ABSTRACT

PERFORMANCE EVALUATION OF SIP AUTHENTICATION AND TLS

By

Swapna Maguluri

August 2012

The Session Initiation Protocol (SIP) is an application-layer control protocol that can establish, modify, and terminate multimedia sessions (conferences) such as Internet telephony calls and it is defined in the RFC3261. SIP is vulnerable to significant risks and vulnerabilities as the signaling is done over open and highly insecure Internet and SIP also offers user mobility. The massive deployment of Voice over Internet Protocol (VoIP) had raised the importance of the security and more precisely of the underlying signaling protocol SIP.

In this thesis, we have studied the various security risks to SIP and various security mechanisms used with SIP to mitigate those risks. We have also evaluated the impact on performance of SIP registrar and proxy servers due to the overheads imposed by SIP authentication and use of Transport Layer Security (TLS) with SIP. The performance impact is evaluated using an experimental test-bed comprising of an Open Source SIP Server (OpenSIPS) and an open source SIP performance (SIPp) bench-marking tool. We have also profiled the system costs in TLS using the OProfile utility of Linux.

PERFORMANCE EVALUATION OF SIP AUTHENTICATION AND TLS

A THESIS

Presented to the Department of Computer Engineering and Computer Science

California State University, Long Beach

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Engineering

Committee Members:

Tracy Bradley Maples, Ph.D. (Chair)
Burkhard Englert, Ph.D.
Min He, Ph.D.

College Designee:

Burkhard Englert, Ph.D.

By Swapna Maguluri

B.Tech., 2004, KLCE, Vijayawada, India

August 2012

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my thesis advisor Dr. Tracy Maples for her excellent guidance and support. Her knowledge, ideas, innovative thinking, and encouragement have been of immense help to me in carrying out the thesis work.

Next, I would also like to express my sincere gratitude to Dr. Burkhard Englert and Dr. Min He for being members of my thesis committee.

Finally, I would also like to thank my husband, parents, and friends for their constant patience, support, and encouragement during the course of this thesis and the Masters degree.

TABLE OF CONTENTS

CHAPTER Page

APPENDICES Page

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER 1

INTRODUCTION

VoIP refers to voice communications over Internet Protocol (IP) networks. It specifies the transmission and reception of audio over the IP based networks. The voice signals are digitized, compressed, packetized (converted to IP packets) and transmitted over the IP network. The receiving node will reproduce the voice signals by de-packetizing, de-compressing, and converting digital data to analog signals. VoIP has gained tremendous acceptance and is widely deployed today mainly due to the reduced costs, demand for multimedia communications, and demand for convergence of voice and data networks.

VoIP uses signaling protocols to establish and tear down calls, carry information required to locate users, and negotiate capabilities. Two standards have emerged for signaling and control for Internet Telephony. One is H.323 defined by International Telecommunication Union (ITU), and the other is Session Initiation Protocol (SIP) defined by the Internet Engineering Task Force (IETF). H.323 embraces the more traditional circuit-switched approach to signaling based on the Integrated Services Digital Network (ISDN) Q.931 protocol and earlier H-series recommendations where as SIP favors the more lightweight Internet approach based on Hyper Text Transfer Protocol (HTTP) [1]

Problem Statement

As with any Internet based service, the security of voice calls is a major issue. The nature of service that SIP provides makes security an even more important feature. SIP based IP telephony system is vulnerable to general Internet attacks, as well as attacks which are unique to SIP. Some of SIP specific security attacks are *registration hijacking*, *message modification*, *impersonation of a server*, *Denial of Service (DoS) attacks, tearing down Sessions*, and *replay attacks*. As most of SIP development so far has focused on features and inter-operability, there exists ample opportunity to work on SIP security. SIP employs security mechanisms like Authentication, and Transport Layer Security (TLS) to counter the above attacks. Before deploying a VoIP network using SIP, it is very important for the network planners and administrators to understand the possible security threats, how to use the available security mechanisms, limitations of the existing security mechanisms, and the performance impact on SIP servers due to these security mechanisms. There is not a lot of emphasis put into understanding the impact of these security mechanisms on SIP server performance [2] [3][4][5].

In this thesis, we have studied the various security mechanisms employed by SIP and their limitations. We have also evaluated the impact on performance of SIP servers due to the Authentication and use of Transport Layer Security (TLS).

Organization of Thesis

The remainder of this thesis is organized in 6 chapters. We started with an overview of VoIP and SIP in Chapter 2. The Chapter 3 discusses SIP security risks. The

security mechanisms employed in SIP are described next in details in Chapter 4. The Chapter 5 describes the test setup and test tools used for evaluating the performance impact of SIP authentication and use of TLS with SIP. The Chapter 6 provides the performance results obtained using the experimental test setup and discusses the impact of SIP authentication and TLS on SIP server performance. Finally, the conclusions are provided in Chapter 7.

CHAPTER 2

OVERVIEW OF VOIP AND SIP

VoIP introduces the actual method of transmitting voice over an IP network and IP telephone. It describes telephony devices that use IP as the native transport for voice and call signaling. IP telephony needs VoIP to send calls over the network.

SIP is widely used today in VoIP as it is a lightweight approach best suited for packet switched IP networks. Note that SIP is only a signaling protocol and not the media transport protocol. For media transport, different VoIP implementations may employ different protocols but the mostly used media transport protocol for multimedia communications over IP networks is the Real Time Protocol (RTP). This RTP is defined by the IETF in Request For Comments (RFC) 3550. The payload format for various COder and DECoders (CODECs) supported by RTP are defined in RFC 3551. The RFC 3550 also defines the Real Time Control Protocol (RTCP) which helps addressing the issues related to delay and jitter in voice communications.

VoIP Technology

As shown in Figure 1 below, the Digital Signal Processor (DSP) segments the voice signal into frames by digitizing the voice signals, compressing the digital data, and framing the voice packets. These voice packets are then embedded into voice transport protocol like RTP or User Datagram Protocol (UDP) and are routed over the Internet

using IP in compliance with the International Telecommunications Union-Telecommunications (ITU-T) specification H.323, the specification for transmitting multimedia (voice, video, and data) across a network.

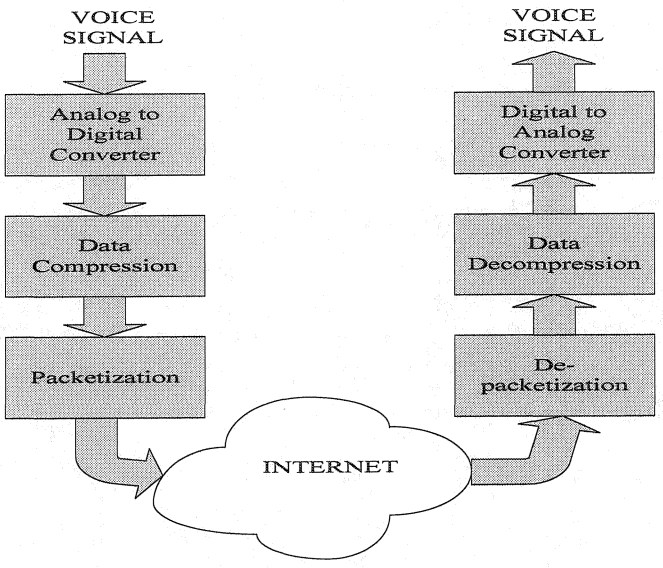

FIGURE 1. VoIP data processing.

Because it is a delay-sensitive application, you need to have a well-engineered, end-to-end network to successfully use VoIP. Fine-tuning your network to adequately support VoIP involves a series of protocols and features to improve Quality of Service (QoS). Traffic shaping considerations must also be taken into account to ensure the reliability of the voice connection.

A connection between a caller and a call recipient is established using a signaling protocol, usually SIP. SIP has many functions, including negotiating the CODECs used

during the call, transferring calls, and terminating calls. During a peer-to-peer call, VoIP phones communicate directly over IP and stream audio directly.

SIP Introduction

SIP is the IETF standard for IP telephony and it is defined in RFC 3261 as an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants [6]. These sessions include Internet telephone calls, multimedia distribution, and multimedia conferences. It seems to be the most promising candidate for call setup signaling for future IP-based telephony services, and it has been chosen by the Third-Generation Partnership Project (3GPP) as the protocol for multimedia application in 3G mobile networks [4].

VoIP requires creation, termination, and management of an audio session. There are many protocols that can provide the session management. However, VoIP signaling protocol has requirements of establishing, terminating, and managing audio sessions with the users whose location, availability, and capabilities can be dynamic. SIP is designed to work well in setting up and managing sessions even when users move between end points, have many different names, and have different capabilities.

SIP Functionality

SIP is developed as an agile, general-purpose tool for creating, modifying, and terminating sessions that works independently of underlying transport protocols and on the type of session being established [6]. SIP provides the capability for one end point (called user agent) to determine the location of another end point and to share one's

capabilities with the other and to agree on characteristics of the session being established. It uses SIP elements called proxy servers to help route requests to the user's current location, authenticate and authorize users for services, implement provider call-routing policies, and provide features to users. It provides the user location by using registration servers with which all the user agents of a domain will register with. The registration is then used by SIP proxy servers for routing incoming SIP requests. When the called user agent is in a different domain, SIP proxy server will contact SIP proxy server of the other domain for user location. SIP proxy server will request the registrar server for determining the user location by inputting a Universal Resource Identifier (URI) and receive back a set of zero or more URIs that help in forwarding the incoming requests. Note that registrations are one way to create the user location information database, but not the only way. The administrator can configure arbitrary mapping functions at his or her discretion.

As defined in RFC 3261, SIP supports five facets of establishing and terminating multimedia communications: user location, user availability, user capabilities, session setup, and session management [6]. The *user location* is about determining the end system to be used for communication. The *user availability* is about determining the willingness of the called party to engage in communications. The *user capabilities* is about the determining the media and media parameters to be used. The *session setup* is about establishing the session between the called and calling parties. And the *session management* is about invoking the services, managing the transfer and termination of services, and modifying session parameters.

It is very important to note that SIP is one protocol among many protocols needed to build a multimedia architecture like VoIP. SIP does not do conference control or resource reservation or session description or real time media transport. Each of these functions is done by a separate protocol. VoIP typically uses SIP for session management, Session Description Protocol (SDP) for describing multimedia sessions, RTP for media transport, RTCP for QoS feedback, Real Time Streaming Protocol (RTSP) for control over the delivery of data with real-time properties, and MGCP (Media Gateway Control Protocol) for controlling Public Switched Telephone Network (PSTN) gateways.

SIP Elements

There are mainly three types of elements in SIP operation: User Agent Clients (UAC), User Agent Servers (UAS), and User Agents (UA). SIP RFC defines these elements as following.

User Agent (UA): The entities interacting in a SIP scenario are called User Agents. A user agent is defined as a logical entity that can act as both a user agent client and user agent server [6]. A user agent will act as either a UAC or as a UAS or as both in a SIP transaction.

User Agent Client (UAC): A user agent client is a logical entity that creates a new request, and then uses the client transaction state machinery to send it [6]. The role of UAC lasts only for the duration of that transaction. In other words, if a piece of

software initiates a request, it acts as a UAC for the duration of that transaction. If it receives a request later, it assumes the role of a user agent server for the processing of that transaction. The set of processing functions required of a UAC that reside above the transaction and transport layers is termed as UAC core. Note that SIP UACs may or may not interact directly with a human user.

User Agent Server (UAS): A user agent server is a logical entity that generates a response to a SIP request [6]. The response accepts, rejects, or redirects the request. This role lasts only for the duration of that transaction. In other words, if a piece of software responds to a request, it acts as a UAS for the duration of that transaction. If it generates a request later, it assumes the role of a user agent client for the processing of that transaction. The set of processing functions required at a UAS that resides above the transaction and transport layers is termed as UAS core.

The various components of VoIP communication using SIP signaling protocol can be categorized depending on the function carried out. Following are these various components (a software component or dedicated equipment) commonly used today.

Soft-phone: A soft-phone is a software application for making and receiving telephone calls using VoIP services.

VoIP phones: The special telephones with built-in VoIP technology for making and receiving calls over an IP network such as the Internet.

SIP Registrar and SIP location server: SIP registrar accepts REGISTER requests and places the information it receives in those requests into the location service for the domain it handles [6]. UAs register with the registrar from time to time to allow other UAs to locate them. A VoIP user is not bound to a host. So, there is a need for SIP or any other signaling protocol used with VoIP to provide the location service. The location service, as the name implies, allows a SIP user agent to locate another user agent. SIP provides location service through registration and location servers. Upon initialization and periodically, a SIP phone (end point) reports its location by registering with the registration server (registrar) of its domain. This information is then stored in the location server database. Note that other mechanisms can be employed along with the registrations to create the location data-base. Registrations are one way to create this information, but not the only way. Arbitrary mapping functions (user name or number to user location) can be configured at the discretion of the administrator. Note that a SIP user can register from multiple SIP phones. This allows a proxy to perform various types of searches to locate a SIP user. Similarly, more than one user can be registered on a single device at the same time. This allows multiple users to use SIP phone on a host.

SIP Proxy Server: SIP proxy is an intermediary entity that acts as both a UAS and a UAC for the purpose of making requests on behalf of other UACs [6]. A proxy server primarily plays the role of routing, which means its job is to ensure that a request is sent to another entity closer to the targeted user. Proxies are also useful for enforcing policy (for example, making sure a user is allowed to make a call). A proxy interprets, and, if necessary, rewrites specific parts of a request message before forwarding it.

SIP proxy server allows routing of incoming SIP requests. When it receives a SIP request, it contacts the location server to resolve the username into an address and then forwards the message. The proxy can be either stateless or stateful. A stateless proxy is a logical entity that does not maintain the client or server transaction state machines defined in SIP RFC when it processes requests. A stateless proxy forwards every request it receives downstream and every response it receives upstream. A stateful proxy is a logical entity that maintains the client and server transaction state machines defined in SIP RFC during the processing of a request, also known as a transaction stateful proxy.

The proxy servers can take flexible routing decisions. For example, it can route to voice server when a SIP phone signals it is busy. It can also do a parallel search, known as forking, by sending an INVITE to a number of locations at the same time. Proxy servers can also provide some mid-call features by remaining SIP messaging path between SIP end points. For this, the proxy will add to the INVITE a required routing header field known as Record-Route that contained a URI resolving to the hostname or IP address of the proxy. This information in Record-Route will be used by when SIP end point on the other end to send messages back through SIP proxy.

Redirect Server: A redirect server is a user agent server that generates responses to requests it receives, directing the client to contact an alternate set of URIs [6]. It generally happens when a recipient has moved from its original position either temporarily or permanently. SIP redirect servers resolve the username into an address by contacting the location server but they do not forward or proxy the incoming SIP

messages. Instead, they provide the address to the sender. The sender then can send SIP message directly using the resolved address of the destination that he got from the redirect server. With this approach, the advantage is that proxy server will not be a bottleneck in forwarding SIP messages. However, SIP end points now have to perform all the routing and they can become complex depending on how much flexible they are in taking routing decisions.

Note that the role of UAC and UAS, as well as proxy and redirect servers, is defined on a transaction-by-transaction basis. For example, the user agent initiating a call acts as a UAC when sending the initial INVITE request and as a UAS when receiving a BYE request from the calling agent. Similarly, the same software can act as a proxy server for one request and as a redirect server for the next request. Also, note that these servers are logical entities and implementations may combine them into a single application or run them on a single physical server.

SIP Proxy Operational Modes: A SIP proxy server can be classified as either an inbound proxy server or outbound proxy server or local proxy server depending on how many UACs are sending requests to that proxy server and how many UASs are receiving calls through that proxy server [5]. An inbound proxy server receives calls from many UACs but forwards them to one UAS (usually to local proxy server). An outbound proxy server receives calls from one UAC and sends them out to several UASs. A local proxy will receive calls from several UACs and sends them to several UASs.

Overview of SIP Operation

This section describes establishing and tearing down a VoIP session through SIP signaling using an example that was used in SIP RFC. The Figure 2 below shows a SIP trapezoid arrangement where two users, Alice and Bob, are in different domains [6]. The proxy servers act on behalf of Alice and Bob to facilitate the session establishment and tear-down.

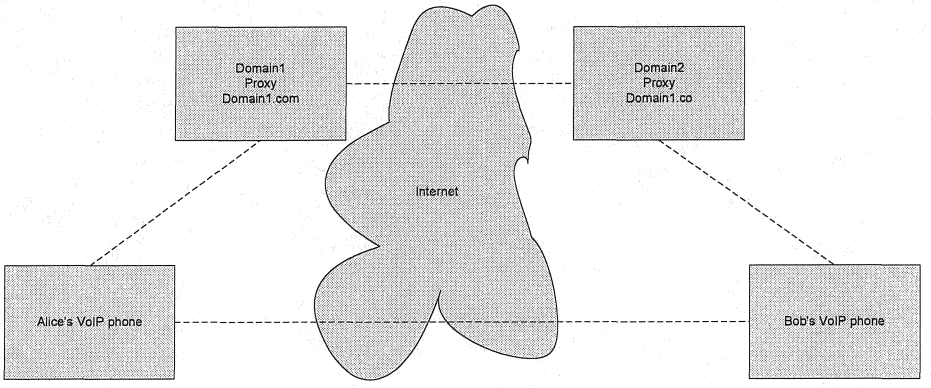

FIGURE 2. SIP trapezoid network.

Session Setup

In this example, Alice uses a soft-phone to call Bob on his SIP phone over the Internet. Alice calls Bob using his SIP identity, a type of Uniform Resource Identifier (URI) called a SIP URI. It has a similar form to an email address, typically containing a username and a host name. In this case, it is sip:bob@domain2.com, where domain2.com is the domain of Bob's SIP service provider. Alice has a SIP URI of sip:alice@domain1.com. SIP also provides a secure URI, called a SIPS URI. An

example would be sips:bob@domain2.com. A call made to a SIPS URI guarantees that secure, encrypted transport (namely TLS) is used to carry all SIP messages from the caller to the domain of the callee. From there, the request is sent securely to the callee, but with security mechanisms that depend on the policy of the domain of the callee.

The Figure 3 below shows a typical example of a SIP message exchange between Alice and Bob [6]. SIP is based on an HTTP-like request/response transaction model. Each transaction consists of a request that invokes a particular method, or function, on the server and at least one response. In this example, Alice initiates a call to Bob by sending an INVITE request addressed to Bob's SIP URI. The body of a SIP message contains a description of the session, encoded in some other protocol format like SDP. Since Alice does not know the location of Bob or SIP server in the domain2.com domain, the INVITE request will be forwarded to SIP proxy of Alice's domain, domain1.com. The address of the domain1.com SIP server could have been configured, or it could have been discovered by DHCP, for example.

The domain1 proxy receives the INVITE request and sends a 100 (Trying) response back to Alice's soft-phone. The response 100 (Trying) indicates that the INVITE has been received and that the proxy is working on her behalf to route the INVITE to the destination.

The domain1.com proxy server locates the proxy server at domain2.com, possibly by DNS and forwards the INVITE request there. Before forwarding the request, the

atlanta.com proxy server adds an additional Via header field value that contains its own address (the INVITE already contains Alice's address in the first Via).

The domain2.com proxy server receives the INVITE and responds with a response 100 (Trying) back to the domain1.com proxy server to indicate that it has received the INVITE and is processing the request.

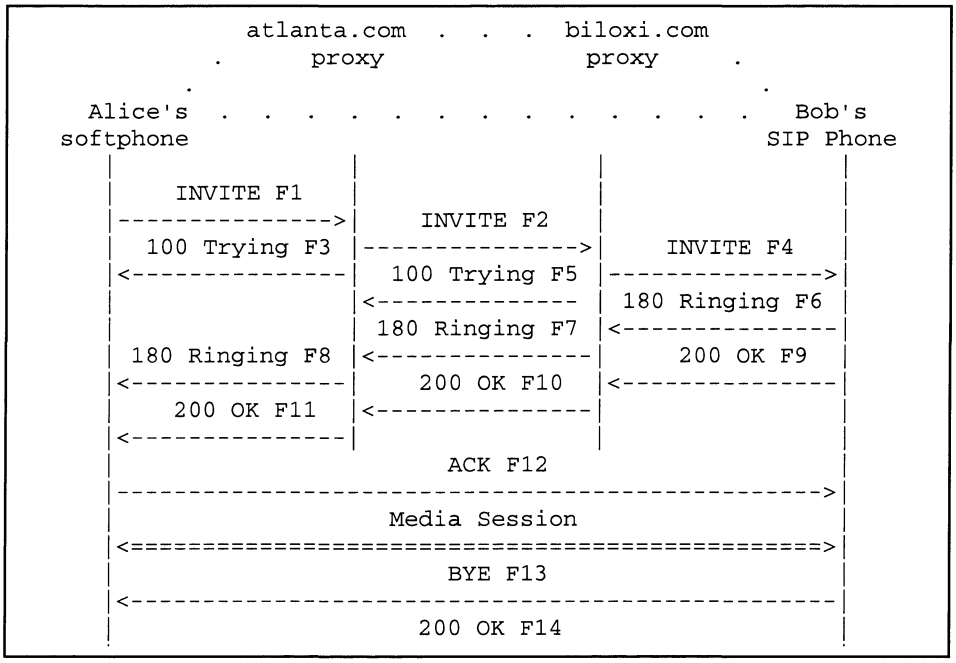

FIGURE 3. Typical SIP session setup and teardown [6].

