic that was filtered out. For attack traffic generation, SIPstone was used to create spoofed call attempts. Floods of requests, responses and out-of-state messages were generated using SIPp. The protocol analyzer was also used to analyze the flow of network packets to estimate the proportion of dropped calls that were part of legitimate or attack traffic respectively. The validation and measurements were all performed at two different loads; at full capacity, to determine the maximum performance of the tested configuration as a reference point, and at half capacity of the proxy to cover the typical workloads in a carrier-class VoIP service.

Base Capacity

Base capacity was determined by generating legitimate traffic through SIPp, using multiple pairs of loaders and handlers, controlled in an automated fashion by the

secureSIP controller. Base capacity was found by incrementing call rate until the proxy was unable to respond to all the incoming requests, dropping legitimate calls. As each loader/handler pair was able to generate 300 CPS, the load was incremented pair by pair until the base capacity for proxy setup without authentication was found to be 690 CPS, with three pairs. Network traffic analysis using **wireshark** also confirmed this base capacity.

Using the same methodology, the digest authentication mechanism was enabled, and the new base capacity was found. The load on proxy server was incremented pair by pair, finding the new base capacity at 480 CPS, using two pairs. The results showed the call handling capacity of the proxy dropped from about 690 CPS to 480 CPS. Considering the proxy server was operating in stateful mode, these results validate the analysis and measurements published in [20], although we present the results at an order of magnitude higher call rates. The observed call drop is attributed to the extra processing required for computation of nonce and hashing, and the extra SIP messages that are introduced into the network as previously shown in Figure 2. Since computation of hashing algorithms causes only 30% of the overhead, the main reason for the drop (70%) in performance is due to the extra messages that digest authentication introduces into the network. For all our subsequent measurements below, we assumed digest authentication to be enabled, and comparisons of call rate handling capacity for various filters are always made against this benchmark of 480 CPS.

Methodology for Filter Effectiveness Validation and Measurement

The next three sub-sections provide a detailed treatment of DoS through spoofing, method-based flooding and composite attacks. These sets of experiments measured the impact of the DoS attacks on the unprotected SIP infrastructure and evaluated the effectiveness of the firewall filters in preventing these attacks.

DoS through Spoofing

These tests verified the operation of the return routability filters. The setup was similar to the performance benchmarking section, with SIPp generating legitimate traffic, and SIPstone used to launch traffic with spoofed addresses. Incremental spoofed traffic attacks were launched under two different workloads, at full capacity (480 CPS) and half capacity representing average load conditions (240 CPS). As expected, the digest authentication mechanism was able to remove the spoofed traffic; however, the performance penalty was such that even at half capacity, the proxy was only able to process 3000 spoofed attempts per second, before collapsing. The return routability filters, however, once enabled, dropped spoofed calls right at the perimeter, thus saving the proxy server from processing the additional messages. Our measurements show that the filters removed all of the 16800 spoofed attempts per second generated by our test-bed, at its maximum workload configuration. It should be noted here that this maximum number is *the limit of our test-bed configuration* and not the limit of the firewall.

DoS through SIP Method-Based Attacks

These tests verified the operation of the rate-limiting and state-validation filters. Method-based attacks included three sub-types, consisting of floods of repetitive

