Bare PC SIP User Agent Implementation and Performance for Secure VoIP

Roman Yasinovskyy, Andre L. Alexander, Alexander L. Wijesinha, and Ramesh K. Karne
Department of Computer & Information Sciences
Towson University
Towson, MD 21252
USA
e-mail: {ryasinovskyy, aalexander, awijesinha, rkarne}@towson.edu

Abstract—**Bare PC systems, which run applications without using any operating system (OS) or kernel, are immune to attacks targeting a specific OS. They also perform better than conventional systems due to their reduced overhead. We describe the design, implementation and performance of a SIP user agent (UA) for secure VoIP on a bare PC system. In particular, we discuss SIP functions and message handling, and CPU tasking. We also give details of the UA design and code that enable a lean implementation of SIP to be intertwined with the network protocols needed for secure VoIP on a bare PC softphone. The interoperability of the bare PC SIP UA is verified by conducting tests using OS-based as well as bare PC SIP servers and UAs. We also study bare PC SIP UA performance by comparing timings for key SIP UA operations for the bare PC softphone with timings for a compatible Linux softphone. The results show that processing times for the SIP register and invite operations for the bare PC SIP softphone are significantly less than the corresponding times for the Linux softphone regardless of whether a bare PC or a Linux-based SIP server is used. Finally, we propose a simple security extension to SIP authentication that enables the session key exchange for media protection to be encrypted without incurring the overhead of TLS or IPsec. Bare PC SIP softphones can be used for building secure and efficient VoIP systems that do not require any OS support.**

Keywords-bare PC; SIP implementation; SIP performance; SIP user agent; VoIP; VoIP security.

I. INTRODUCTION

The session initiation protocol (SIP) is a general-purpose protocol that can be used for video conferencing, instant messaging and gaming. However, its primary use today is in VoIP systems, where it serves as a support protocol for registering and locating users, and for call set up and management. Conventional SIP implementations in servers and softphones require the support of an operating system (OS) such as Windows or Linux, or some form of an OS kernel. SIP phones are also frequently implemented in hardware/firmware typically with an embedded OS. The SIP implementations in OS-based systems take advantage of their rich supporting environment and capabilities, and are convenient to use.

Bare PC systems (also known as bare machine computing systems) enable self-supporting applications to run directly on the hardware of an ordinary PC (desktop or laptop) without using an OS or kernel. A bare PC provides immunity against OS-based attacks, and bare PC applications perform better than applications running on conventional systems due to the elimination of OS overhead.

We describe the design, implementation and performance of a SIP user agent (UA) for secure VOIP on bare PC systems. The SIP UA is integrated with the bare PC softphone application. In [1], the design and implementation of a SIP server and UA for VoIP on a bare PC were described. However, that study did not consider UA performance or the elimination of additional protocols such as TLS or IPsec to secure the key exchange over SIP to protect the VoIP call. Other studies [2], [3] dealt extensively with bare PC SIP server (but not SIP UA) implementation and performance. This paper extends [1] by 1) providing software design and code details for the UA to illustrate protocol intertwining; 2) presenting performance data in the form of timings for key SIP operations on the UA that are compared with timings on a compatible Linux-based SIP softphone; and 3) proposing a simple security extension to SIP/SDES authentication that enables the key exchange for media protection (via SRTP) to be encrypted using AES without incurring the overhead of any additional protocols.

The design details and code snippets for the bare PC SIP UA provided here help in understanding how the lean implementation of SIP is intertwined with the other necessary network protocols in the self-supporting bare PC environment with no OS or kernel resulting in improved performance of the UA compared to OS-based SIP UAs. This paper only focuses on the implementation, performance, and security aspects of the bare PC SIP UA since bare PC SIP server implementation and performance are discussed in detail in [3].