The domain2.com proxy server consults the location server for Bob's IP address and forwards the INVITE to Bob's SIP phone. The domain2.com proxy server also adds another Via header field value with its own address to the INVITE before it is proxied.

Bob's SIP phone receives the INVITE and rings to alert Bob to the incoming call. Bob's SIP phone indicates this in a response 180 (Ringing), which is routed back through the two proxies in the reverse direction. Each proxy uses the Via header field to determine where to send the response and removes its own address from the top. As a result, although DNS and location service lookups were required to route the initial INVITE, the response 180 (Ringing) can be returned to the caller without lookups or without state being maintained in the proxies. This also has the desirable property that each proxy that sees the INVITE will also see all responses to the INVITE.

When Alice's softphone receives the response 180 (Ringing), it passes this information to Alice, perhaps using an audio ring-back tone or by displaying a message on Alice's screen.

In this example, Bob decides to answer the call. When he picks up the handset or accepts the call on his soft-phone, his SIP phone sends a 200 (OK) response. The 200 (OK) is routed back through the two proxies and is received by Alice's soft-phone, which then stops the ring-back tone and indicates that the call has been answered. The 200 (OK) contains a message body with the SDP media description of the type of session that Bob is willing to establish with Alice. As a result, there is a two-phase exchange of SDP messages: Alice sent one to Bob, and Bob sent one back to Alice. This two-phase exchange provides basic negotiation capabilities and is based on a simple offer/answer model of SDP exchange. If Bob did not wish to answer the call or was busy on another

call, an error response would have been sent instead of the 200 (OK), which would have resulted in no media session being established.

Finally, Alice's soft-phone sends an acknowledgement message, ACK, to Bob's SIP phone to confirm the reception of the final response (200 (OK)). In this example, the ACK is sent directly from Alice's soft-phone to Bob's SIP phone, bypassing the two proxies. This occurs because the endpoints have learned each other's address from the Contact header fields through the INVITE/200 (OK) exchange, which was not known when the initial INVITE was sent.

Alice and Bob's media session has now begun, and they send media packets using the format to which they agreed in the exchange of SDP. In general, the end-to-end media packets take a different path from SIP signaling messages.

Session Modification

During the session, either Alice or Bob may decide to change the characteristics of the media session. This is accomplished by sending a re-INVITE containing a new media description. This re-INVITE references the existing dialog so that the other party knows that it is to modify an existing session instead of establishing a new session. The other party sends a 200 (OK) to accept the change. The requestor responds to the 200 (OK) with an ACK. If the other party does not accept the change, he sends an error response such as 488 (Not Acceptable Here), which also receives an ACK. The failure of the re-INVITE does not cause the existing call to fail--the session continues using the previously negotiated characteristics.

Session Teardown

At the end of the call, Bob disconnects (hangs up) first and generates a BYE message. This BYE is routed directly to Alice's soft-phone, again bypassing the proxies. Alice confirms receipt of the BYE with a 200 (OK) response, which terminates the session and the BYE transaction.

Query Capabilities

SIP supports querying an end point's capabilities. The OPTIONS SIP method allows one user agent to query the capabilities (supported SIP methods, content types, CODECs, and extensions) of another user agent without ringing it. This allows a user agent to insert only the headers that the destination SIP phone supports

CHAPTER 3

SIP SECURITY THREATS

The flexibility and rich feature-set of SIP based IP telephony compared to traditional PSTN based phone comes with the additional security risks. SIP based IP telephony system is vulnerable to general Internet attacks, as well as attacks which are specific to SIP. As most of SIP development so far has focused on features and inter-operability, there exists ample opportunity to work on SIP security. In this section, let us discuss the various security attacks and threats applicable to SIP.

Network Security

In general, network security refers to providing secure communications over a public network such as Internet. The network security requires confidentiality, authentication, message integrity, and availability. The confidentiality allows only the sender and intended receiver can understand the messages being exchanged. The authentication allows the sender and receiver to confirm the identity of each other. The message integrity ensures that the messages are not altered (in transit, or afterwards) without detection. And the availability ensures that the services are accessible and available to users.

SIP Security Threats

As with any other network protocol, SIP is exposed to a wide range of security attacks. When deployed in a private network where network equipment and users are trustworthy and physical security is agreeably sufficient, SIP security may not be needed. However, since SIP can be deployed in an unreliable and untrustworthy environment like Internet, it is susceptible to various security attacks that include the common TCP/IP attacks. The various SIP security threats can be classified as external or internal. External threats happen when packets are traversing through third-party networks where as internal threats happen due to malicious users within the same network. SIP security threats can also be classified as given below based on what feature of network security is being attacked.

Confidentiality threats: These threats expose the content of the conversation or other data that is supposed to be confidential between the two SIP end points. Examples of these threats are sniffing and traffic analysis. In general, confidentiality is achieved through the encryption techniques.

Integrity threats: These threats impact the ability to trust the identity of the caller, the integrity of the messages, or the identity of the recipient. Examples of these threats are registration hijacking, message tampering, and spoofing.

Availability threats: These threats attempt to jeopardize the ability to make or receive a call. Examples of these threats are message fabrication, replay, and various DoS attacks.

The various security threats we discuss next are registration hijacking, message modification, impersonating a server, Denial of Service attacks, tearing down sessions, and replay attacks [7]. Understanding these threats helps us to understand the security mechanisms employed in SIP and to evaluate their impact on performance.

Registration Hijacking

In SIP registration, the registrar is not obliged to challenge the UA that has sent the registration request for authentication (In RFC3261, registrars are only RECOMMENDED to challenge registration requests). The absence of authentication or weak authentication makes the registration hijacking possible where an attacker can hijack the registration requests from a valid UA by impersonating that UA to a registrar. The attacker can then direct all requests (for example, incoming calls) for the affected UA (the UA being impersonated by the attacker) to his or her device by de-registering all existing contacts associated with the affected UA's URI and registering his/her own device as the contact address for the affected UA's URI.

The registration hijacking can result in Denial of Service to a legitimate UA, eavesdropping by intercepting to listening to voice calls, attacker tricking the caller into leaving a message, Man-In-The-Middle (MITL) attack where attacker transparently sits between the calling and called UAs and collects and/or modifies both the signaling and media, or toll fraud by redirecting the incoming call to a media gateway.

This registration hijacking relies on absence or weak form of authentication for registration requests. A weak authentication like simple username/password is not enough as it can be easily broken using dictionary style attack. This demonstrates the need for strong authentication of SIP requests. However, implementing strong authentication is particularly difficult in SIP as SIP messages may traverse through a number of SIP elements that may legitimately modify the messages.

Impersonating a Proxy Server

In this threat, an attacker could impersonate the proxy server, and trick one of SIP users or proxy servers into communicating with him [7]. This occurs again due to lack of authentication of proxy servers. An attacker can get into the signaling stream through either Domain Naming Service (DNS) spoofing or Address Resolution Protocol (ARP) cache spoofing or by changing the proxy address for a SIP phone. This impersonation of a SIP server allows various security attacks like DoS, eavesdropping, and toll fraud.

If the attacker is successful in impersonating a proxy server, he can get a complete control of a call. All outbound calls from a domain can be intercepted, blocked, and manipulated if the proxy server of that domain is impersonated using DNS spoofing. Similarly, the calls originating from a UA can be intercepted, blocked, and manipulated by the attacker when ARP cache spoofing is used against a network switch to trick a UA into communicating with it [7]. Prevention of this threat requires a means by which UAs can authenticate the servers to whom they send requests.

Tampering with Message Bodies

In this attack, the attacker intercepts and modifies SIP messages. This message tampering is possible as SIP messages have no built-in mandatory means to insure integrity. This attack usually occurs because of a compromised proxy server which is trusted by the UAs in the domain of the proxy server. This attack can also occur through registration hijacking, proxy impersonation, or through any compromised SIP element which is trusted to process SIP messages [7].

For a specific example of this attack, consider a UA that is communicating session encryption keys for a media session using SIP messages. The UA may trust proxy in delivering SIP messages but not necessarily trust proxy for insuring integrity. In other words, the proxy administrators may be able to decrypt SIP messages and find out the encryption keys being exchanged. If an attacker can gain access to proxy, then he can gain access to all the information in SIP messages and this allows attacker to either tamper the message bodies or play the man in middle attack.

To achieve protection against message tampering, SIP message bodies and some header fields need to be secured from end-to-end through encryption services. These end-to-end message integrity services should work together with and be independent of the means used to secure interactions with intermediaries such as proxy servers.

Denial of Service

DoS is a common attack that targets one or more SIP elements to make one or more SIP services unavailable, usually by directing a high volume of traffic towards the

service thereby denying it to legitimate clients. The DoS attack can also be launched using multiple network hosts to flood a SIP element with a large amount of network traffic. These DoS attacks are called as Distributed DoS attacks. DoS is a major issue in SIP systems, as some kind of trust is involved in any deployment and that the DoS can be launched in a variety of ways.

DoS due to high volume of traffic: In this DoS attack, SIP servers, voice gateway devices, firewalls, and DNS lookup servers will be bombarded with high volume of traffic; thereby make them unavailable for legitimate users. By gaining access to SIP element in a network, it can be used as a DoS launching point.

DoS due to malformed SIP messages: In this DoS attack, malformed SIP messages will be sent to manipulating SIP states and cause DoS.

DoS due to unauthenticated register requests: In this attack, a user is prevented from receiving further calls by deregistering that user with the registrar. The DoS is also caused by sending huge numbers of registration requests and thereby bringing down the registrar by depleting the memory resources of registrar. The DoS attack can also be launched by registering a huge number of bindings for the same host and thereby amplifying SIP traffic.

DoS due to spoofed SIP messages: By spoofing SIP messages, the DoS attack can be launched in few ways. One way is through reflection where the attacker sends a

spoofed request with the spoofed IP address of target being attacked to many SIP elements, thereby generating a huge amount of traffic aimed back at target.

Cancel/Bye DoS attack: This DoS attack is launched by tearing down the sessions using the BYE request as the BYE requests are not authenticated and are not acknowledged.

Re-Invite DoS attack: This DoS attack is launched using re-INVITE messages that are used to change session parameters. One way is to redirect the media to broadcast address which generates a huge amount of traffic. Another way is to redirect the media sessions to a proxy or a gateway to bring it down.

DoS through amplification: These DoS attacks use amplification by using the forking feature of proxy servers. One way is to put the victim's IP address into a spoofed Router header request, and send it to forking proxies, who will greatly amplify the number of messages returned to the victim. The Record-Route header could also be used for amplification by sending a large number of requests to many SIP users with the victim's IP address in the Record Route header and thereby making the victim to receive a large amount of traffic.

Session Teardown Attack

In this attack, the attacker tears down the sessions by sending spoofed BYE messages. This spoofing of BYE messages is possible by capturing some initial messages in a dialog. It is also possible to tear down the sessions by flooding the firewall

with BYE messages and thereby causing the firewall to tear down the UDP/TCP ports being used for legitimate calls. The session tear-down results in DoS by abrupt termination of existing calls or calls being setup. Preventing this session teardown attacks requires authentication of BYE request.

Replay Attack

In this attack, the attacker intercepts SIP messages and retransmits them as is so that the victim will have to reprocess these messages. Preventing this kind of attacks require encryption techniques and use of nonce.

CHAPTER 4

SIP SECURITY MECHANISMS

In this section, the various security mechanisms employed in SIP protocol today and their limitations are discussed. Also described are SIP security implementation requirements and issues.

SIP Security Challenges

The major difficulty with employing SIP security solutions is to make them work, without extensive co-ordination, in a wide variety of environments and usages. The security in SIP is very challenging as SIP uses many intermediaries like proxy servers, registrar servers, and redirect servers, SIP has many elements and supports multi-faceted trust relationships between them, SIP is deployed in a wide variety of environments, and SIP requires user-to-user operation.

SIP Security Mechanisms

SIP employs encryption mechanisms to provide confidentiality and to prevent malicious users from modifying the messages. However, the end-to-end encryption of SIP messages is not possible as SIP intermediaries need access to some information in SIP headers. For this reason, SIP supports both end-to-end encryption and hop-to-hop encryption techniques.

The end-to-end encryption is supported using S/MIME mechanisms described later in this section and is used for all the information that is not required to be accessed by the intermediaries. The hop-by-hop encryption is supported using IPSec or TLS and is used for preserving the confidentiality of the information that needs to be seen by the intermediaries. The encryption algorithms commonly used with SIP are the Data Encryption Standard (DES) and Advanced Encryption Standard (AES). Note that although encryption provides confidentiality of SIP messages, it can be detrimental to QoS.

SIP employs a cryptographic authentication mechanism for providing the message integrity and verifying the autheticity of the senders of SIP messages. It is important to note that the authentication allows a UA to verify the identity of another UA but does not provide message integrity and hence the need for cryptographic authentication which combines the encryption and authentication techniques in order to provide authentication, confidentiality, message integrity, and protection against replay attacks. SIP authentication mechanism is based on the HTTP Digest authentication. SIP also supports a scheme called SIPS URI which allows indicating SIP intermediaries of the need to forward SIP messages using TLS security.

Authentication

SIP provides an authentication mechanism, which is based on the HTTP Digest authentication defined in RFC2617, for a UAC to identify itself to the UAS in another SIP element (proxy or registrar or another user) [6] [8]. The authentication is a challenge

based mechanism where SIP element receiving a message can challenge the sender for credentials. SIP element receiving the challenge through a response for its request should provide its credentials to assure its identity. SIP element, which receives the credentials from the sender of a SIP message in response to the challenge it sent, will verify the identity of the sender and ascertain whether the sender is authorized to make the request that it sent.

The replay attacks are prevented by using a nonce in the authentication mechanism. Nonce stands for *Number used once* and is a unique number (often generated as a random number or a pseudo-random number) that will not be used again with in a pre-defined time so that the receiver can use it to identify the replay attacks.

SIP authentication provides the message authentication and replay attack protection but not message integrity and confidentiality. SIP authentication should be used in conjunction with cryptographic techniques to prevent active attackers from modifying SIP requests and responses.

User to User Authentication: A user can challenge another user for determining authenticity of a received message. This is accomplished through the use of the 401 and 407 response codes as well as header fields for carrying challenges and credentials. The WWW-Authenticate header field is used by an UAS to challenge an UAC for credentials. The UAC authenticating will use the Authorization header to supply the credentials.

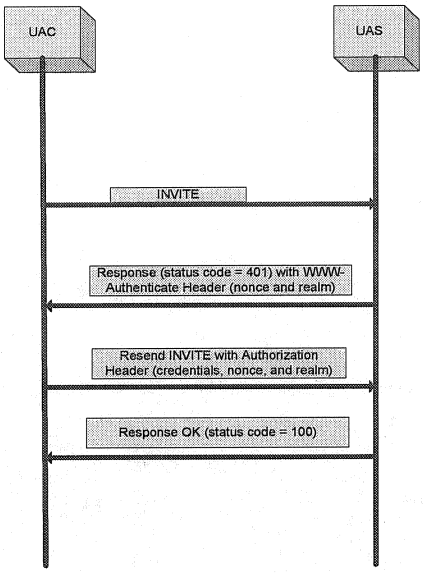

FIGURE 4. User to user authentication

As shown in above Figure, a UAC sends a request to a UAS. The UAS challenges the UAC by sending a response with 401 status code (Unauthorized) and WWW-Authenticate header in the response. The UAS indicates the authentication scheme(s) and parameters applicable to the realm to the UAC through this WWW-Authenticate header. Upon receiving the response with Unauthorized (401) status code and WWW-Authenticate header, the UAC will locate its credentials and resend the request along with proper credentials using the Authorization header in the request. The Authorization header will also consist of other parameters required in support of authentication and replay protection.

UAS will use the supplied credentials to verify the identity of the sender and whether the sender is authorized for the request made. If the sender is authenticated successfully, it will proceed with processing the request. Note that a UAC may send a request with Authorization header without being challenged to help reducing the overhead.

Proxy-to-User Authentication: A SIP proxy or Registrar or Redirect server can authenticate a SIP UAC in the same way as described above except now the headers and response codes are different.

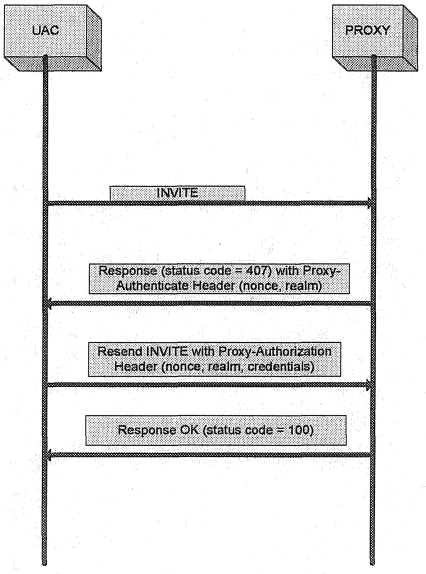

FIGURE 5. Proxy to user authentication

As shown in Figure above, a UAC sends a request to a Proxy. The Proxy challenges the UAC by sending a response with 407 status code (Proxy Authorization Required) and Proxy-Authenticate header in the response. Upon receiving the response with Proxy Authorization Required (407) status code and Proxy-Authenticate header, the UAC will locate its credentials and resend the request along with proper credentials for the realm of the proxy using the Proxy-Authorization header field in the request. Proxy will use the supplied credentials to verify the identity of the sender. If the sender is authenticated successfully, proxy will forward the request.

As proxies can fork requests, it is possible for a response to have multiple challenges (multiple Proxy-Authenticate and/or WWW-Authenticate messages). These different challenges may be for different realms or for the same realm. A proxy authorization header field is differentiated from that of another proxy using the realm parameter. The same credentials are used for the challenges with the same realm and the credentials will be different for the challenges with different realms.

Proxy-to-Proxy Authentication: SIP does not support the proxy-to-proxy authentication as it does not define a mechanism for a proxy to identify itself to another proxy. If needed, this proxy-to-proxy authentication will have to be achieved with other means like IPSec and/or TLS that are supported by SIP.

S/MIME

The entire SIP message cannot be encrypted end-to-end as SIP intermediaries need to view certain SIP headers and may modify or add some SIP headers. As a result,

the end-to-end encryption techniques can only be used with SIP message bodies. SIP message bodies use the MIME format [8]. SIP supports S/MIME (Secure MIME) for securing the contents of MIME bodies within SIP messages. S/MIME encrypts SIP bodies with the public key of the receiver and signs it with the private key of sender. The receiver will use the public key of sender to verify the signature and private key of the receiver to decrypt SIP MIME bodies. This means, senders must know the public key of recipients and receivers must know the public key of sender. So, S/MIME requires exchanging or sharing the keys and is achieved through the use of certificates.

S/MIME uses certificates that assert the association of the holder with the end-user address that is formed by the concatenation of the *userinfo*, @, and *domainname* portions of a SIP or SIPS URI. These certificates are associated with the keys that are used to sign and encrypt SIP messages. The certificates may be issued by public certificate authorities or self-generated or pre-configured. The pre-configuration is not scalable and is used in deployments in which a previous trust relationship exists between all SIP entities. The self-generated certificates provide message integrity to some extent but may not provide the authentication. When cryptographic keys are exchanged with the use of self-signed certificates or certificates signed by an obscure authority, SIP is vulnerable to security attacks like replay attacks, impersonation attacks, and man-in-the-middle attacks. The use of public certificate authorities is scalable and widely used but one main issue is that there is virtually no consolidated public certificate authority.