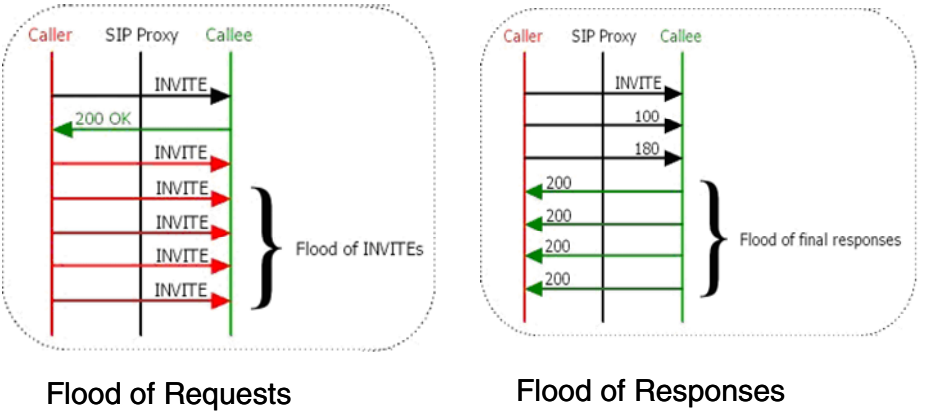

Fig. 12. Different types of rate-limiting attacks

requests, repetitive responses and various sequences of out-of-state messages. The proxy was subjected to these types of attacks, with and without the corresponding filters. We defined three types of attacks – request flood, response flood and out-of-state flood.

The first attack consisted of sending a flood of **INVITE** requests (exact replica of each other, with same transaction ID) after the call was setup with the initial request. The second type consisted of sending a barrage of responses (any of **1XX Provisional**, **2XX Success** or **4XX Error**). The last type consisted of flooding the proxy with requests/responses sequences in random order. For all three types of attack traffic, the flood packets that follow the first packet will have the same transaction ID, as seen in the call-flow diagrams schematic view in Figure 12. SIPp loader/handler pairs were used to generate both legitimate and attack traffic for these measurements.

DoS Filters Performance Results

Measurements from the different test scenarios, including benchmarking, return routability filters and rate-limiting filters are summarized in Table 1 below. The array of sixteen machines was used to generate the high volumes of different types of legitimate as well as attack traffic. As observed in Table 1, in general, the inbuilt software mechanisms in the SIP proxy provide negligible performance against the attack traffic in the absence of filters. In particular, the proxy server, without the benefit of filters, breaks down with fewer than 200 spoofed requests, when already at maximum, but even at half load, the proxy is only able to handle less than 3000 spoofed attempts per second. In the same setup, but with filters turned on, the performance increased considerably, to well over 17,000 spoofed attempts per second. As noted earlier, the amount of attack traffic handled in these experiments was determined by our specific test-bed hardware constraints, and not by the capacity of the filters.

The effectiveness of the rate-limiting filters can be assessed by comparing results in similar setup initially without the filters, with results with filters enabled. While setup without filters can only deal with a maximum flood of fewer than 600 calls (requests) per second, filters pushed the handling capacity to over 7,000 attacks per

Table 1. Measurements from different test scenarios

	Firewall Filters OFF			Firewall Filters ON		
	Good CPS	Attack CPS	CPU Load	Good CPS	Attack CPS	CPU Load
Traffic Composition			%			%
Non-Auth Traffic	690	0	88	690	0	88
	240	0	20	240	0	40
Auth Good Traffic	480	0	81	480	0	82
Auth Good Traffic +	240	2950	84	240	16800	41
Spoof Traffic	480	195	85	480	14400	83
Auth Good Traffic +	240	3230	84	240	8400	41
Flood of Requests	480	570	86	480	7200	83
Auth Good Traffic +	240	2970	87	240	8400	41
Flood of Responses	480	330	87	480	7200	83
Auth Good Traffic +	240	2805	86	240	8400	40
Flood of Out-of-State	480	290	85	480	7200	82

second. Even at average normal load settings, at half the capacity of proxy server, the results without filters were not impressive, as the proxy could only handle up to 3,000 attacks per second. For measurements involving attack traffic comprised of floods of responses, or floods of out-of-state messages, the performance without filters was slightly worse, as the proxy collapsed around 300 attacks per second, when at maximum load. At half load, 2,800 attacks per second could be handled. The addition of filters, however, enhanced the proxy capacity to over 8,000 attacks per second, which again was the maximum attack traffic we could generate in our hardware configuration.

Furthermore, we measured zero false positives and negligible false negatives. Through protocol analysis, we could confirm that none of the legitimate traffic was dropped while the filters dropped 99% of attack traffic, leading to 1% false negatives. The filter algorithm is adaptive, and requires training, based on network conditions, before it can isolate bad traffic from good traffic. Due to this adaptive nature of filters, some amount of attack traffic manages to pass through filtering system, giving a rate of false negatives of 1%. Additionally, since the filters did not drop any packet before they were trained, rate of false positives was zero.