SIP also specifies the use of SIP message tunneling for providing protection to SIP header fields. In this case, the entire SIP message is encrypted and embedded within another SIP message for tunneling. This makes it difficult to tamper with SIP headers and easy to detect any tampering of SIP headers in the tunneled SIP message. However, this creates additional overhead. When tunneling is used, it is recommended to use TCP rather than UDP to avoid problems due to UDP fragmentation of larger messages.

Transport Layer Security (TLS)

TLS provides the security at transport layer. TLS encrypts signaling traffic, guaranteeing message confidentiality and integrity. TLS protects SIP signaling from replay attacks, man-in-the-middle attacks, eavesdropping, or unauthorized access by providing integrated key-management with mutual authentication and secure key distribution. Although TLS can be used for transport layer security when using any connection-oriented protocol, SIP specifies the use of TLS with TCP as TCP is the widely used transport protocol. The TLS protocol version 1.0 is defined in RFC2246.

TLS performs a handshake process before encrypted data is transported. During the handshake, the TLS server and client perform peer authentication and exchange and/or negotiation of various parameters needed for selecting the cipher-suite and session keys [6] [8] [9].

The Figure below shows the TLS call flow diagram [6][9]. An outbound SIP request is sent from the client and the transport indicates the use of TLS. A TLS connection is opened towards the server on a specific IP address and port. All the

algorithms used in a TLS session, including those for key exchange, bulk data encryption and message digest, are specified by a cipher suite. The client sends a 'Client Hello' message specifying the TLS version and a list of suggested cipher suites it supports.

The server responds with several messages. It first sends a 'Server Hello' message with the TLS version and a chosen cipher suite. It then presents a certificate or certificate chain to the client using the 'Certificate' message. Usually only the client authenticates the server. But SIP has support for mutual authentication. So when a server is configured for mutual authentication, it also requests a certificate from the client using the 'Certificate Request' message. The server also sends the client a 'Server Key Exchange' message when the 'Server Certificate' message does not contain enough data to allow the client to exchange a premaster secret. If sent, this message will immediately follow the 'Certificate' message from the server (or the 'Server Hello' message for the case of anonymous negotiation). The server then sends a 'Server Hello Done' message to tell the client that it has finished the initial negotiations.

If the server requested a certificate, the client sends the certificate or certificate chain using the 'Certificate' message. The client then sends a Client Key Exchange message which may contain a Premaster Secret, Public Key, or nothing depending on the cipher suite chosen. This Premaster Secret is encrypted using the public key of the server certificate.

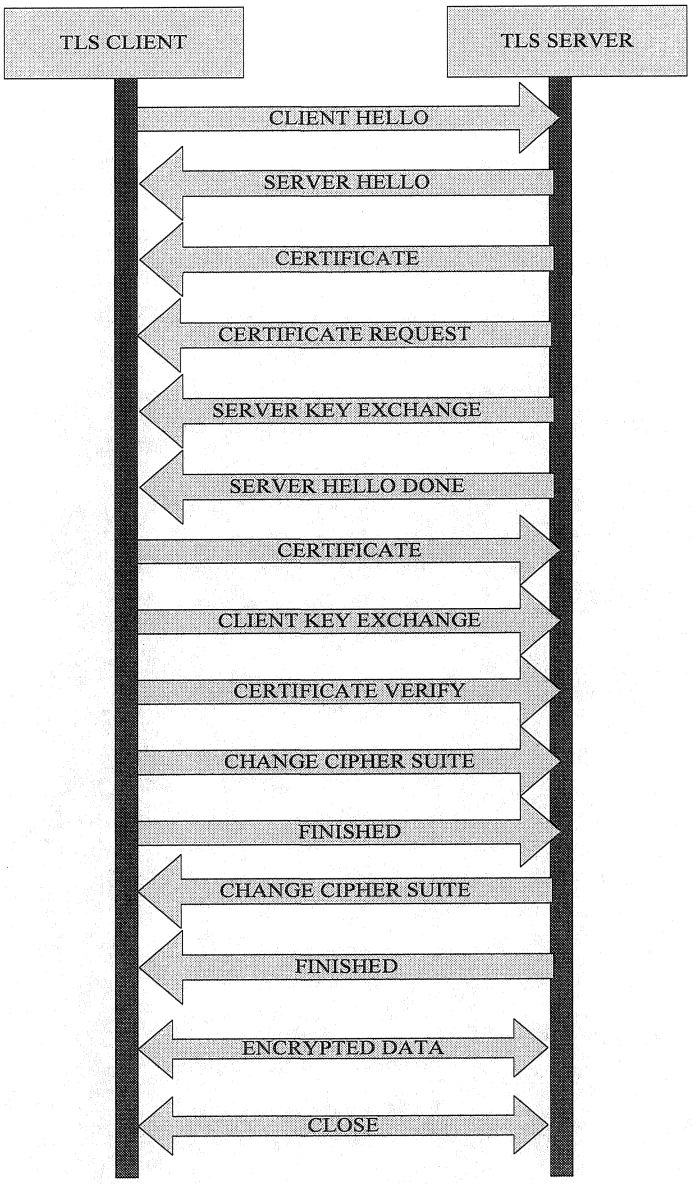

FIGURE 6. TLS call flow diagram.

If the client is being authenticated by the server, the client will send the 'Certificate Verify' message which is a signature over the previous handshake messages

using the client's certificate's private key. The server will verify this signature using the client's certificate's public key and thus can be assured of that certificate received is authenticated. This 'Certificate Verify' is sent only when client is presenting its certificate upon server's request.

The client sends a 'Change Cipher Spec' message to tell the server that all future messages will be signed and encrypted as negotiated. And then it sends an authenticated and encrypted 'Finished' message indicating server that client is done with the handshake phase.

After receiving the 'Finished' message from client, the server will decrypt and verify the hash and MAC. After successful verification, the server will send a 'Change Cipher Spec' message to tell the client that all future messages will be signed and encrypted as negotiated. The server then sends the authenticated and encrypted 'Finished' message which client will verify before sending the encrypted data.

Encrypted data: The client and the server communicate using the symmetric encryption algorithm and the cryptographic hash function negotiated, and the shared secret key exchanged. At the end of the connection, each side will send a 'Close Notify' message to inform the peer that the connection is closed.

Session Reuse: The session reuse of TLS allows previously negotiated set of premaster secret and cipher suite to be reused. When a new session is established, the server stores the session information for reuse. Also, the session id used by server is

conveyed to the client in the 'Server Hello' message. So, when a client wants to establish a session again, then it can use the previous session-id that was conveyed by the server. If session reuse is enabled, then server will first check if there was an established session with the given session id in the 'Client Hello' message. If the server finds that the session id requested was previously used, then it can agree to session reuse by specifying the same session id in the 'Server Hello' message. When the client realizes that server has agreed to session reuse, it will proceed directly to the 'Change Cipher Spec' and 'Finished' messages avoiding the need for re-computation of premaster secret and renegotiation of cipher suite.

SIPS URI scheme: SIPS URI scheme is specified in RFC 3261. It allows SIP elements to specify that end to end security is needed. The syntax of SIPS URI is same as that of SIP URI but begins with *sips*. When used as the Request-URI of a request, SIPS scheme signifies that each hop over which the request is forwarded, until the request reaches SIP entity responsible for the domain portion of the Request-URI, must be secured with TLS [6]. Once it reaches the domain in question it is handled in accordance with local security and routing policy. Note that in SIPS URI scheme, transport is independent of TLS, and thus "sips:alice@atlanta.com;transport=tcp" and "sips:alice@atlanta.com;transport=sctp" are both valid.

IPSec

IPSec is a set of network-layer protocol tools, defined in RFCs 2401-2411 and 2451, that collectively can be used as a secure network protocol. IPSec provides

authentication and encryption services at the network layer [10]. For this, it uses two protocols: AH (Authentication Header) and ESP (Encapsulated Security Payload). The AH is used, as the name suggests, to authenticate the packets, and optionally anti-replay protection. It ensures the sender's authentication but does not provide confidentiality of data. The ESP is used to ensure that the data transmitted between the two hosts securely, optionally with authentication and integrity checking. An IPSec implementation can use either or both of these protocols.

Both ESP and AH use security associations (SAs). A Security Association (SA) is a relationship between two or more entities that describes how the entities will use security services to communicate securely. The security association is unidirectional. It is uniquely identified by a randomly chosen unique number called the security parameter index (SPI), IPSec protocol (AH or ESP), and the destination IP address. When a system sends a packet that requires IPSec protection, it looks up the security association in its database, applies the specified processing, and then inserts the SPI from the security association into the IPSec header. When the IPSec peer receives the packet, it looks up the security association in its database by destination address and SPI and then processes the packet as required. In summary, the security association is simply a statement of the negotiated security policy between two devices.

AH Protocol Fields: The next header field specifies the next protocol. The length field specifies length of the AH header. The SPI field is an index used in combination with destination address to identify the correct security association for the

communication. The sequence number field is a 32-bit number to identify an IP packet and provide replay protection. The authentication data field contains the integrity check value (ICV), which is calculated over the IP header, the AH header, and the IP payload. The receiver calculates the ICV value and verifies it against the ICV in the AH header.

ESP Protocol Fields: The ESP protocol uses an ESP header, an ESP trailer, and an ESP authentication trailer. The ESP header includes Security Parameters Index (SPI) and sequence number. The ESP trailer contains the padding and next header fields. The padding is the bytes added (up to 255 bytes) to align encrypted payload as needed by the encryption algorithm. The next header field specifies the next protocol. The ESP authentication trailer contains the authentication data which is nothing but the integrity check value (ICV), calculated over the ESP header, the payload data, and the ESP trailer.

IPSec Modes: The IPSec supports two modes of operation: transport mode and tunnel mode. In transport mode, only the IP payload is encrypted and/or authenticated. In tunnel mode, the entire IP datagram is encrypted and/or authenticated by encapsulating it with another IP header. The transport mode is used mostly for host-to-host communications whereas the tunnel mode is used for network-to-network communications as in VPNs and host-to-network communications.

AH Protocol – Transport Mode: In this mode, the AH header is inserted between the IP header and the IP payload as shown in below Figure. The AH protocol signs the entire IP datagram except for the mutable fields in the IP header.

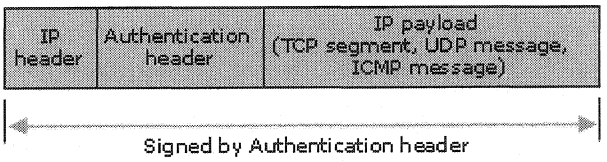

FIGURE 7. AH protocol--transport mode [10].

ESP Protocol –Transport Mode: The ESP protocol uses an ESP header, an ESP trailer, and an ESP authentication trailer as shown in below Figure. The ESP header is placed before the IP payload, the ESP trailer is placed after the IP payload, and the ESP authentication header is placed after the ESP trailer. The ESP encrypts the IP payload and ESP trailer and signs the ESP header, IP payload, and ESP trailer using the authentication data in ESP authentication trailer.

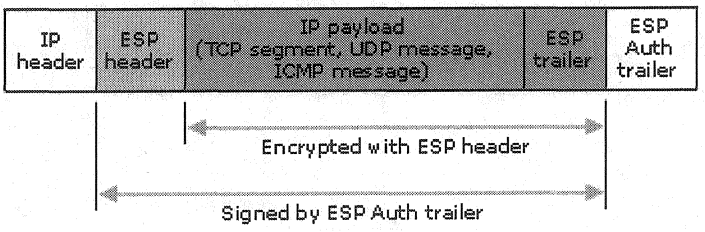

FIGURE 8. ESP protocol--transport mode [10]

AH Protocol –Tunnel Mode: In the tunnel mode, the IP datagram is encapsulated by the authentication header and an additional IP header as shown in below Figure. The

entire IP datagram including the additional IP header is signed except the mutable fields of the additional IP header.

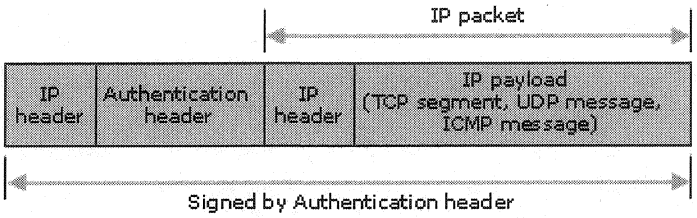

FIGURE 9. AH protocol--tunnel mode [10].

ESP Protocol –Tunnel Mode: The original IP data gram is added with ESP header, ESP trailer, and ESP authentication header and encapsulated within another IP header. The original IP datagram and ESP trailer will be encrypted with the ESP header for confidentiality. The ESP authentication trailer provides integrity for the ESP header, original IP datagram and the ESP trailer as shown in below Figure.

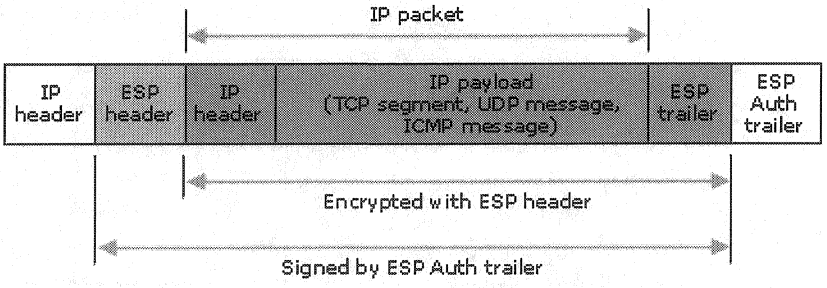

FIGURE 10. ESP protocol--tunnel mode [10]

Key Exchange: To avoid issues with manual keying and pre-shared keys like symmetric key problem, scalability, and keeping keys secret, the IPSec uses Internet Key Exchange (IKE) to automatically exchange randomly generated keys which are transmitted using asymmetric encryption technology, according to negotiated algorithm details. The IKE negotiates the connection parameters, which includes what type of connection, what encryption algorithms to use, and what keys are used.

Suitable Deployments: As IPSec is decoupled from SIP protocol and provides security at network layer, it is most suited for deployments where adding security to SIP would be arduous. The IPSec can be used on a hop-by-hop basis and hence is suited for deployment between two hosts or two administrative domains that have a trust relationship. The IPSec is also commonly used with UAs that have an existing trust relation-ship with their immediate proxy server.

Limitations of Security Mechanisms in SIP

The proxies need visibility into header fields and some features of messages. Also, NAT must be able to decrypt/re-encrypt SIP messages. This means encryption of full messages cannot be done end-to-end. The digest based authentication offers protection only for some parameters. The S/MIME lacks public key exchange infrastructure and the key exchange mechanism in SIP is susceptible to man-in-the-middle attack [4]. Also, S/MIME can result in very large messages [4]. TLS is not supported over connection less protocols like UDP [4]. Also TLS requires maintenance

of many simultaneous long-lived connections [4]. Another major problem for use of TLS is that there is no guarantee that all SIP elements in the path support TLS. As well, the handshake can be costly because of PKI based authentication and key calculation for each TLS session. End-to-end IPSec deployment can be very challenging in a typical VoIP environment where end points are dynamic. Not suitable for protecting VoIP and unified communications data from end to end. SIP proxies and hops along the way will not be able to decrypt or modify the information in SIP packets.

CHAPTER 5

PERFORMANCE EVALUATION TEST SETUP AND TOOLS

SIP supports several distinct security mechanisms as described in the previous section. Choosing and deploying the security solutions in a SIP based VoIP network requires not only the extensive knowledge of how various security mechanisms work and what their limitations are, but also the impact of these security mechanisms on SIP servers. This chapter describes the setup, tools, and methodology used for evaluating the performance of SIP proxy, and registrar servers which are the bottleneck for performance compared to SIP user agents.

Experimental Setup

The performance impact of security mechanisms under consideration are evaluated using the real test bed implementation shown in below Figure. The performance (either CPS or RPS or CPU utilization for a given load) of SIP server under test is measured by running tests with and without employing the security mechanism whose performance impact is being evaluated.

As can be seen in the below Figure, the test bed comprises of server under test connected to various load generators and call handlers. The load generators act as the UAC agents and the call-handlers act as the UAS agents. The UACs make calls to UASs via one or more SIP proxy servers.

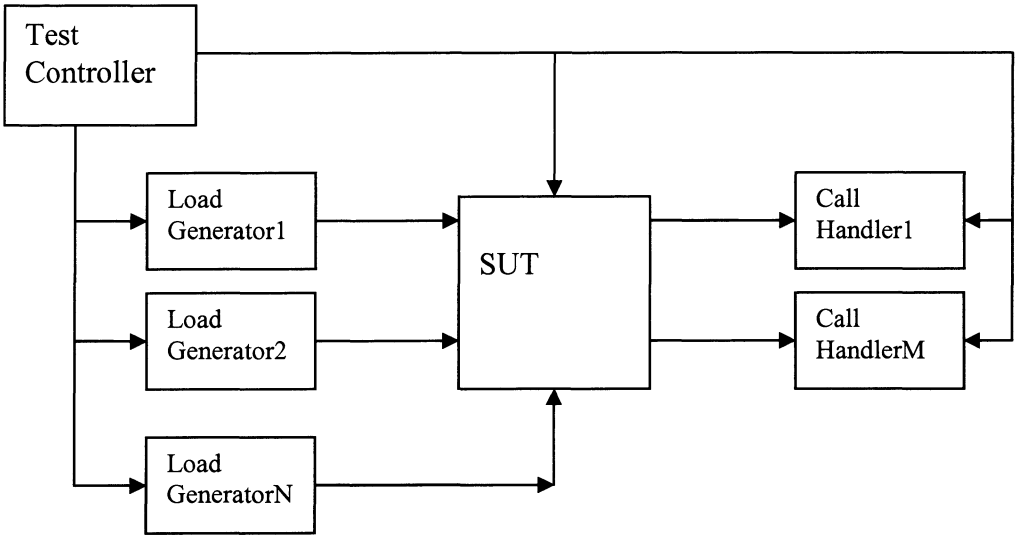

FIGURE 11. Experimental setup.

Server under Test (SUT): The SUT consists of the hardware and software required to support SIP servers (SIP proxy, redirect, and registration servers) whose performance is to be measured. The OpenSIPS server is used as the SUT in our performance evaluation setup.

Load Generators: These are SIP UACs which generate the workload required for performance evaluation. One or more instances of load generators will be used depending on the required work load. Also, the load generators have configuration options for various parameters like transport protocol, type of SIP messages, enabling or disabling SIP advanced features, the number of requests, inter arrival time for requests,

and so on. These load generators are implemented using the open source SIPp [11] testing tool by developing the UAC XML scenarios as required. SIPp is described in the next section.

Call Handlers: These are SIP UASs which handle the calls by responding to the requests from load generators within stipulated times [12]. These call handlers are also implemented using the open source SIPp testing tool by developing the UAS XML scenarios as appropriate for the test.

Test Controller: The test controller configures the load generators, SUTs, and call-handlers, and starts the tests for evaluating performance. It also collects the necessary statistics of performance metrics for reporting.

Performance Metrics

Registrations per second (RPS): Registrations per second is the average number of successful registrations per second during the measurement interval [12].

Calls per second (CPS): Calls per second is defined as the average number of calls per second completed with a 2xx or 4xx response during a measurement interval [12]. For the proxy server performance evaluation test case, a single call includes both the INVITE and corresponding BYE transaction.

CPU Utilization: The CPU utilization of SIP servers with different test scenarios at various work-loads. The profiling information is also used to identify the CPU utilization by different software components.

Test Methodology

The performance evaluation is performed by stressing the SUT using many requests from the load generators. A number of tests with different load levels and different measurement intervals will be used to get the results as accurate as possible.

Also, as the aim is to evaluate and compare the performance of a SIP server with and without a given security mechanism, all other performance bottlenecks like network throughput bottlenecks, network loading, path MTU, etc should be avoided. Also for, the same exact setup will be used for both cases of SIP with and without a security mechanism whose impact on performance is being evaluated.

The RPS and CPS values are determined by taking an average over the results from tests run for different measurement intervals. In each test run, the RPS and CPS value is determined as the highest sustained value when the load on the server is increased until the transaction failure reaches a value of ~4%. The CPU utilization is also measured by taking average with the tests running for long enough time (around 5-10 minutes) to minimize the measurement error.

The load generators will not wait for the response for a request sent before sending the more requests. This way, the transaction round trip time will not be the bottleneck in performance measurements.

SIPp Overview

SIPp is a free Open Source test tool/traffic generator for SIP protocol [11]. SIPp includes few basic user agent scenarios (UAC and UAS scenarios) to be able to establish and release multiple calls by generating the INVITE and BYE messages and their responses. SIPp supports the capability to read custom XML scenario files for generating and handling SIP user agent scenarios. This allows us to generate the complex SIP flows quickly. SIPp also supports collecting various statistics about running tests, display of the collected statistics dynamically, and periodic dump of statistics. SIPp also allows using different transports like UDP, TCP, and TLS. The call rates can also be adjusted dynamically. SIPp also supports SIP authentication, conditional scenarios, UDP retransmissions, error robustness (call timeout, protocol defense), call specific variable, POSIX regular expressions to extract and re-inject any protocol fields, custom actions (log, system command exec, call stop) on message receive, and field injection from external CSV file to emulate live users, etc. SIPp also supports media traffic through RTP echo and RTP/pcap replay. Media can be audio or audio and video.

SIPp is used in our experimental setup for implementing the load generators and call handlers. This allows us to emulate thousands of user agents calling SIP system under test and thus facilitates us to evaluate the impact on SIP system performance due to the overhead incurred by SIP security features.

SIPp Embedded Scenarios

SIPp supports the various embedded scenarios: UAC, UAC with media, UAS, regexp, Branch, UAC Out-of-call Messages, and 3PCC. The most commonly used scenarios are described below.

UAC Scenario: This is the most common scenario of establishing and terminating a call. SIPp UAC will send an INVITE request, wait for the ACK, pauses little bit after the ACK, and terminates the call by sending BYE request. This scenario is shown in below Figure.

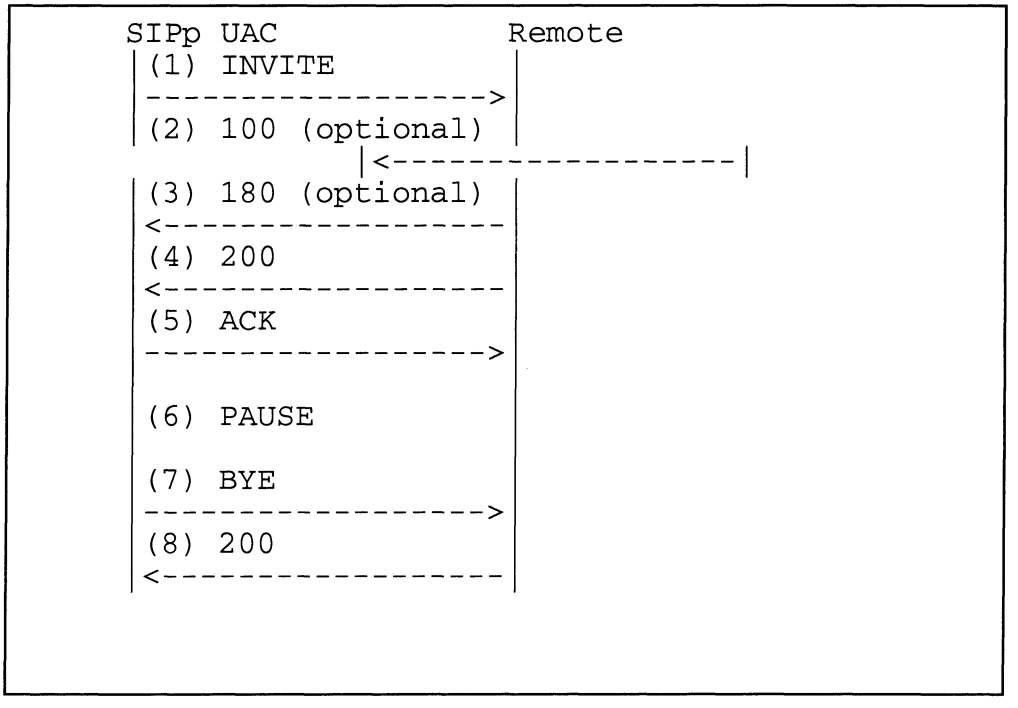

FIGURE 12. SIPp embedded UAC scenario.

UAC Scenario with Media: This scenario is similar to UAC scenario except that there is media traffic using RTP protocol after establishing a call. This scenario is illustrated in below Figure.

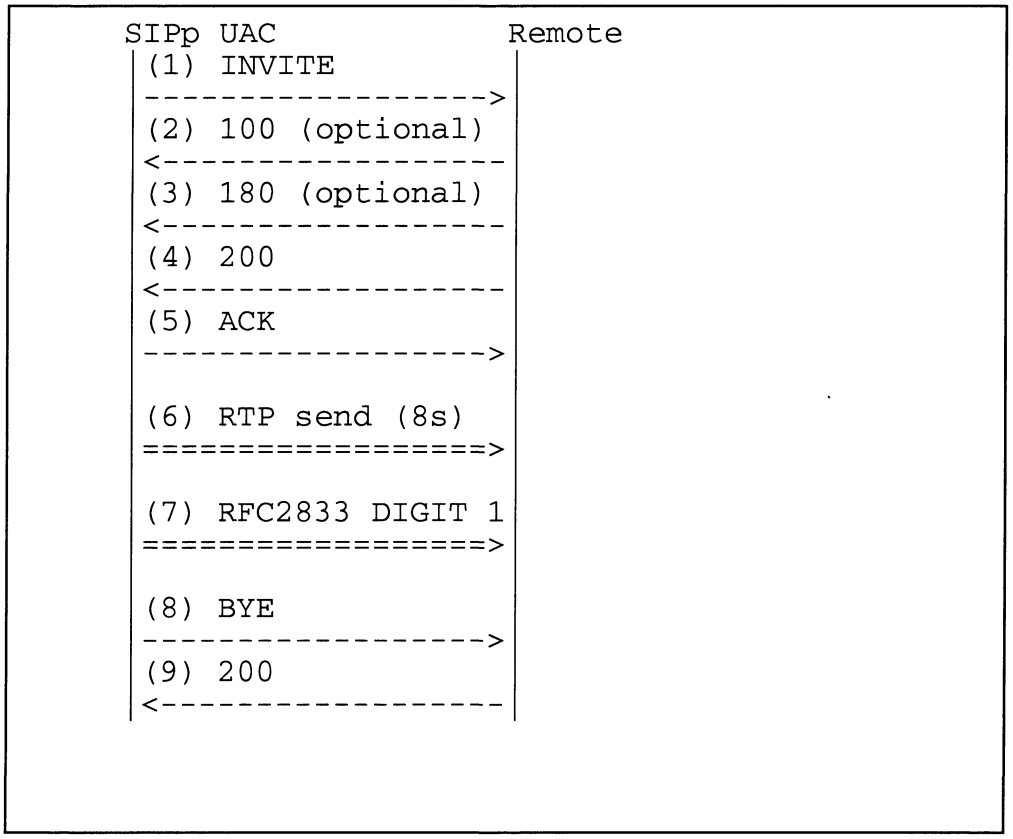

FIGURE 13. SIPp embedded UAC with media scenario.

UAS Scenario: This is the server scenario for establishing and terminating a call. As shown in the below Figure, the UAS will respond with 180 and 200 messages

for an INVITE request and with a 200 message for a BYE request. This scenario can be used when benchmarking SIPp UACs.

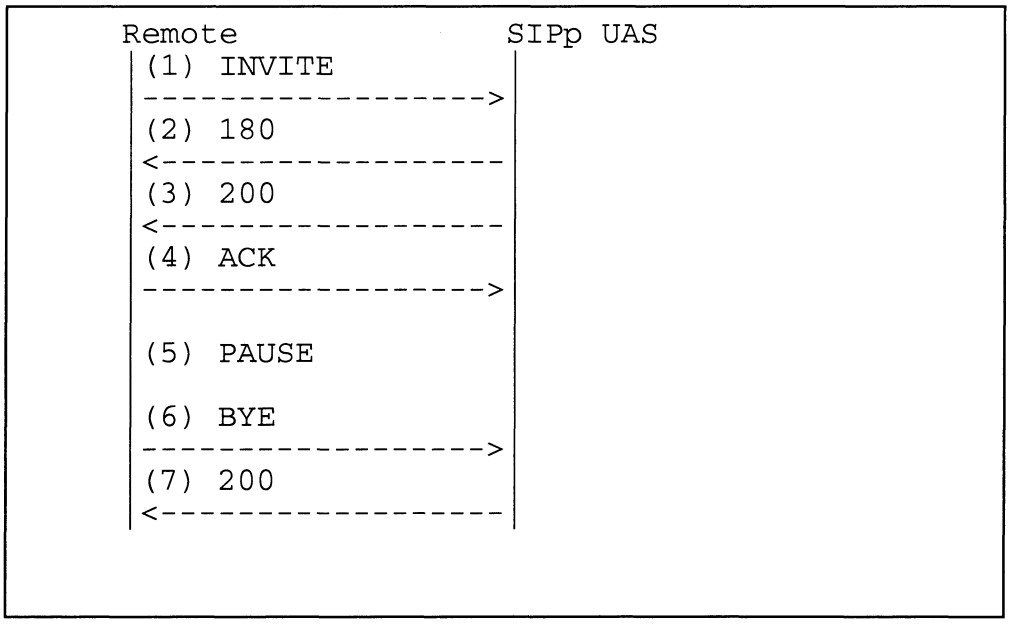

FIGURE 14. SIPp embedded UAS scenario.

Using SIPp with Integrated Scenarios

The below command starts the UAS scenario which will be waiting for incoming calls to respond with OK response.

```
# ./sipp -sn uas
```

In order to generate the calls, we can now start the UAC scenario either on the same host or on a different host that is connected to the host running the UAS scenario.

Below is the command to start the UAC scenario on the same host where UAS scenario is running. This command starts the UAC scenario and sends the call requests to the server on 127.0.0.1 interface which is the local loopback interface.

```
# ./sipp -sn uac 127.0.0.1
```

The Figure below shows the UAS scenario screen capture while the calls are in progress between UAC and UAS scenarios running on the same host.

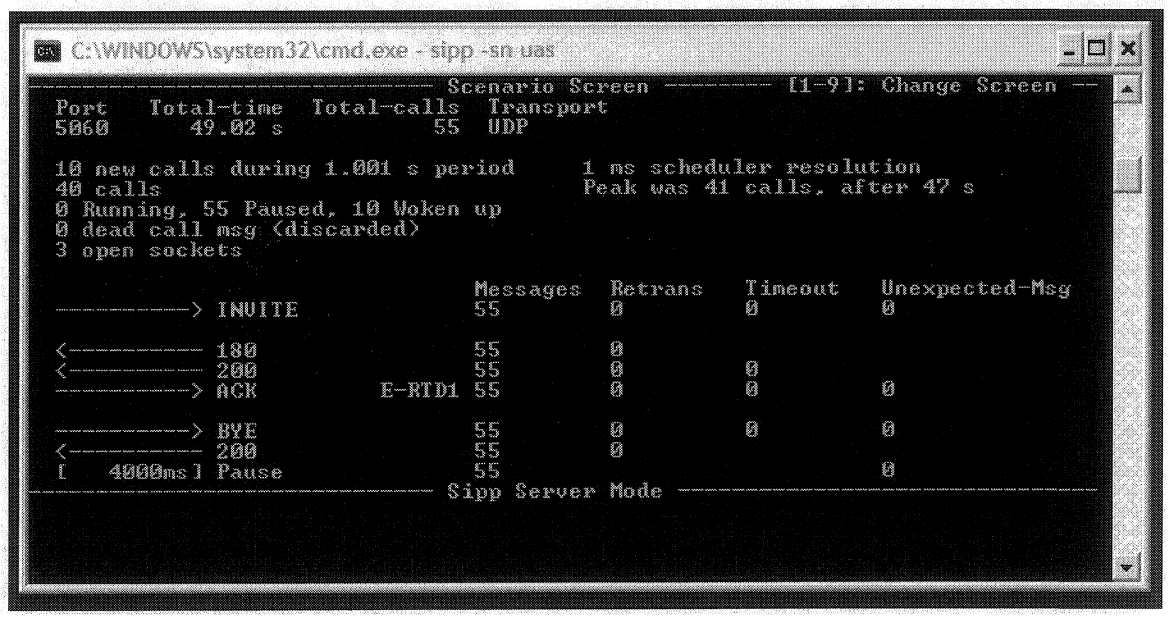

FIGURE 15. SIPp UAS scenario screen capture.

The Figure below shows the UAC scenario screen capture while the calls are in progress between UAC and UAS scenarios running on the same host. This screen

capture shows that SIPp dynamically reports the number of generated calls per second, total number of calls generated, and various SIP messages sent and received.

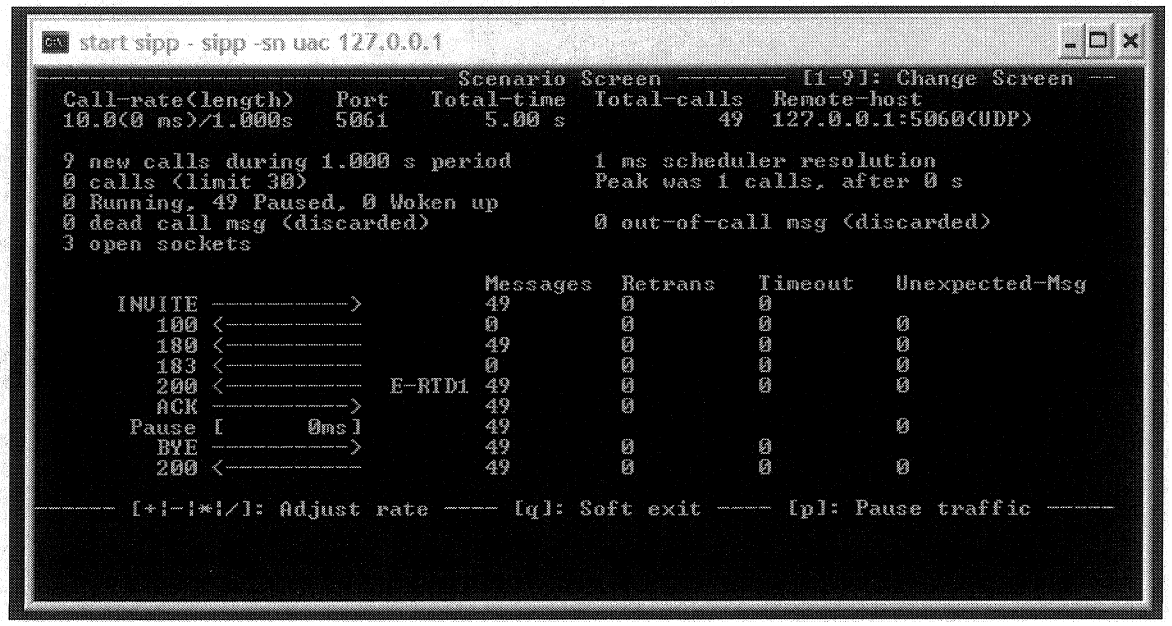

FIGURE 16. SIPp UAC scenario screen capture.

The Figure below shows the UAS statistics after terminating the UAS scenario. As seen in the screen capture in below Figure, SIPp reports a summary of various statistics that include start time, last reset time, current time, elapsed time, call rate, number of incoming calls created, number of outgoing calls created, total calls created, number of successful calls, number of failed calls, average response time of the calls, and the call length. As this is an UAS scenario, we only have the incoming calls.

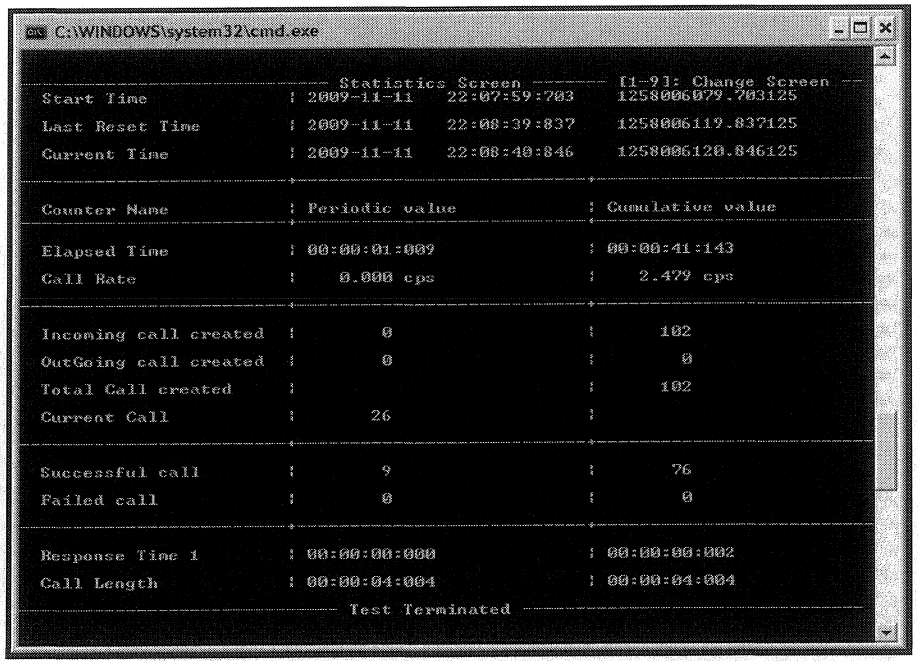

FIGURE 17. SIPp UAS scenario statistics

The Figure below shows the UAC statistics after terminating the UAC scenario. As seen in the screen capture in below Figure, SIPp reports a summary of various statistics that include start time, last reset time, current time, elapsed time, call rate, number of incoming calls created, number of outgoing calls created, total calls created, number of successful calls, number of failed calls, average response time of the calls, and the call length. As this is an UAC scenario, we only have the outgoing calls.

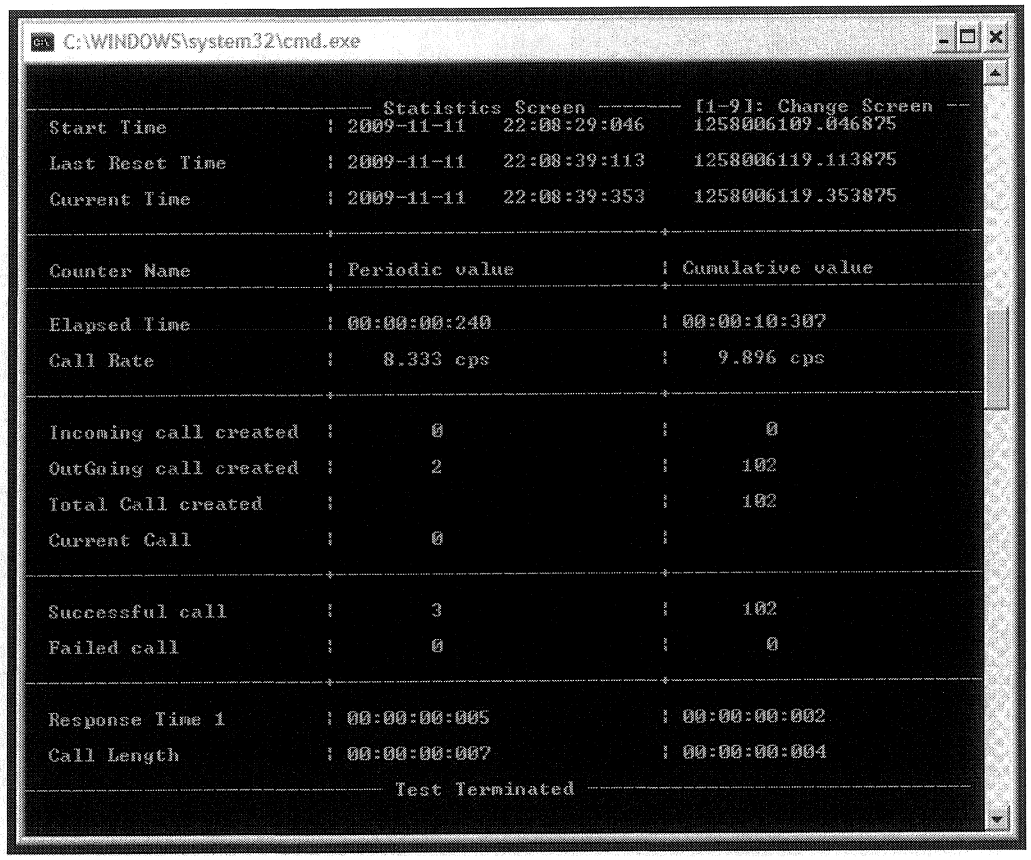

FIGURE 18. SIPp UAC scenario statistics

Developing and Using New SIPp XML scenarios

With just the embedded scenarios, the use of SIPp will be limited. The ability to develop new scenarios using XML syntax quickly makes SIPp most powerful SIP testing tool. Please refer to SIPp documentation for the syntax, keywords, and examples to create a new SIPp scenario. The various XML scenarios developed as part of performance evaluation will be described in the next section along with the description of tests.