DoS through Composite Attacks

To test our DoS prevention mechanisms against extreme but perhaps more realistic scenarios, as attackers will attempt every attack permutation at once, all the above described attacks were launched together. Different **secureSIP** clients in the distributed network were configured to launch different types of attack traffic. For instance, the sixteen machines in the network could be configured, such that six machines generated spoofed traffic, four machines flooded the network with requests, two machines introduced out-of-state messages and the remaining four were used to generate legitimate traffic. The proxy was subjected to this composite attack, initially with no filters,

and subsequently with all the filters loaded on the firewall. All the other measurement and traffic generation conditions were kept the same.

As seen in the measurement results in Table 2, a proxy conforming to the protection mechanisms specified in [1] was unable to withstand composite attacks. Without filters, even at half capacity, the proxy was only able to handle less than 1000 CPS of different types of attack traffic before it started dropping legitimate calls within a few seconds of the attack (18 seconds in this specific instantiation). At maximum capacity, the results were much worse, and showed practically no tolerance for any kind of attack traffic. Once filters were enabled, the proxy dealt efficiently with as much attack traffic as could be generated in our test-bed. Both at half load capacity, and full capacity, the proxy server showed no sign of performance lag, operating with reasonable CPU resources. Even in case of composite attacks, we observed zero false positives and almost negligible false negatives with filters turned on.

Table 2. Performance measurements of composite attacks

	Traffic Rate (CPS)				
Filters	Good	Spoof	Flood of Req	Out of State	Avg. CPU (%)
Off	240	800	800	800	85
On	240	7200	2400	2400	42
Off	480	100	100	100	87
On	480	4800	2400	2400	83

Benchmarking Summary
The CPU resource consumption increases linearly with the increased attack traffic when the firewall filters are disabled. But once enabled, the filters off-loaded the proxy of all the attack traffic, as evidenced by the non-increasing resource consumption versus attack traffic load, in all of the measurements. When all filters were enabled, they worked together to protect the system under test from a variety of attack traffic. No perceivable performance loss or overhead was observed in the SIP proxy, even at the peak of the attack traffic, clearly indicating that the hardware filters had removed the attack traffic completely.

7 Conclusions and Future Work

The solution presented in this work experimentally demonstrated various SIP vulnerabilities that may potentially result in DoS attacks. As perimeter security is becoming a factor of prime importance to VoIP service providers and carriers, this work suggests highly scalable detection and mitigation strategies against these new SIP-specific DoS attacks. This implementation leveraged a fast parallel processing packet-processing server using CAM databases for storing the huge connection state tables associated with high volumes of concurrent calls, while providing full SIP conformance. A large-scale distributed test-bed, including a high-powered SIP-specific DoS attack tool, was built to measure and verify the effectiveness and scalability of the

solution. The web-based controller, also developed as a part of this framework, provides an effective tool-kit for easy use in testing laboratories. The prototype filtering handling capacity presented, with rates in the hundreds of calls per second, is indicative that these systems can be utilized in carrier class environments. In the short term, enhancements that cover a broader range of attack cases may be desirable. For example, floods of **INVITE**s (and/or responses) with *different* transaction IDs within dialogs, is a closely related, but harder, problem that needs further study. Longer term, the application of anomaly detection, pattern recognition and learning systems, will also be desirable for future systems based on the concepts developed in this work.

The methodologies described in this paper, are applicable to wireline and wireless topologies, and can be extended to secure emerging technologies such as Internet Multimedia Systems (IMS), as well as presence and unified communications infrastructures. Other efforts also continue to extend SIP to support Presence, Messaging and Unified Communications such as Web Services SIP (WSIP), leveraging the dual http and SIP stacks, to allow for reliable unified communication services. The work presented in this paper, may also help achieve secure end-to-end communication for these services.