The other important reason that makes SIPp a powerful SIP testing tool is the support of various options that can be used to fine tune the performance measurements and fine tune SIPp scheduling of its various tasks.

Open SIP Server (OpenSIPS)

The Open SIP Server (OpenSIPS) is a mature Open Source implementation of a SIP server [13]. It includes SIP registrar server, SIP proxy/router server, and also application level functionalities. It supports UDP, TCP, and TLS transport layers. We have chosen OpenSIPS as it is the open source implementation and one of the enterprise or carrier-grade class servers with high performance.

The OpenSIPS can be used as a SIP registrar server, a SIP proxy server, a SIP redirect server, a SIP location server, a SIP presence agent, a SIP back-to-back UA, SIP IM server, SIP to SMS gateway, SIP to XMPP gateway, SIP load-balancer, SIP front end for gateways, SIP NAT traversal unit, or SIP application server. In our study, we have used it as a SIP registrar server or as a SIP proxy server. The information on installing the OpenSIPS is provided in the OpenSIPS Installation Notes [14]. The OpenSIPS configuration is through the opensips.cfg script file. Various configuration parameters and the default opensips.cfg script file with TLS support are given in the OpenSIPS TLS Support documentation [15].

OProfile

The OProfile is a profiling system for Linux 2.2/2.4/2.6 systems on a number of architectures [16]. It supports profiling all parts of a running system (kernel, modules, interrupt handler, shared libraries, binaries, etc...). It is capable of profiling all parts of a running system, from the kernel (including modules and interrupt handlers) to shared libraries to binaries. It runs transparently in the background collecting information at a low overhead. These features make it ideal for profiling entire systems to determine bottle necks in real-world systems. OProfile uses various performance counters (like cache misses counter, CPU cycles counter, etc...) provided by the hardware and provides with profiles of code based on the number of these occurring events. It also supports call-graph option where it will record the function stack every time it takes a sample. OProfile has proven to be useful in a number of scenarios where we need a low overhead and less intrusive profiling system for capturing the performance behavior of the entire system. The instructions on using OProfile are given in the OProfile website [17].

The opreport utility is used to report formatted data out of OProfile. It produces two types of data: image summaries and symbol summaries. An image summary lists the number of samples for individual binary images such as libraries or applications. Symbol summaries provide per-symbol profile data. Following is the specific command used for reporting the profiling information logged by OProfile.

```
#opreport  --demangle=smart --symbols usr/local/sbin/opensips
```

Systems Used for Testing

For running SIPp UAC and UAS scenarios as load generators and call handlers, I have used the Dell Inspiron 14R, Dell Inspiron 15R, HP Compaq CQ57-315NR laptops. SIPp compiled with TLS support is installed on these laptops. The Open Secure Socket Layer (OpenSSL) library, IPv6 extension for cygwin, and libncurses needed for running SIPp with TLS are also installed. Below are the instructions used for compiling SIPp. When running in Windows, cygwin environment will be needed for compiling SIPp.

```
# gunzip sipp-xxx.tar.gz
# tar -xvf sipp-xxx.tar
# cd sipp
# make ossl
```

For running the OpenSIPS, I have used the Dell Precision T3400 workstation which has Intel Pentium 4 CPU running at 2.80GHz. The cache size is 512KB and the total main memory is 1GB. Only one CPU core is enabled for the performance evaluation as we want to load the CPU utilization to 100% with few load generators to keep the setup simple and to exclude the overhead due to SMP environment. The Netgear ProSafe gigabit Ethernet switch (GS116NA) is used for interconnecting SIP elements.

Procedure for Testing

Configure all the load generators (SIPp UAC agents), SIP proxy server, SIP registrar server, and call handlers (SIP UAS agents) to be in the same network. Modify the opensips.cfg as needed. Various configurations include specifying the listening interface, transport protocol, L4 port id, configuration for relaying the messages to caller.

Start the OpenSIPS server on SIP server machine (server under test) using the below command.

#opensips

Add the user location (caller location) to the database using the below command. In this example, the user name is 1003 and its location is identified by SIP URI of 'sip:1003@192.168.1.100:5061'.

opensipsctl ul add 1003 sip:1003@192.168.1.100:5061

Start the UAS agents on call handling machines by running SIPp UAS scenarios listening for the incoming calls. Below is SIPp command to start a UAS scenario listening for incoming call requests using TLS on 192.168.1.100 interface. In this command, the L4 port being listened to is also provided. Also, the TLS certificate and private key are provided.

sipp -sn uas -i 192.168.1.100 -p 5061 -t ln -l 900 -tls_cert ./user-cert.pem -tls_key ./user-privkey.pem

Now, start the load generators (SIPp UAC scenarios) so that calls are established through SIP proxy server. Below is the command to start the UAC scenarios using TLS. With this command, SIPp sends the call requests at the rate of 100 calls per second using TLS. It sends these call requests out on the 192.168.1.200 interface and they are targeted to the 192.168.1.2 interface on SIP proxy server.

#sipp -sn uac 192.168.1.2:5061 -i 192.168.1.200 -p 5061 -t ln -tls_cert ./user-cert.pem -tls_key ./user-privkey.pem –l 900 –s 1003 –r 100

In order to change the load, stop the current UAC scenario and start a new scenario with calls per second configured as needed. Or, more UAC agents can be started for generating more loads to SIP proxy server. While a test is in progress, measure the CPU utilization on SIP proxy server using the average over the test duration. Also, repeat the test to profile the CPU cycles spent on various software components. For profiling, start the OProfile while the test is running and stop the OProfile after the test duration has elapsed. Save the OProfile logged output so we have the information needed to analyze the percentage of CPU cycles spent among various software components.

CHAPTER 6

PERFORMANCE EVALUATION

In this chapter, we discuss the test cases used for measuring SIP server throughput rates and corresponding CPU utilization, the performance results obtained, and the OProfile profiling information noted.

Evaluating Performance Impact Due to Authentication

The performance impact due to digest based authentication in SIP is evaluated by measuring the RPS of SIP registrar servers with and without the digest authentication enabled.

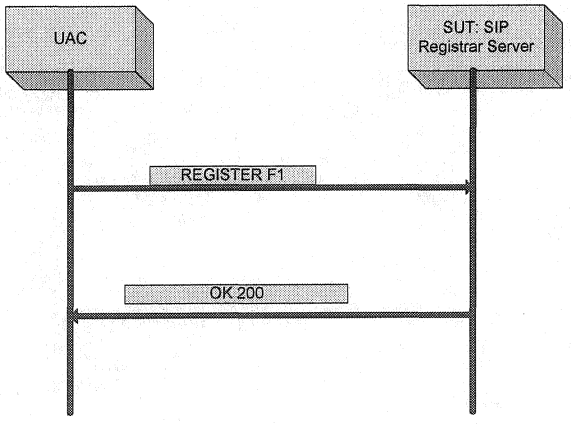

FIGURE 19. Registration without authentication

The Figure 19 above shows the call flow between the UAC and SIP registrar server when registrar does not implement Authentication. In this case, the UAC sends the REGISTER requests without authentication request and the Registrar Server responds with the OK responses. A sample REGISTER request without authentication is shown in below Figure 20.

```
REGISTER sip:registrar SIP/2.0
Via: SIP/2.0/UDP origin
From: <sip:>;tag=1
To: sip:registrand
Call-ID: call-id value
CSeq: 1 REGISTER
Contact: <sip:IP address>
Expires: 7200 Content-Length: 0
```

FIGURE 20. REGISTER request without authentication.

The message flow for registrations with authentication is shown in below Figure 21. In this case, the UAC first sends the REGISTER request without authentication. The UAS responds back with the UNAUTHORIZED response. The UAC then sends the REGISTER request with the AUTHORIZATION header. A sample REGISTER request with the AUTHORIZATION header is shown in below Figure 22.

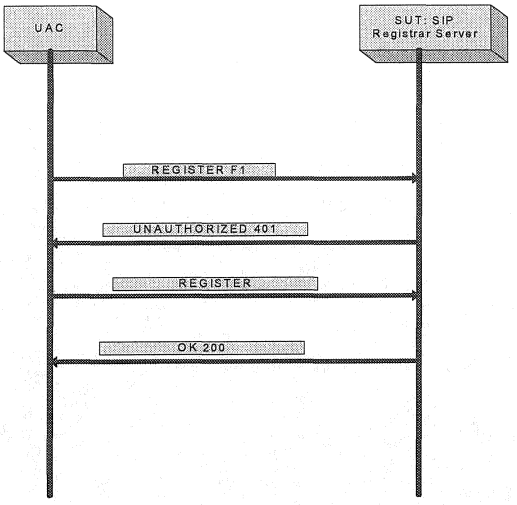

FIGURE 21. Registration with authentication.

```
REGISTER sip:registrar SIP/2.0
Via: SIP/2.0/UDP origin
From: <sip:registrand>;tag=1
To: sip:registrand
Call-ID: call-id value
CSeq: 1 REGISTER
          Contact: <sip:IP address>
Expires: 7200 Authorization:Digest
username="registrand",
realm="SIPstone",
nonce="ea9c8e88df84f1cec4341ae6cbe5a359",
opaque="", uri="registrar",
response="dfe56131d1958046689cd83306477ecc"
Content-Length: 0
```

FIGURE 22. Authenticated REGISTER request

SIPp does not have built in UAC and UAS scenarios for SIP authentication. So, we have developed an XML file for this scenario and it is provided in Appendix-A. The Figure 23 below shows SIPp UAC using the XML scenario for the Registration with Authentication.

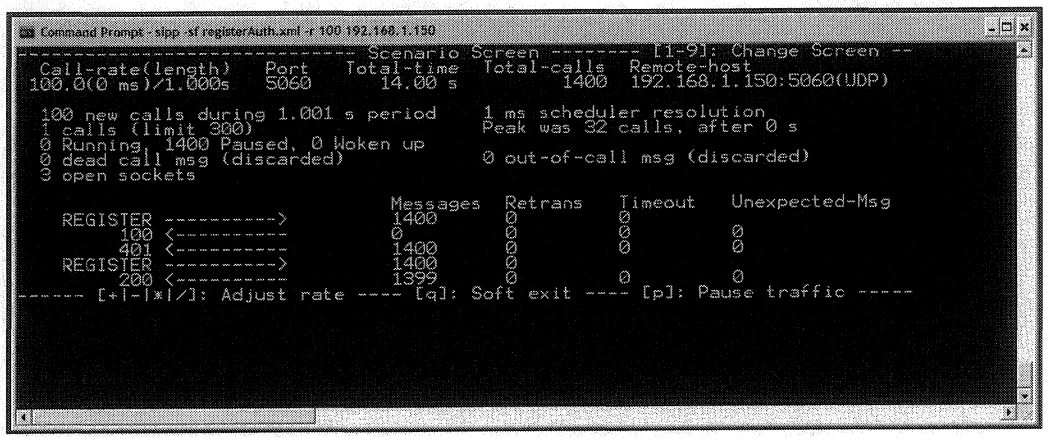

FIGURE 23. SIPp screenshot of registration with authentication

SIP Registrar Performance Results

The CPU utilization of SIP server is measured with UDP or TCP as transport, and with or without Authentication. This means, we ran four different test cases: UDP transport and no authentication, UDP transport with authentication, TCP transport and no authentication, and TCP transport with authentication. For each of these four test cases, the CPU utilization is measured with varying workloads. These measurements are shown in graphical form in Figures 24 to 28. These measurements in table form are also provided in Appendix C.

The Figure 24 below shows comparison of SIP performance with and without Authentication using UDP as transport. As expected, the CPU utilization is higher for a given number of RPS when authentication is enabled. With UDP transport, the CPU utilization reaches 100% for approximately 5750 Registrations Per Second (RPS) when authentication is enabled where as the CPU utilization is only 29.7% for approximately 8000 RPS when authentication is disabled.

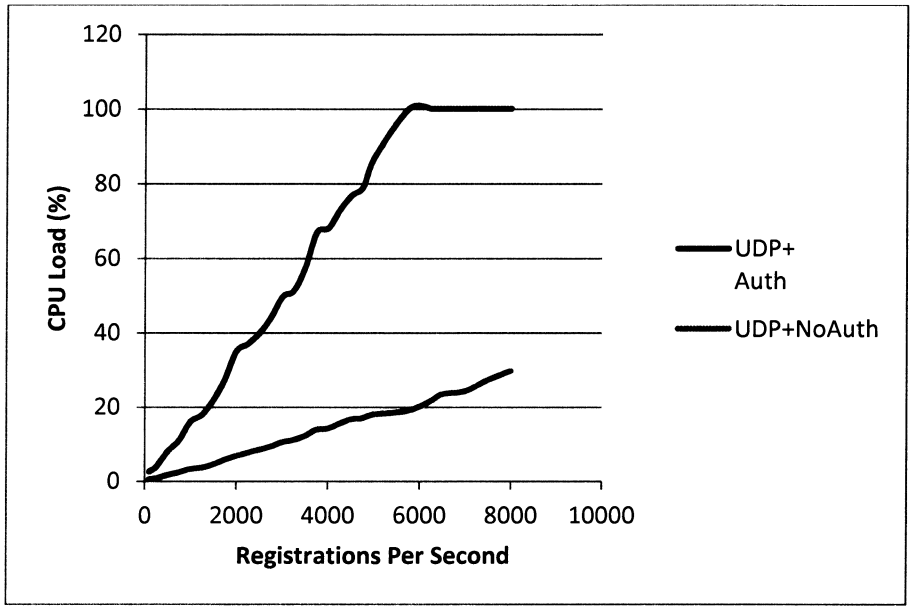

FIGURE 24. SIP registrar performance comparison--UDP.

One way to compare the two test cases is by looking at the CPU utilization for a given number of RPS. For 5750 RPS, the CPU utilization is 19% when authentication is disabled and 100% when authentication is enabled. This means the degradation in

performance is approximately five times at 5750 RPS. In a similar fashion, the performance degradation for various RPS loads is calculated. The Figure 25 below shows the degradation in SIP registrar performance at various RPS loads. From this graph, we can note that the performance degradation varied from 4.63 times to 5.26 times depending on the RPS load. Taking an average of these results (minimizing measurement error), we have noted a performance degradation of approximately five times due to the SIP authentication. The performance degradation is approximately five times at any given RPS load and this suggests that this information can also be used for extrapolation when needed. However, it should be noted that platform specific improvements in CPU speed and memory access affecting the digest calculations may yield different numbers. We hope that these measurements give a good approximation of costs to be incurred when using SIP digest based authentication of SIP users.

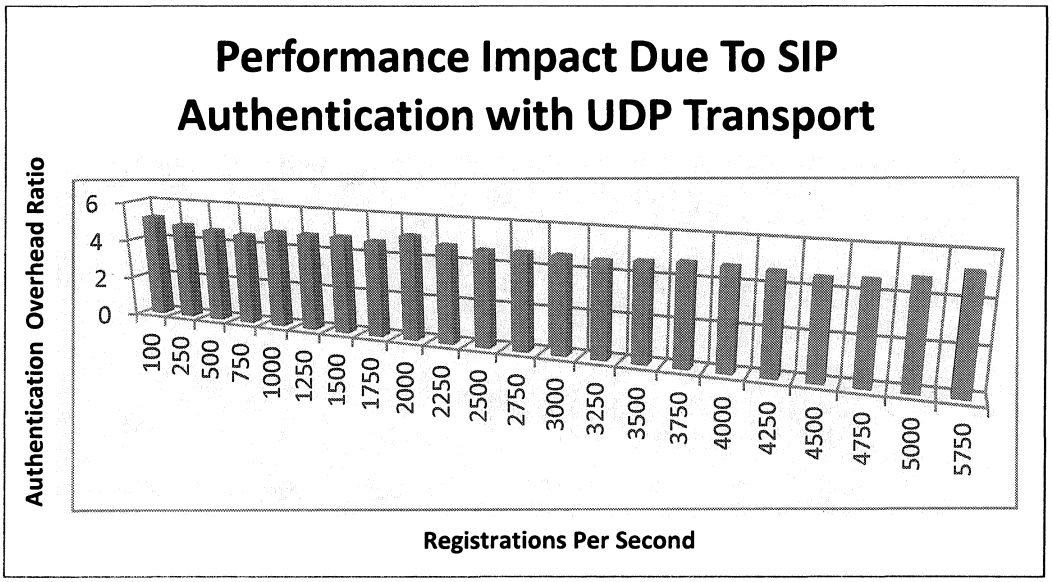

FIGURE 25. SIP registrar performance degradation--UDP.

The Figure 26 below shows comparison of SIP performance with and without Authentication using TCP as transport. With TCP transport, the CPU utilization reaches 100% for approximately 4500 Registrations Per Second (RPS) when authentication is enabled. The CPU utilization is only 42.5% for 8000 RPS when authentication is disabled. As expected the performance (RPS) has come down when TCP is used as transport protocol compared to UDP case due to the TCP overhead.

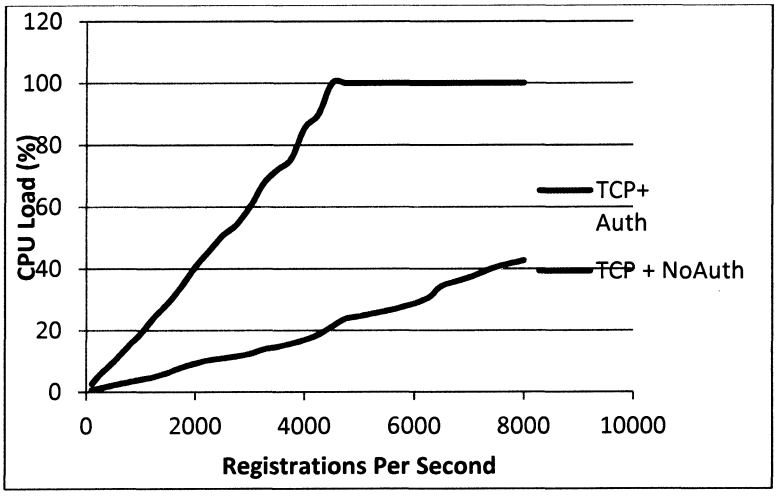

FIGURE 26. SIP registrar performance comparison--TCP.

The Figure 27 below shows the degradation in SIP registrar performance at various RPS loads. As with the UDP case, the performance degradation has varied from 4.5 times to 5.4 times depending upon the RPS load. Taking the average of performance degradation measured at different loads, we can note that the performance degradation

due to SIP authentication is approximately five times. These results also indicate that the performance degradation due to SIP digest based authentication is approximately five times whether the transport protocol is UDP or TCP.

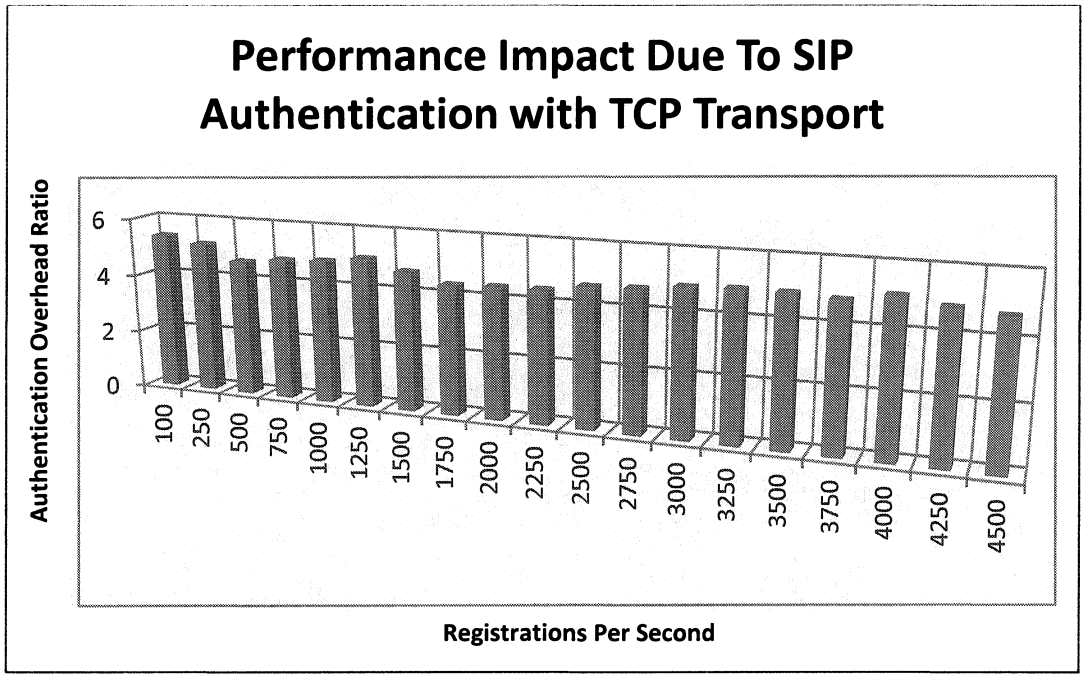

FIGURE 27. SIP registrar performance degradation--TCP.

The below Figure 28 shows all the four (two TCP and two UDP) test cases in one graph. It is clear from the graph that UDP provides more performance than TCP as UDP has less overhead compared to TCP. It is also clear that the performance degradation in both UDP and TCP cases is almost same.

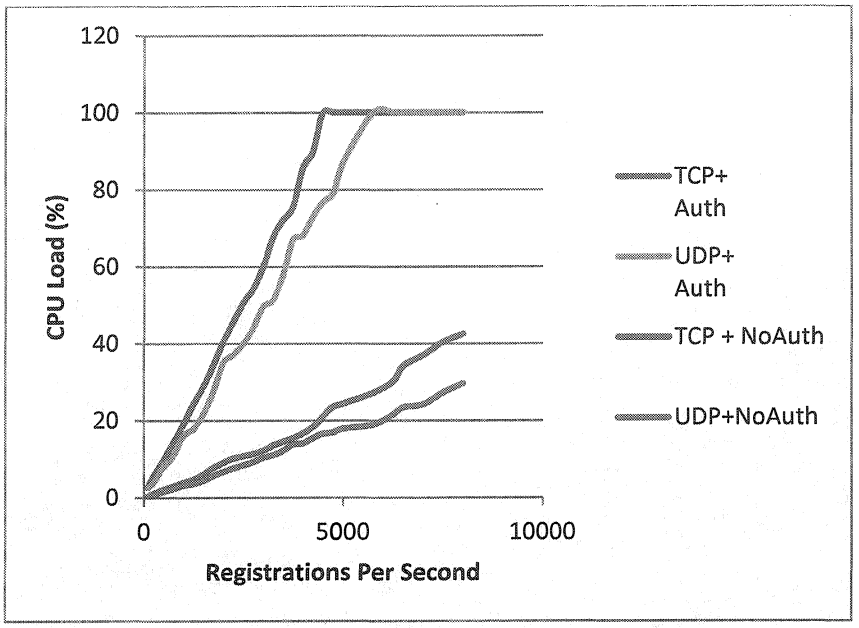

FIGURE 28. SIP registrar performance comparison.

Evaluating Performance Impact Due to TLS

The performance impact due to TLS in SIP is evaluated by measuring the CPU utilization under varying loads of CPS using different test cases that have different transports and TLS settings. There are three different TLS settings to be taken into account: authentication (only client authenticating server or mutual), proxy server operating mode (proxy in the middle of proxy chain or inbound proxy or outbound proxy or local proxy), and session reuse (enabled or disabled). Note that the operating mode of a proxy is logical and all the other operating modes are special cases of local proxy mode. As our focus is in evaluating the impact of TLS overhead, we have chosen only one

operating mode (local proxy mode) so as to minimize the number of test cases while still covering the test cases needed for our evaluation.

Most of the use cases are where clients authenticate server and not the mutual authentication. So, we need only one test case with mutual authentication to evaluate the performance impact due to mutual authentication compared to the case of only client authenticating server. Similarly, the benefit with session reuse is evaluated by picking up only one test case with session reuse enabled. The TLS test cases we have chosen are: Local proxy with client authentication and no session reuse (TLS-Local-Client), Local proxy with client authentication and session reuse (TLS-Local-Client-SessionReuse), and Local proxy with mutual authentication and no session reuse (TLS-Local-Mutual). To summarize, the different test cases we have used are: a) UDP transport, b) TCP Mono (Single socket), c) TCP Multi (Multiple sockets), d) TLS-Local-Client, e) TLS-Local-Client-SessionReuse, and g) TLS-Local-Mutual. The results measured are provided in the Appendix D and plotted in this section.

The performance degradation due to server also authenticating client is obtained by comparing the results from TLS-Local-Client and TLS-Local-Mutual test cases. The difference between these two cases is the additional overhead for the server to request the certificate from client and authenticating it in case of mutual authentication. The Figure 29 below shows the performance degradation due to mutual authentication compared to client only authentication. The CPU utilization has reached 100% for around 95 calls per

second in case of mutual authentication and for around 135 calls per second in case of client only authentication.

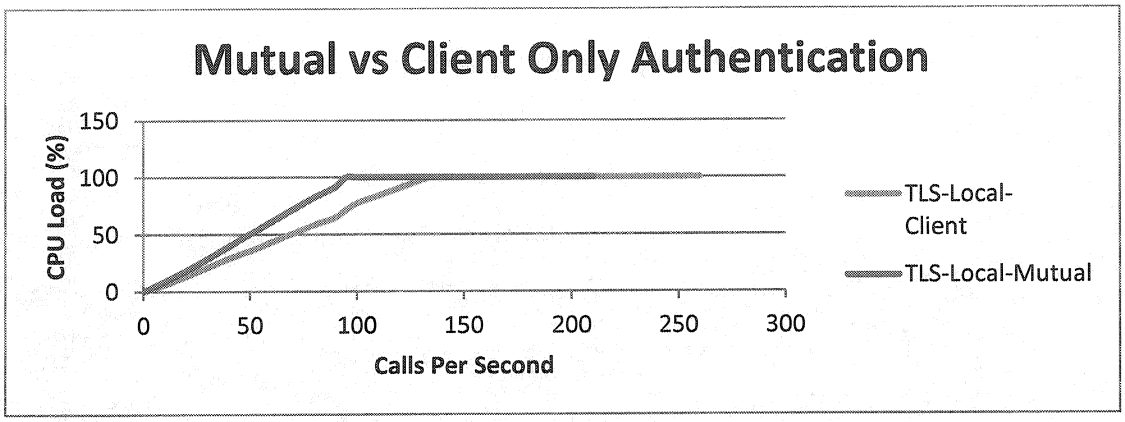

FIGURE 29. SIP proxy server performance impact due to mutual authentication.

The performance gain with session reuse enabled is obtained by comparing the results from TLS-Local-Client and TLS-Local-Client-SessionReuse test cases. These results are shown in below Figure 30. As can be seen from the Figure, the CPU utilization has reached 100% for about 135 calls per second in case of TLS-Local-Client case which is not using the session reuse and for about 320 calls per second in case of TLS-Local-Client-SessionReuse case which is using the session reuse. For this test, note that all the calls are made under few sessions which are established first time and are cached for subsequent use as session reuse is enabled.

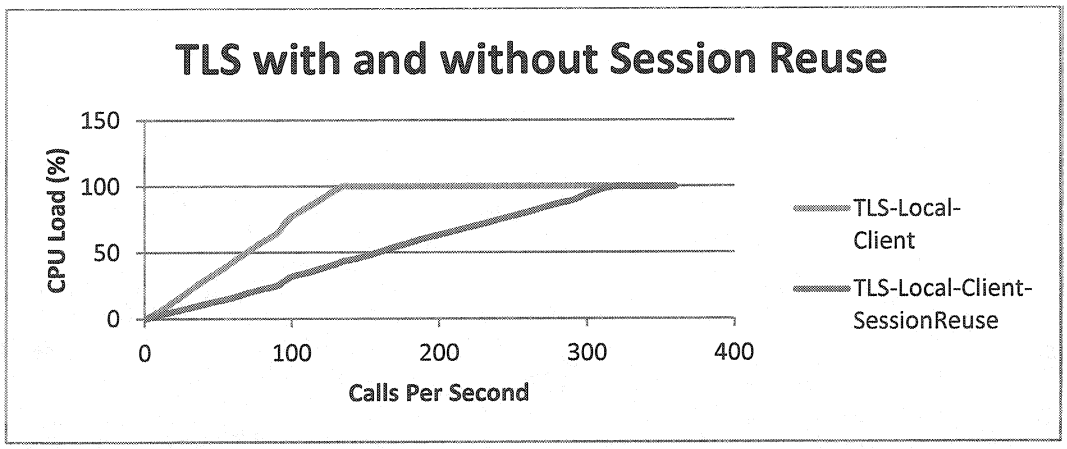

FIGURE 30. SIP proxy server performance gain with session reuse.

The performance impact on SIP proxy due to TLS is obtained by comparing the results from UDP and TCP test cases with the results from TLS-Local-Client and TLS-Local-Mutual test cases. These results are plotted in below Figure 31. The CPU utilization reached 100% for around 95 calls per second load in case of TLS-Local-Mutual test case where TLS is enabled with mutual authentication and multiple incoming and outgoing TLS connections. In case of TLS with client only authentication and session reuse not enabled, the peak throughput (where CPU utilization has reached 100%) is around 135 calls per second. In case of TLS with client only authentication and session reuse enabled, the peak throughput is measured to be around 320 calls per second. The peak throughput for TCP case is around 865 calls per second when multiple sockets are used and around 1130 calls per second. Finally, the peak throughput measured for UDP case is 1810 calls per second.

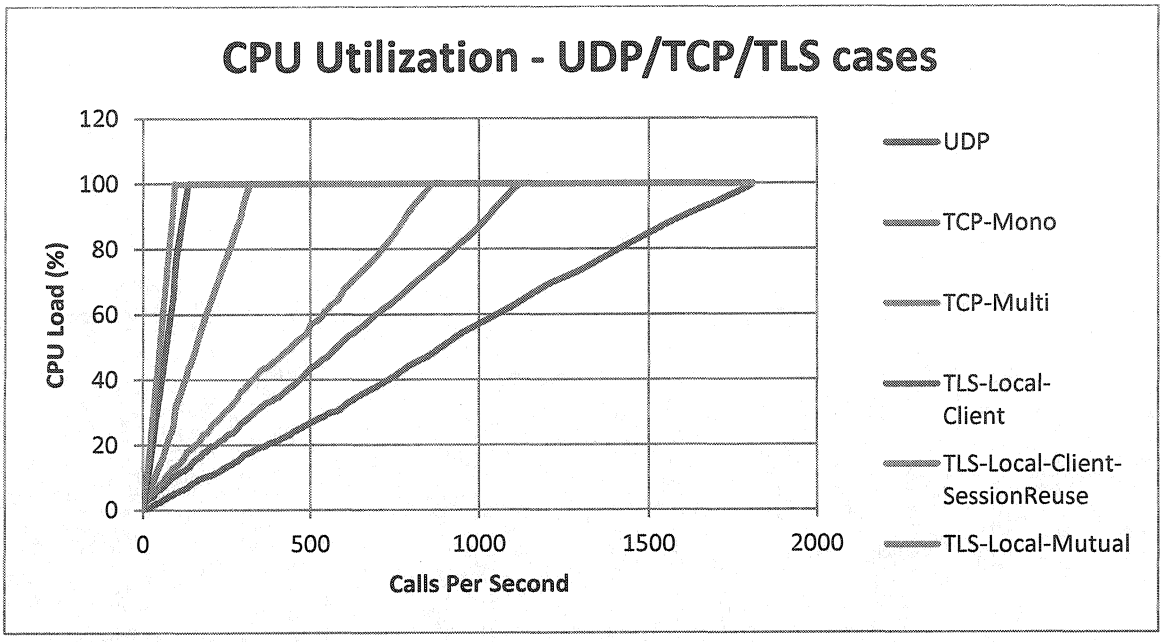

FIGURE 31. SIP proxy server performance comparison.

The peak CPS throughput for the different test cases is obtained from the data in above Figure 31 and is shown in graphical form in below Figure 32. From this data, we can infer that there is about nineteen times peak throughput degradation when TLS-Local-Mutual is compared with the UDP case. And the degradation is about nine times when compared to the TCP multiple socket case. The TCP multiple socket case provides only 47% of peak performance that can be achieved with UDP. The overhead due to several TCP connections can be obtained by comparing the TCP Mono and TCP Multi cases and this overhead is about 14 %. The performance degradation due to the server also authenticating server when compared to the case of only client authenticating server is about 30%. When session reuse is enabled, the performance gain due to bypassing the

asymmetric cryptographic operations and avoiding the overhead of exchange of few messages needed for negotiation of secret key and other parameters is about 2.37 times.

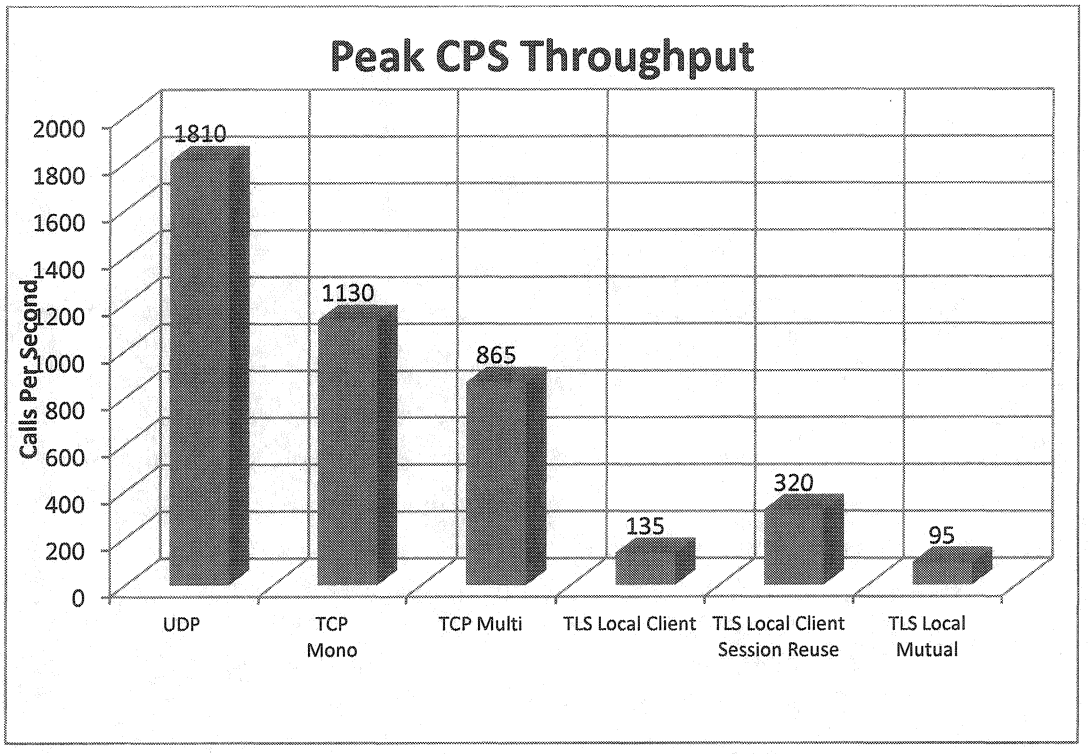

FIGURE 32. SIP proxy peak CPS throughputs.

Impact of TLS--OProfile Results

The SUT is profiled using Linux OProfile utility while different test are running and the collected logs are analyzed for determining the CPU time spent on various software components of interest. The summary of OProfile Results measured is provided in tables in Appendix E and are plotted and discussed in this section. The Figure 33

below shows the OProfile results summary for the UDP case with various loads. The three software components where CPU time is spent apart from the kernel are the opensips server, libc library, and tm library. Around 25% of CPU cycles are spent in opensips, 20% of CPU cycles are spent in libc, and 10% of CPU cycles are spent in tm library. What this means is that 25% of CPU utilization measured is due to the opensips software, and 20% of CPU utilization measured is due to the libc library functions, and 10% of CPU utilization measured is due to the timer library (tm library). The CPU utilization measured will be different for different traffic loads and we have already noted this information as part of our TLS performance measurements. Another important point we can also note from Figure 33 is that this division of CPU cycles between different software components is almost same under different CPS loads.

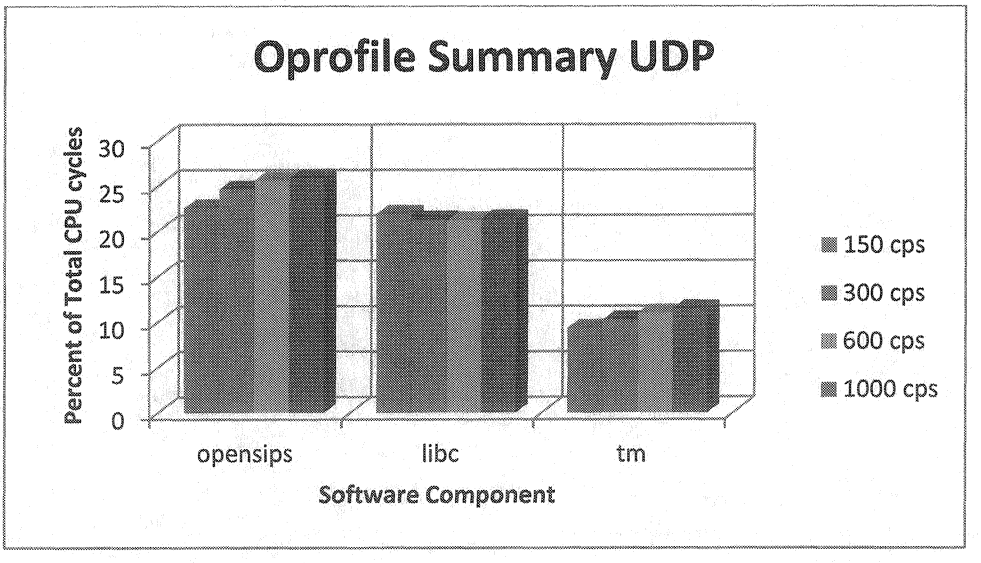

FIGURE 33. OProfile results summary--UDP

The Figure 34 below shows the OProfile results summary for the TCP single socket case with various loads. The three software components where CPU time is spent apart from the kernel are the opensips server, libc library, and tm library. Around 25% of CPU cycles are spent in opensips, 16% of CPU cycles are spent in libc, and 7% of CPU cycles are spent in tm library. These results obtained with the OProfile will have to be multiplied with the CPU utilization measured to obtain the CPU utilization due to each of these software components of interest. We can also note that the division of CPU cycles between different software components is almost same under different CPS loads.

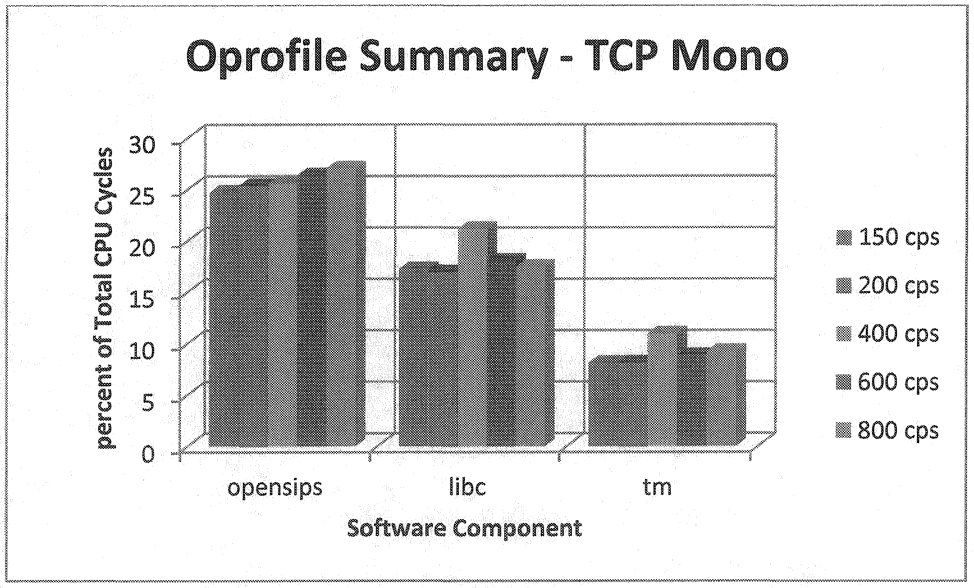

FIGURE 34. OProfile results summary--TCP mono.

The Figure 35 below shows the OProfile results summary for the TCP multi socket case with various loads. The three software components where CPU time is spent apart from the kernel are the opensips server, libc library, and tm library. Around 27% of CPU cycles are spent in opensips, 14% of CPU cycles are spent in libc, and 6% of CPU cycles are spent in tm library. These results obtained with the OProfile will have to be multiplied with the CPU utilization measured to obtain the CPU utilization due to each of these software components of interest. We can also note that the division of CPU cycles between different software components is almost same under different CPS loads.