Acknowledgements

We would like to thank Stu Elby, Vice President of Network Architecture, Chris Mayer, Vice President of Systems Integration and Testing, and Mike Daigle, Vice President of Network Planning, at Verizon Technology Organization, for the sponsorship of this work, and continued interest and support. We would also like to thank Robert Ormsby, David Dumas, James Flowers and Haidar Chamas, also at Verizon Technology Organization for their continued support and encouragement. Haidar also offered extensive commentary on the paper. David Helms from CloudShield Technologies provided excellent technical support in the implementation and fine-tuning of the filters. At Columbia University we would like to thank Jonathan Lennox, the primary architect of our SIP proxy ***sipd***, and Sankaran Narayanan the primary architect of our benchmarking tool, SIPStone, for their contributions.

References

[1] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol, RFC 3261 (June 2002)

[2] Baugher, M., McGrew, D., Naslund, M., Carrara, E., Norrman, K.: The Secure Real-time Transport Protocol (SRTP), RFC 3711 (March 2004)

[3] VOIPSA VoIP Security and Privacy Threat Taxonomy, `http://www.voipsa.org/Activities/VOIPSA_Threat_Taxonomy_0.1.pdf`

[4] Worldwide, I.S.P.: Security Report, Arbor Networks (September 2005), `http://www.arbor.net/downloads/Arbor_Worldwide_ISP_Security_Report.pdf`

[5] CERT Advisory CA-, -06 Multiple vulnerabilities in implementations of SIP (2003), `http://www.cert.org/advisories/CA-2003-06.html`

[6] Wieser, C., Laakso, M., Schulzrinne, H.: Security testing of SIP implementations. Technical Report (February 20, 2005), http://www1.cs.columbia.edu/~library/TRrepository/reports/reports-2003/cucs-024-03.pdf
[7] Roedig, U., Ackermann, R., Steinmetz, R.: Evaluating and Improving Firewalls for IP-Telephony Environments. In: IP-Telephony Workshop (IPTel) (April 2000)
[8] Yardeni, E., Schulzrinne, H., Ormazabal, G.: SIP-aware Application Layer Firewall with Dynamic Pinholes for Media, Columbia Technical Report (2006), http://www.cs.columbia.edu/~hgs/papers/Yard06_Large.pdf
[9] Yardeni, E., Patnaik, S., Schulzrinne, H., Ormazabal, G., Helms, D.: SIP-aware Application Layer Firewall with Dynamic Pinholes for Media, NANOG 38 (October 2006), http://www.nanog.org/mtg-0610/mcbride.html
[10] Wu, Y., Bagchi, S., Garg, S., Singh, N., Tsai, T.K.: Scidive: A stateful and cross protocol intrusion detection architecture for VoIP environments. In: International Conference on Dependable Systems and Networks (June 2004)
[11] Niccolini, S., Garroppo, R.G., Giordano, S., Risi, G., Ventura, S.: SIP Intrusion Detection and Prevention: Recommendations and Prototype Implementation. In: IEEE Workshop on VoIP Management and Security (April 2006)
[12] Sengar, H., Wijesekera, D., Wang, H., Jajodia, S.: Intrusion Detection Through Interacting Protocol State Machines. In: International Conference on Dependable Systems and Networks (2006)
[13] Nassar, M., State, R., Festor, O.: VoIP Honeypot Architecture. In: IEEE International Symposium on Integrated Network Management (May 2007)
[14] Chen, E.Y.: Detecting DoS Attacks on SIP Systems. In: IEEE Workshop on VoIP Management and Security at NOMS (April 2006), http://www.comsoc.org/confs/noms/2006/docs/14_Chen.ppt
[15] Sengar, H., Wijesekera, D., Wang, H., Jajodia, S.: Fast Detection of Denial-of-Service Attacks on IP Telephony. In: IEEE International Workshop on Quality of Service (June 2006)
[16] Geneiatakis, D., Dagiouklas, A., Kambourakis, G., Lambrinoudakis, C., Gritzalis, S., Ehlert, S., Sisalem, D.: Survey of Security Vulnerabilities in Session Initiation Protocol. IEEE Communications Surveys and Tutorials 8(3) (2006)
[17] Sisalem, D., Kuthan, J., Ehlert, S.: Denial of Service Attacks Targeting a SIP VoIP Infrastructure- Attack Scenarios and Prevention Mechanisms. IEEE Network Special Issue on Securing VoIP 20(5) (2006)
[18] CloudShield,CS- (2000), http://www.cloudshield.com/Products/cs2000.asp
[19] Columbia InterNet Extensible Multimedia Architecture (CINEMA), http://www.cs.columbia.edu/IRT/cinema
[20] Salsano, S., Veltri, L., Papalilo, D.: SIP security issues: the SIP authentication procedure and its processing load. IEEE Network 16(6) (2002)
[21] Singh, K., Schulzrinne, H.: Failover and load sharing in SIP telephony. In: International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Philadelphia, Pennsylvania (July 2005), http://www1.cs.columbia.edu/~kns10/publication/sipload.pdf
[22] Schulzrinne, H., Narayanan, S., Lennox, J., Doyle, M.: SIPstone - benchmarking SIP server performance. sipstone 0402.pdf (April 2002), http://www.sipstone.org/files/
[23] SIPp, http://sipp.sourceforge.net
[24] wireshark, http://www.wireshark.org/docs/man-pages/wireshark.html
[25] MySQL, Open Source SQL server, http://www.mysql.com