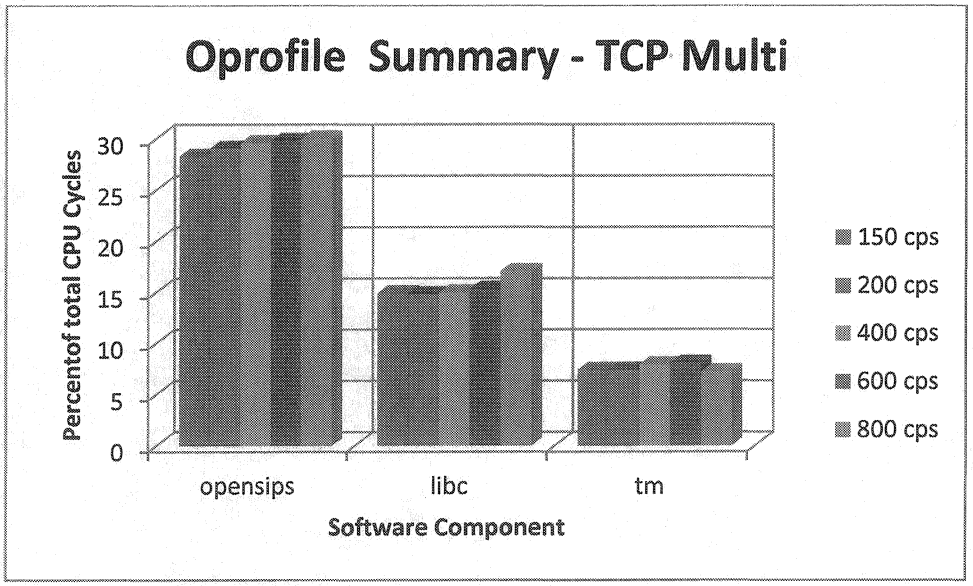

FIGURE 35. OProfile results summary--TCP multi.

The Figure 36 below shows the OProfile results summary for the TCP-Local-Client-SessionReuse case (TLS with client only authentication and session reuse enabled)

with various loads. The four software components where CPU time is spent apart from the kernel are the opensips server, libcrypto library, libssl library, and libc library. Around 18% of CPU cycles are spent in libcrypto library, 23% of CPU cycles are spent in libc, 3% of CPU cycles are spent in libssl library , and 11% of CPU cycles are spent in libc library. These results obtained with the OProfile will have to be multiplied with the CPU utilization measured to obtain the CPU utilization due to each of these software components of interest. We can also note that the division of CPU cycles between different software components is almost same under different CPS loads.

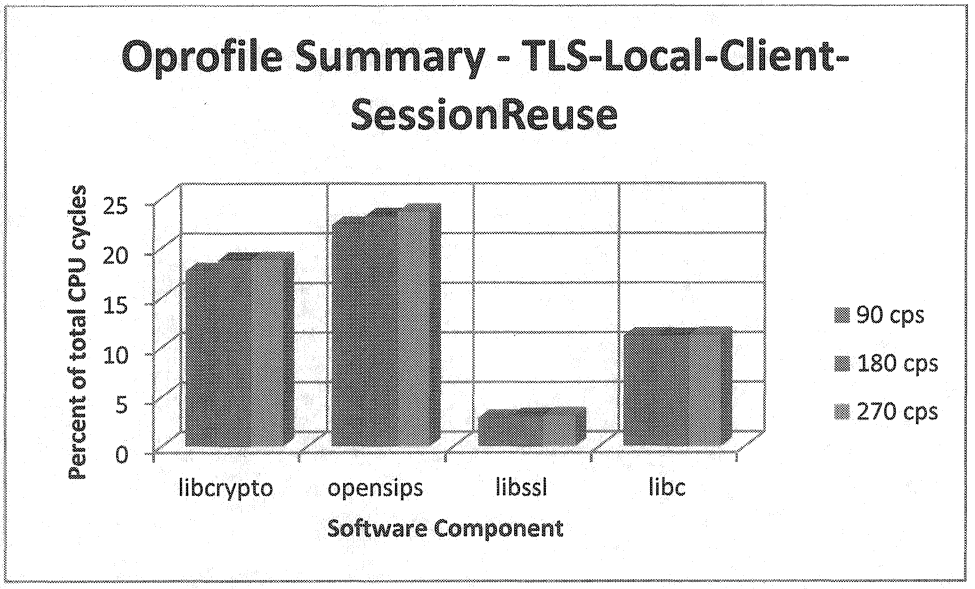

FIGURE 36. OProfile results summary--TLS local client session reuse.

The Figure 37 below shows the OProfile results summary for the TCP-Local-Client case with various loads. The four software components where CPU time is spent apart from the kernel are the opensips server, libcrypto library, libssl library, and libc

library. Around 48% of CPU cycles are spent in libcrypto library, 14% of CPU cycles are spent in libc, 3% of CPU cycles are spent in libssl library , and 8% of CPU cycles are spent in libc library. These results obtained with the OProfile will have to be multiplied with the CPU utilization measured to obtain the CPU utilization due to each of these software components of interest. We can also note that the division of CPU cycles between different software components is almost same under different CPS loads.

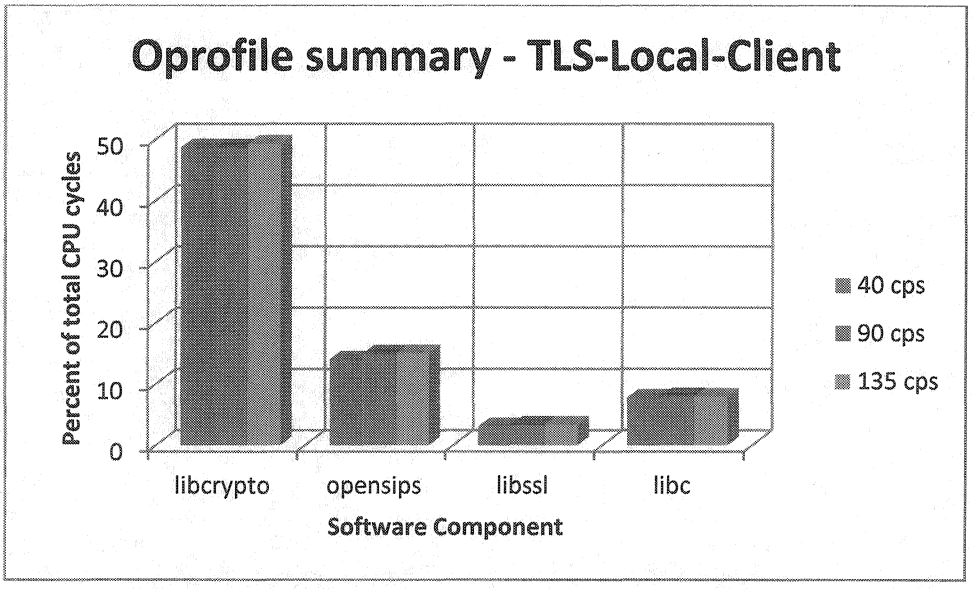

FIGURE 37. OProfile results summary--TLS local client.

The Figure 38 below shows the OProfile results summary for the TCP-Local-Mutual case with various loads. The four software components where CPU time is spent apart from the kernel are the opensips server, libcrypto library, libssl library, and libc library. Around 58% of CPU cycles are spent in libcrypto library, 10% of CPU cycles are

spent in libc, 2% of CPU cycles are spent in libssl library , and 5% of CPU cycles are spent in libc library. These results obtained with the OProfile will have to be multiplied with the CPU utilization measured to obtain the CPU utilization due to each of these software components of interest. We can also note that the division of CPU cycles between different software components is almost same under different CPS loads.

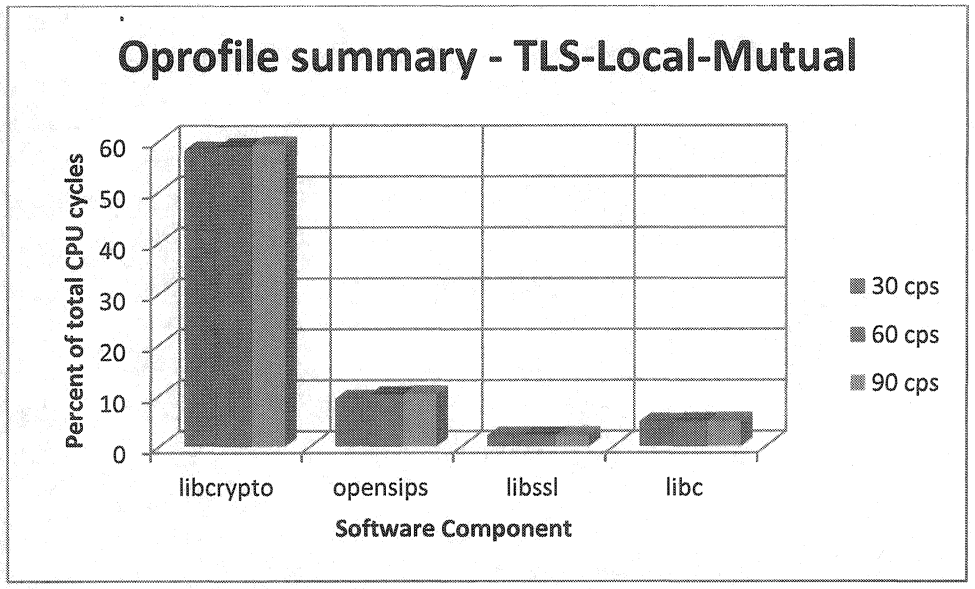

FIGURE 38. OProfile results summary--TLS local mutual.

To compare the results from different test cases, we have normalized the results with CPU utilization measured. These normalized results for different test cases at a CPU load of 90 CPS are plotted in the below Figure 39. From this Figure, we can see that about 54.23% of CPU time is spent in libcrypto for the TLS-Local-Mutual, about

31.51% of CPU time is spent in libcrypto for the TLS-Local-Client, and about 4.43% of CPU time is spent in libcrypto for the TLS-Local-Client-SessionReuse. This means, the asymmetric cryptographic operation in TLS session establishment have increased the CPU utilization from 4.43% to 31.51% or 54.23% depending on whether the authentication is client only or mutual. Note that even though the cost due to libcrypto has gone up by 7.11 % between the TLS-Local-Client and TLS-Local-Client-SessionReuse cases, the overall benefit of session reuse is about 2.37 times. As can be seen from the below Figure, this is due to the fact the costs associated with other components do not scale in the same ratio as that of libcrypto between the 2 cases.

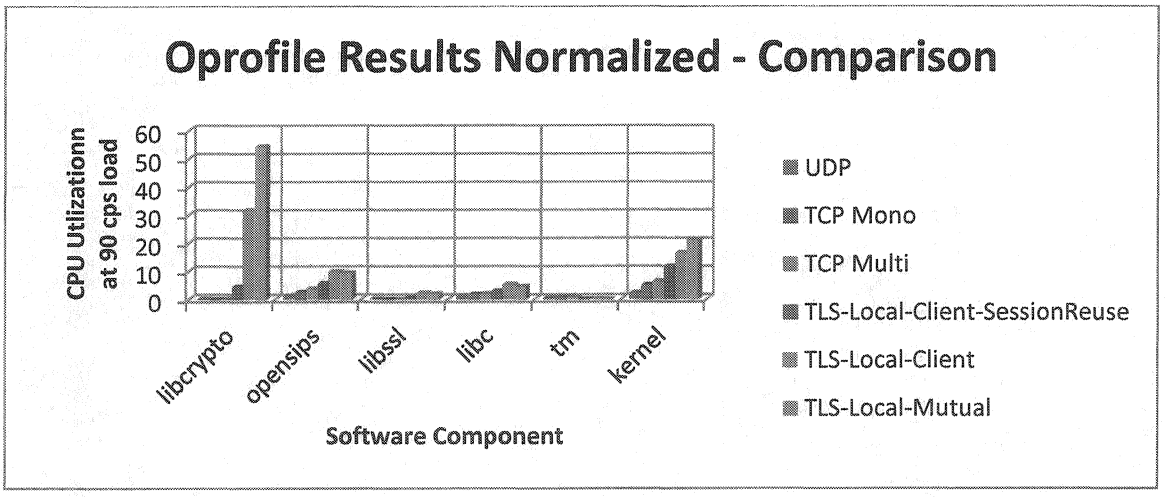

FIGURE 39. Comparison of OProfile results normalized

From the OProfile summary results plotted in Figures 33 to 38 above, we can also notice that this division of CPU cycles, for a given test case, between different software components is almost same under different CPS loads. This means, we can also extrapolate the measured results for higher CPS loads as long as the platform under test can be assumed to have similar processor and memory access performance, though the measured results are specific to a given test setup.

CHAPTER 7

CONCLUSIONS

With rapid evolution of VoIP with SIP widely used as the signaling protocol, SIP security is becoming extremely important. In this thesis, we have studied the various security threats to SIP like *registration hijacking, message modification, impersonating a server, Denial of Service (DoS), session tear down, and Replay attack.* We have also studied the various security mechanisms (Authentication, S/MIME, and TLS) in practice to mitigate these threats. We have also evaluated the impact on performance of SIP servers due to Authentication and TLS. The open source OpenSIPS server is used as the registration or proxy server in our performance testing. The OpenSIPS server is configured through the configuration scripts for the required SIP proxy operations and security mechanisms. SIPp traffic generator is used for generating SIP traffic. The embeded UAC and UAS scenarios are used for generating and accepting the voice calls. SIPp test scenarios for registration with and without authentication are developed using the XML. We have used both UDP and TCP as transport protocols in evaluating the performance impact of Authentication on the SIP registrar server. The results show that the performance of the SIP registrar server has degraded by approximately five times due to SIP authentication. The results also show that the performance degradation noted is consistent across different SIP traffic loads with either UDP or TCP as the transport protocol.

The performance impact of TLS on the SIP proxy server is evaluated using same test-bed employing the OpenSIPS server and SIPp traffic generators. We.have used UDP, TCP, and TLS as transport protocols in this performance evaluation. We have noted that the performance of the SIP proxy server has improved by approximately 2.37 times when session reuse is enabled by comparing the peak loads obtained when using TLS with and without session reuse. The performance degradation due to mutual authentication in TLS compared to client only authentication is measured to be around 1.5 times. The performance degradation of the TLS local proxy server with mutual authentication is measured to be about nineteen times when compared to UDP case and about nine times when compared to TCP case.

We have further analyzed the CPU utilization between the various TLS test cases using the OProfile. The OProfile utility allows us to profile the system by measuring the number of CPU cycles used by various software components (libcrypto, opensips, libssl, libtm, kernel, etc) within a given period of time. Using the OProfile results and the CPU utilization measurements, we have noted that cost incurred in TLS asymmetric cryptography operations in establishing the TLS session is approximately 7.11 times by comparing the CPU time spent in libcrypto between the 2 cases of TLS with and without session reuse. The OProfile results also show that the percentage of CPU cycles used by various software components like libcrypto, opensips, and libssl for a given test case is approximately same with various SIP traffic loads. This means the results obtained can also be extrapolated to different loads without introducing a significant error.

APPENDICES

APPENDIX A

SIP MESSAGES FOR REGISTRATION WITH AUTHENTICATION

The listing below shows the various SIP messages in SIP registration with authentication enabled.

1) REGISTER Request without Authentication sent from the UAC.

```
REGISTER sip:192.168.1.150 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.151:5060;branch=z9hG4bK-1100-1-0
From: sipp <sip:sipp@192.168.1.151:5060>;tag=1
To: sut <sip:service@192.168.1.150:5060>
Call-ID: 1-1100@192.168.1.151
CSeq: 1 REGISTER
Contact: sip:sipp@192.168.1.151:5060
Max-Forwards: 70
Expires: 1800
User-Agent: SIPp/Linux
Content-Length: 0
```

2) UNAUTHORIZED Response received from SIP Registrar Server.

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 192.168.1.151:5060;branch=z9hG4bK-1100-1-0
From: sipp <sip:sipp@192.168.1.151:5060>;tag=1
To: sut <sip:service@192.168.1.150:5060>;tag=116f6bbb187428b76dfe896dc69c98ee.e87d
Call-ID: 1-1100@192.168.1.151
CSeq: 1 REGISTER
WWW-Authenticate: Digest realm="192.168.1.150",
nonce="4b1486500000110d07a8a496d88b628973d159afa7a85bcf"
Server: OpenSIPS (1.6.0-notls (i386/linux))
Content-Length: 0
```

3) REGISTER Request with the Authorization Header

```
REGISTER sip:192.168.1.150 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.151:5060;branch=z9hG4bK-1100-1-3
From: sipp <sip:sipp@192.168.1.151:5060>;tag=1
To: sut <sip:service@192.168.1.150:5060>
Call-ID: 1-1100@192.168.1.151
CSeq: 2 REGISTER
Contact: sip:sipp@192.168.1.151:5060
Authorization:Digest username="user1", realm="192.168.1.150", uri="sip:192.168.1.150:5060",
nonce="4b1486500000110d07a8a496d88b628973d159afa7a85bcf",
response="c9361531c988bf253b1e9177548973fe",algorithm=MD5
Max-Forwards: 70
Expires: 1800
User-Agent: SIPp/Linux
Content-Length:    0
```

4) OK Response from SIP Registrar upon verifying the provided credentials.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.151:5060;branch=z9hG4bK-1100-1-3
From: sipp <sip:sipp@192.168.1.151:5060>;tag=1
To: sut <sip:service@192.168.1.150:5060>;tag=116f6bbb187428b76dfe896dc69c98ee.515e
Call-ID: 1-1100@192.168.1.151
CSeq: 2 REGISTER
Contact: <sip:sipp@192.168.1.151:5060>;expires=1800
Server: OpenSIPS (1.6.0-notls (i386/linux))
Content-Length:    0
```

APPENDIX B

SIPp XML SCENARIO FOR AUTHENTICATION

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="Basic Sipstone UAC">
 <send retrans="500">
  <![CDATA[
   REGISTER sip:[remote_ip] SIP/2.0
   Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
   From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
   To: sut <sip:[service]@[remote_ip]:[remote_port]>
   Call-ID: [call_id]
   CSeq: 7 REGISTER
   Contact: sip:sipp@[local_ip]:[local_port]
   Max-Forwards: 70
   Expires: 1800
   User-Agent: SIPp/Linux
   Content-Length: 0
  ]]>
 </send>

 <recv response="100" optional="true">
 </recv>

 <recv response="401" auth="true">
 </recv>

 <send>
  <![CDATA[
   REGISTER sip:[remote_ip] SIP/2.0
   Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
   From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
   To: sut <sip:[service]@[remote_ip]:[remote_port]>
   Call-ID: [call_id]
   CSeq: 8 REGISTER
   Contact: sip:sipp@[local_ip]:[local_port]
   [authentication username=user1 password=password1]
   Max-Forwards: 70
   Expires: 1800
   User-Agent: SIPp/Linux
   Content-Length: [len]
  ]]>
 </send>