Appendix A

Call Flows during Digest Authentication, as seen in Figure 2

The first INVITE message received by proxy server from user agent does not contain any credentials.

F1 INVITE UA -> Proxy

```
INVITE sip:test1@cs.columbia.edu SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1:7898
Max-Forwards: 70
From: sip:test5@cs.columbia.edu
To: sip:test1@cs.columbia.edu
Contact: sip:test5@127.0.0.1:7898;transport=UDP
Subject: SIPstone invite test
CSeq: 1 INVITE
Call-ID: 1736374800@lagrange.cs.columbia.edu
Content-Type: application/sdp
Content-Length: 211
v=0
o=user1 53655765 2353687637 IN IP4 128.3.4.5
s=Mbone Audio
t=3149328700 0
i=Discussion of Mbone Engineering Issues
e=mbone@somewhere.com
c=IN IP4 128.3.4.5
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

After receiving the first INVITE message, the proxy sends back a “407 Authentication Required” asking user for authentication. This message contains a freshly computed nonce value that must be sent back by user to prove their identity.

F3 407 Proxy Authentication Required Proxy -> UA

```
SIP/2.0 407 Proxy Authentication Required
Via: SIP/2.0/UDP 127.0.0.1:7898
From: sip:test5@cs.columbia.edu
To: sip:test1@cs.columbia.edu; tag=2cg7XX0dZQvUIlbUkFYWGA
Call-ID: 1736374800@lagrange.cs.columbia.edu
CSeq: 1 INVITE
Date: Fri, 14 Apr 2006 22:51:33 GMT
Server: Columbia-SIP-Server/1.24
Content-Length: 0
Proxy-Authenticate:Digest
realm="cs.columbia.edu",
        nonce="6ydARDP51P8Ef9H4iiHmUc7iFDE=",
        stale=FALSE,
        algorithm=MD5,
        qop="auth,auth-int"
```

User replies back to the “407 Authentication Required” challenge by providing authorization credentials, the nonce value to the proxy server:

F6 INVITE UA -> Proxy

```
INVITE sip:test1@cs.columbia.edu SIP/2.0
Via: SIP/2.0/UDP 127.0.0.1:7898
Max-Forwards: 70
From: sip:test5@cs.columbia.edu
To: sip:test1@cs.columbia.edu
Contact: sip:test5@127.0.0.1:7898;transport=UDP
Subject: SIPstone invite test
CSeq: 3 INVITE
Call-ID: 1736374800@lagrange.cs.columbia.edu
Content-Type: application/sdp
Content-Length: 211
Proxy-Authorization:Digest
username="anonymous",
realm="cs.columbia.edu",
nonce="6ydARDP51P8Ef9H4iiHmUc7iFDE=",
uri="sip:test1@cs.columbia.edu",
response="0480240000edd6c0b64befc19479924c",
opaque="", algorithm="MD5"

v=0
o=user1 53655765 2353687637 IN IP4 128.3.4.5
s=Mbone Audio
t=3149328700 0
i=Discussion of Mbone Engineering Issues
e=mbone@somewhere.com
c=IN IP4 128.3.4.5
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```