 <recv response="200">
 </recv>
</scenario>
```

APPENDIX C

SIP AUTHENTICATION – PERFORMANCE MEASUREMENTS

The table below summarizes the CPU loads for the four different test cases we have considered for evaluating the impact of SIP authentication on SIP Registrar performance.

TABLE 1. CPU Utilization With and Without Authentication

RPS	UDP+ NoAuth	TCP + NoAuth	UDP+ Auth	TCP+ Auth	(UDP+Auth)/ (UDP+No Auth)	(TCP+Auth)/ (TCP +No Auth)
100	0.5	0.5	2.6	2.7	5.2	5.4
250	0.8	1.1	3.9	5.7	4.875	5.181818182
500	1.7	2.1	8	9.8	4.705882353	4.666666667
750	2.4	3	11	14.5	4.583333333	4.833333333
1000	3.3	3.9	16	19.1	4.848484848	4.897435897
1250	3.7	4.8	17.9	24.4	4.837837838	5.083333333
1500	4.5	6.1	21.8	28.9	4.844444444	4.737704918
1750	5.8	7.8	27.5	34.4	4.74137931	4.41025641
2000	6.8	9.1	35	40.6	5.147058824	4.461538462
2250	7.7	10.2	37	45.7	4.805194805	4.480392157
2500	8.5	10.8	39.9	50.7	4.694117647	4.694444444
2750	9.3	11.4	44	54.2	4.731182796	4.754385965
3000	10.5	12.2	49.6	60	4.723809524	4.918032787
3250	11.1	13.6	51.3	67.6	4.621621622	4.970588235
3500	12.2	14.5	57.7	72	4.729508197	4.965517241
3750	13.9	15.5	67	75.5	4.820143885	4.870967742
4000	14.2	16.7	68	85.5	4.788732394	5.119760479
4250	15.5	18.4	73	89.9	4.709677419	4.885869565
4500	16.7	21	76.7	100	4.592814371	4.761904762
4750	17	23.6	79	100	4.647058824	
5000	18	24.4	87	100	4.833333333	
5750	19	27.2	100	100	5.263157895	
6250	21.7	30.3	100	100		
6500	23.5	34.2	100	100		
7000	24.3	37	100	100		
7500	27.3	40.4	100	100		
8000	29.7	42.5	100	100		

APPENDIX D

TLS PERFORMANCE EVALUATION RESULTS

TABLE 2. CPU Utilization With and Without TLS

CPS	CPU Utilization					
	UDP	TCP-Mono	TCP-Multi	TLS-Local-Client	TLS-Local-Client-SessionReuse	TLS-Local-Mutual
0	0	0	0	0	0	0
10	0.5	2.5	2.8	6	3.3	8.9
20	0.7	3.4	4	13.3	5.7	18.3
30	1.4	4	5.1	21.1	8.3	28.8
40	2	5.5	5.9	28.9	11.1	39.6
50	2.4	6.2	7.7	35.7	13.6	50.4
60	3	6.9	8.6	43.2	16.1	61.2
70	3.7	8	10.3	50.8	19.6	71.8
80	4.3	9.2	11.4	58.2	22.5	82.3
90	4.8	10.1	12.9	64.8	25.1	91.6
95	5.2	10.3	13.1	71.6	28.7	100
100	5.4	10.8	13.5	77.2	32.1	100
110	5.9	11.5	14.3	83.9	34.8	100
120	6.3	11.9	15.7	90.4	38.1	100
130	6.9	12.5	17.8	97.2	41.3	100
135	7.3	12.9	18.2	100	43.6	100
140	7.8	13.3	18.5	100	44.5	100
150	8.4	14.7	19.3	100	47.2	100
160	9	15.6	20.2	100	50.6	100
170	9.6	16.2	21.3	100	54.1	100
180	10	17.5	23.1	100	57.3	100
190	10.4	18.1	23.7	100	60.7	100
200	10.6	19.4	25.1	100	63.4	100
210	11.1	19.6	26.6	100	66.3	100
220	11.5	20.2	27.3	100	69.1	100
230	12.1	21	28.4	100	71.8	100
240	12.8	21.7	29.7	100	74.9	100
250	13.4	22.8	30.7	100	77.7	100
260	13.9	23.4	31.6	100	80.6	100
270	14.5	23.7	32.9	100	83.8	100
280	15.1	24.9	34.1	100	86.9	100
290	15.9	26.3	36.2	100	89.4	100
300	17	27.1	36.9	100	94.1	100

310	17.3	27.8	38.2	100	97.8	100
320	17.8	28.7	39.6	100	100	100
330	18.4	29.7	40.5	100	100	100
340	18.9	30.6	41.7	100	100	100
350	19.3	31.4	42.8	100	100	100
360	19.5	32.1	43.4	100	100	100
370	20.1	32.9	43.9	100	100	100
380	20.6	33.2	44.7	100	100	100
390	20.9	33.9	45.5	100	100	100
400	21.3	34.4	46.3	100	100	100
410	21.8	34.9	47.2	100	100	100
420	22.3	35.8	48.2	100	100	100
430	22.8	36.7	48.9	100	100	100
440	23.3	37.6	49.8	100	100	100
450	24.1	38.3	51.1	100	100	100
460	24.6	39.1	51.9	100	100	100
470	25.1	40.1	52.7	100	100	100
480	25.7	41.2	53.5	100	100	100
490	26.2	42.3	54.5	100	100	100
500	26.8	43.4	56.8	100	100	100
510	27.3	44.1	57.5	100	100	100
520	27.9	44.8	58.2	100	100	100
530	28.3	45.7	58.9	100	100	100
540	28.7	46.4	60.7	100	100	100
550	29.4	47	61.5	100	100	100
560	29.9	48.1	62.7	100	100	100
580	30.4	50.2	64.5	100	100	100
600	32.1	52.3	67.9	100	100	100
610	32.8	53.1	68.8	100	100	100
650	35.4	56.1	73.1	100	100	100
700	38.1	60.5	78.4	100	100	100
750	41.3	64.2	84.9	100	100	100
800	44.7	68.8	92.5	100	100	100
850	47.2	73.1	98.6	100	100	100
865	48.3	74.5	100	100	100	100
900	50.8	77.4	100	100	100	100
950	54.3	82.2	100	100	100	100
1000	57	86.9	100	100	100	100

1100	62.6	98.5	100	100	100	100
1130	64.5	100	100	100	100	100
1200	68.9	100	100	100	100	100
1300	73.5	100	100	100	100	100
1400	79.2	100	100	100	100	100
1500	84.7	100	100	100	100	100
1600	89.8	100	100	100	100	100
1700	94.3	100	100	100	100	100
1800	99.1	100	100	100	100	100
1810	100	100	100	100	100	100
1900	100	100	100	100	100	100

APPENDIX E

OPROFILE RESULTS

TABLE 3. OProfile Summary--UDP

	UDP			
CPS	90	300	600	1000
opensips	22.5	24.59	25.56	25.83
libc	21.84	21.25	21.16	21.35
tm	9.29	10.19	10.95	11.4

TABLE 4. OProfile Results Normalized--UDP

	UDP			
CPS	90	300	600	1000
opensips	1.08	4.1803	8.20476	14.7231
libc	1.04832	3.6125	6.79236	12.1695
tm	0.44592	1.7323	3.51495	6.498

TABLE 5. OProfile Summary--TCP Single Socket

	TCP Mono				
CPS	90	200	400	600	800
opensips	24.7	25.33	25.56	26.36	27.03
libc	17.23	16.91	21.16	18.02	17.47
tm	8.1	8.12	10.95	8.85	9.22

TABLE 6. OProfile Results Normalized--TCP Single Socket

	TCP Mono				
CPS	90	200	400	600	800
opensips	2.4947	4.91402	8.79264	13.78628	18.59664
libc	1.74023	3.28054	7.27904	9.42446	12.01936
tm	0.8181	1.57528	3.7668	4.62855	6.34336

TABLE 7. OProfile Summary--TCP Multiple Socket

	TCP Multi				
CPS	90	200	400	600	800
opensips	28.24	29.02	29.57	29.75	29.97
libc	14.9	14.82	14.98	15.31	17.03
tm	7.39	7.39	7.9	8.05	7.22

TABLE 8. OProfile Results Normalized--TCP Multiple Socket

	TCP Multi				
CPS	90	200	400	600	800
opensips	3.64296	7.28402	13.69091	20.20025	27.72225
libc	1.9221	3.71982	6.93574	10.39549	15.75275
tm	0.95331	1.85489	3.6577	5.46595	6.6785

TABLE 9. OProfile Summary--TLS Local Client

	TLS-Local-Client		
CPS	40	90	135
libcrypto	48.62	48.63	49.21
opensips	14	14.92	15.03
libssl	3.06	3.3	3.35
libc	7.74	7.94	7.93

TABLE 10. OProfile Results Normalized--TLS Local Client

	TLS-Local-Client		
CPS	40	90	135
Libcrypto	14.05118	31.51224	49.21
Opensips	4.046	9.66816	15.03
Libssl	0.88434	2.1384	3.35
Libc	2.23686	5.14512	7.93

TABLE 11. OProfile Summary--TLS-Local-Client-Session Reuse

	TLS-Local-Client-SessionReuse		
CPS	90	180	270
libcrypto	17.67	18.73	18.79
opensips	22.33	23.18	23.64
libssl	2.82	3.05	3.1
libc	11.12	11.1	11.18

TABLE 12. OProfile Results Normalized --TLS Local Client Session Reuse

	TLS-Local-Client-SessionReuse		
CPS	90	180	270
libcrypto	4.43517	10.73229	15.74602
opensips	5.60483	13.28214	19.81032
libssl	0.70782	1.74765	2.5978
libc	2.79112	6.3603	9.36884

TABLE 13. OProfile Summary--TLS Local Mutual

	TLS-Local-Mutual		
CPS	30	60	90
libcrypto	58.22	58.83	59.21
opensips	9.34	10.1	10.23
libssl	2.06	2.13	2.15
libc	4.74	4.91	4.96

TABLE 14. OProfile Results Normalized--TLS Local Mutual

	TLS-Local-Mutual		
CPS	30	60	90
libcrypto	16.76736	36.00396	54.23636
opensips	2.68992	6.1812	9.37068
libssl	0.59328	1.30356	1.9694
libc	1.36512	3.00492	4.54336

TABLE 15. OProfile Results Normalized--90cps Load

	90 cps					
TestCase	UDP	TCP Mono	TCP Multi	TLS-Local-Client	TLS-Local-Client-Session Reuse	TLS-Local-Mutual
libcrypto	0	0	0	31.51224	4.43517	54.23636
opensips	1.08	2.4947	3.64296	9.66816	5.60483	9.37068
Libssl	0	0	0	2.1384	0.70782	1.9694
Libc	1.04832	1.74023	1.9221	5.14512	2.79112	4.54336
Tm	0.44592	0.8181	0.95331	0	0	0
Kernel	2.22576	5.04697	6.38163	16.33608	11.56106	21.1138

Given below is the output of opreport showing the profiling information for the case of TLS-Local-Client with a CPS load of 90.

```
debian:/var# opreport --exclude-dependent --threshold=0.1
CPU: P4 / Xeon, speed 2793.37 MHz (estimated)
Counted GLOBAL_POWER_EVENTS events (time during which processor is not stopped) with a unit mask of 0x01 (mandatory)
count 100000
GLOBAL_POWER_E...|
  samples|      %|
------------------
   263995 48.6188 libcrypto.so.0.9.8
   111293 20.4963 no-vmlinux
    76032 14.0025 opensips
    42025  7.7396 libc-2.7.so
    16629  3.0625 libssl.so.0.9.8
    13488  2.4840 tm.so
     5088  0.9370 oprofiled
     2917  0.5372 syslogd
     1443  0.2658 acc.so
      898  0.1654 Xorg
      744  0.1370 libglib-2.0.so.0.1600.6
      725  0.1335 maxfwd.so
      669  0.1232 libpthread-2.7.so
      590  0.1087 libgobject-2.0.so.0.1600.6
      562  0.1035 rr.so

debian:/var#  opreport --demangle=smart --symbols /usr/local/sbin/opensips
CPU: P4 / Xeon, speed 2793.37 MHz (estimated)
Counted GLOBAL_POWER_EVENTS events (time during which processor is not stopped) with a unit mask of 0x01 (mandatory)
count 100000
samples  %        image name               symbol name
10340    13.2907  opensips                 fm_malloc
7045     9.0554   opensips                 tcpconn_new
4845     6.2276   opensips                 parse_via
4419     5.6800   opensips                 fm_free
4161     5.3484   opensips                 tcp_main_loop
3629     4.6646   opensips                 ser_malloc
3279     4.2147   opensips                 ser_free
2420     3.1106   opensips                 parse_to
2333     2.9988   opensips                 tcp_read_headers
1898     2.4396   opensips                 do_action
1621     2.0836   opensips                 fm_realloc
1360     1.7481   opensips                 receive_msg
1324     1.7018   opensips                 parse_headers
1266     1.6273   opensips                 tcp_send
1177     1.5129   opensips                 get_hdr_field
1168     1.5013   opensips                 parse_uri
1093     1.4049   opensips                 eval_expr
1081     1.3895   opensips                 process_lumps
1062     1.3651   opensips                 parse_first_line
990      1.2725   opensips                 forward_reply
822      1.0566   [vdso] (tgid:12018 range:0xb7f71000-0xb7f72000) (no symbols)
815      1.0476   opensips                 tcp_receive_loop
802      1.0309   opensips                 lumps_len
759      0.9756   opensips                 tcp_read_req
726      0.9332   opensips                 handle_io
697      0.8959   opensips                 parse_hname2
623      0.8008   opensips                 handle_ser_child
612      0.7866   opensips                 build_req_buf_from_sip_req
519      0.6671   opensips                 parse_cseq
510      0.6555   [vdso] (tgid:12022 range:0xb7f71000-0xb7f72000) (no symbols)
491      0.6311   opensips                 .plt
477      0.6131   opensips                 receive_fd
475      0.6105   opensips                 check_ip_address
413      0.5309   [vdso] (tgid:12019 range:0xb7f71000-0xb7f72000) (no symbols)
```

402	0.5167	opensips	tls_read
401	0.5154	opensips	branch_builder
389	0.5000	opensips	parse_msg
367	0.4717	opensips	handle_io
355	0.4563	opensips	build_res_buf_from_sip_req
335	0.4306	opensips	parse_method
330	0.4242	opensips	clean_hdr_field
323	0.4152	opensips	tls_blocking_write
321	0.4126	opensips	run_action_list
307	0.3946	opensips	tls_update_fd
288	0.3702	opensips	free_lump_list
268	0.3445	opensips	get_stat_var_from_num_code
268	0.3445	opensips	parse_content_length
267	0.3432	opensips	tls_fix_read_conn
261	0.3355	opensips	exec_pre_req_cb
259	0.3329	opensips	pv_printf
244	0.3136	opensips	get_ticks
239	0.3072	opensips	grep_sock_info
231	0.2969	opensips	free_sip_msg
230	0.2956	opensips	anchor_lump
214	0.2751	opensips	del_flaged_lumps
211	0.2712	opensips	eat_line
207	0.2661	opensips	core_hash
203	0.2609	opensips	send_all
186	0.2391	opensips	reset_avps
185	0.2378	opensips	via_builder
178	0.2288	opensips	tls_accept
175	0.2249	opensips	forward_request
173	0.2224	opensips	free_hdr_field_lst
169	0.2172	opensips	id_builder
168	0.2159	opensips	free_via_list
164	0.2108	opensips	build_res_buf_from_sip_res
161	0.2069	opensips	resetsflag
154	0.1979	opensips	get_send_socket
146	0.1877	opensips	set_avp_list
134	0.1722	opensips	exec_post_req_cb
134	0.1722	opensips	hostent_shm_cpy
134	0.1722	opensips	tcpconn_destroy
133	0.1710	opensips	str2s
130	0.1671	opensips	_shm_resize
123	0.1581	opensips	tls_tcpconn_init
121	0.1555	opensips	parse_from_header
112	0.1440	opensips	handle_tcp_child
104	0.1337	opensips	handle_new_connect
101	0.1298	opensips	run_top_route
95	0.1221	opensips	exec_pre_rpl_cb
85	0.1093	opensips	check_against_blacklist
85	0.1093	opensips	io_watch_del
85	0.1093	opensips	parse_sip_msg_uri
83	0.1067	opensips	received_test
80	0.1028	opensips	verify_callback
78	0.1003	opensips	send2child
77	0.0990	opensips	fixup_get_svalue
74	0.0951	opensips	free_to
74	0.0951	opensips	io_watch_add
72	0.0925	opensips	pv_get_spec_value
69	0.0887	opensips	get_branch
67	0.0861	opensips	set_ruri
65	0.0835	opensips	del_lump
62	0.0797	opensips	tcp_addr_hash
61	0.0784	opensips	error_text
61	0.0784	opensips	print_ip
61	0.0784	opensips	reset_bl_markers
59	0.0758	opensips	start_timer_processes
58	0.0746	opensips	send_fd
57	0.0733	opensips	init_err_info
57	0.0733	opensips	ser_realloc

```
55    0.0707 opensips            clear_branches
55    0.0707 opensips            insert_cond_lump_after
53    0.0681 opensips            insert_new_lump_after
52    0.0668 opensips            adjust_clen
51    0.0656 opensips            pv_get_xuri_attr
50    0.0643 opensips            check_transaction_quadruple
50    0.0643 opensips            tls_find_server_domain
48    0.0617 opensips            tcpconn_add
47    0.0604 opensips            pv_get_ruri
45    0.0578 opensips            insert_cond_lump_before
38    0.0488 opensips            insert_new_lump_before
37    0.0476 opensips            exec_post_rpl_cb
34    0.0437 opensips            insert_subst_lump_after
32    0.0411 opensips            getb0flags
31    0.0398 opensips            tls_dump_cert_info
27    0.0347 opensips            release_tcpconn
26    0.0334 opensips            check_self
25    0.0321 opensips            check_self_op
25    0.0321 opensips            tls_close
24    0.0308 opensips            init_sock_opt
22    0.0283 opensips            pv_get_ruri_attr
22    0.0283 opensips            tls_print_errstack
21    0.0270 opensips            tls_tcpconn_clean
20    0.0257 opensips            setflag
19    0.0244 opensips            free_cseq
19    0.0244 opensips            get_authorized_cred
11    0.0141 opensips            recv_all
9     0.0116 opensips            setb0flags
7     0.0090 [vdso] (tgid:12020 range:0xb7f71000-0xb7f72000) (no symbols)
6     0.0077 [vdso] (tgid:12016 range:0xb7f71000-0xb7f72000) (no symbols)
5     0.0064 [vdso] (tgid:12021 range:0xb7f71000-0xb7f72000) (no symbols)
4     0.0051 [vdso] (tgid:12015 range:0xb7f71000-0xb7f72000) (no symbols)
3     0.0039 opensips            setbflag
2     0.0026 opensips            delete_expired_routine
2     0.0026 opensips            set_ruri_q
```

REFERENCES

REFERENCES

[1] H. Schulzrinne, and J. Rosenberg, "A Comparison of SIP and H.323 for Internet Telephony," 1998; http://www.cs.columbia.edu/~hgs/papers/Schu9807_Comparison.pdf.

[2] EC. Cha. HK. Choi, and SJ. Cho, "Evaluation of Security Protocols for the Session Initiation Protocol," Proceedings of 16th International Conference on Computer Communications and Networks, August 2007; http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4317885.

[3] E. Nahum, J. Tracey, and C. Wright, "Evaluating SIP Proxy Server Performance," ACM, June 2007; http://www.research.ibm.com/people/n/nahum/papers/nossdav07-sip-perf.pdf.

[4] S. Salsano, L. Veltri, and D. Papalilo, " SIP Security Issues: The SIP Authentication Procedure and its Processing Load," IEEE Network, December 2002; http://ieeexplore.ieee.org/ xpl/freeabs_all.jsp?arnumber=1081764.

[5] C. Shen, E. Nahum, H. Schulzrinne, and C. Wright, "The Impact of TLS on SIP Server Performance," IPTComm 2010, 2-3 August, 2010 Munich, Germany; http://www.cs.columbia.edu/~hgs/papers/Shen1008_TLS.pdf.

[6] IETF Network Working Group, "SIP: Session Initiation Protocol," 2002; http://www. ietf.org/rfc/rfc3261.txt.

[7] M. Collier, "Basic Vulnerability Issues for SIP Security," March 2005; http://download.securelogix.com/library/SIP_Security030105.pdf.

[8] A. Steffen, D. Kaufmann, and A. Stricker, "SIP Security," 2004; http://security.hsr.ch/docs/DFN_SIP.pdf.

[9] Dialogic, Inc., "SIP signaling over TLS," 2010; http://excelsupport.dialogic.com/imgpubs/webhelp/sip_over_tls_ov.htm.

[10] Microsoft, Inc., "Understanding Internet Protocol Security," January 2005; http://technet.microsoft. com/en-us/library/cc739674(WS.10).aspx.

[11] Sipp.sourceforge.net, "SIPp open source test tool," October 2010; http://sipp.sourceforge.net.

[12] H. Schulzrinne, S. Narayanan, J. Lennox, and M. Doyle, "SIPstone - Benchmarking SIP Server Performance," April 2002; http://www.cs.columbia.edu/~library/TR-repository/reports/reports-2002/cucs-005-02.pdf.

[13] Opensips.org, "OpenSIPS Server," April 2012; http://opensips.org.

[14] Opensips.svn.sourceforge.net, "OpenSIPS Installation Notes," March 2012; http:// www.opensips.org/Resources/Install.

[15] Opensips.org, "OpenSIPS - TLS Support," March 2012; http://www.opensips.org/html/docs/tutorials/tls-1.4.x.html.

[16] Oprofile.sourceforge.net, "OProfile – A System Profiler for Linux," March 2012; http://oprofile.sourceforge.net.

[17] Oprofile.sourceforge.net , "OProfile Manual – Getting Started" March 2012; http://oprofile.sourceforge.net/doc/ overview.html#getting-started.