In secure environments, an OS-based full SIP implementation may not be needed and a lean implementation of SIP on a bare PC is useful due to the low overhead and inherent immunity of the application against attacks targeting vulnerabilities in a typical OS such as Linux or Windows. Compared to their OS-based counterparts, bare PC systems also have reduced code complexity and code size, making it easier to analyze the code and fix security

2012, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

flaws. Moreover, bare PC systems have fewer avenues open for attackers to exploit since they only provide essential functionality and services. The low overhead and performance improvements of bare PC applications compared to applications on conventional systems are due to using direct interfaces to the hardware, fewer context switches, application-specific optimizations, efficient inter-layer communication, and streamlined versions of the necessary protocols and drivers.

As with other bare PC applications, the SIP UA implementation and interfaces to the hardware constitute a single self-supporting executable, implements only the essential elements of SIP and is UDP-based. The bare PC SIP UA and softphone application currently runs on an IA32 (Intel Architecture 32-bit) or Intel 64-bit architecture in 32-bit mode.

The rest of this paper is organized as follows. In Section II, we briefly survey related work. In Section III, we give an overview of bare machine computing. In Section IV, we describe the bare PC SIP UA software design and operations. In Section V, we give details of the UA implementation and provide some code snippets. In Section V, we describe testing scenarios, and in Section VI, we give UA performance data for bare PC and Linux-based SIP UAs. In Section VIII, we propose a modification to SIP authentication for protecting session key exchange for media protection, and in Section IX, we present the conclusion.

II. RELATED WORK

There are numerous implementations of conventional SIP servers and SIP softphones on various OS platforms. These SIP servers and UAs run on conventional OSs. In [4], a SIP server is implemented on top of an existing SIP stack. In [5], SIP servers and SIP UAs are implemented on the Solaris 8 OS. A client-side SIP service offered to all applications based on a low-level SIP API is described in [6]. In [7], the features of a new language StratoSIP for programming UAs that can act respectively as a UA server to one endpoint and as a UA client to another are presented. In [8], the UA is a SIP-based collaborative tool implemented by using existing SIP and SDP stacks. In [9], a Java-based SIP UA is proposed for monitoring manufacturing systems over the Internet. The focus of [10] is a SIP adaptor for both traditional SIP telephony and user lookup on a P2P network that does not have a SIP server. The goal of such SIP servers and SIP UAs is to offer enhanced services to clients by using existing low-level SIP stacks that rely on an OS. In contrast, bare PC SIP servers and UAs that are implemented directly on the hardware will have less overhead and are more suited for secure low-cost environments.

Intertwining bare PC Web server or email server application and application protocols (i.e., HTTP or SMTP) with the TCP protocol contributes to its improved performance over OS-based servers [11], [12]. In [2] and [3], the performance of a bare PC SIP server is compared with that of OS-based servers, and it is shown that the bare PC SIP server performs better except in a few cases. The SIP server performance studies did not discuss the SIP UA design details or its implementation. The design, implementation, and performance of a bare PC softphone for peer-to-peer communication are discussed in [13] and [14]. However, the softphone does not support SIP and does not incorporate a UA of any kind; hence it lacks the ability to communicate with SIP servers and other SIP softphones. Details of the SIP protocol itself are given in [15].

III. BARE MACHINE COMPUTING

Bare PC application development is based on the bare machine computing paradigm, also referred to as the dispersed operating system (DOSC) paradigm [16]. In this paradigm, a single self-supporting application object (AO) encapsulating all of the necessary functionality for a few (typically one or two) applications executes on the hardware without an OS. Bare machine applications only use real memory; a hard disk is not used. The AO, which is loaded from a USB flash drive or other portable storage medium, includes the application and boot code.

The application code is intertwined with lean implementations of the necessary network and security protocols. If required by the application, the AO also includes cryptographic algorithms, as well as network interface and other device drivers, such as an audio driver in case of the bare PC softphone. The interfaces enabling the application to communicate with the hardware [17] are also included in the AO. The AO code is written in C++ with the exception of some low-level assembler code. The AO itself manages the resources in a bare machine including the CPU and memory. For example, every bare PC AO has a main task that runs whenever no other task is running, and network applications require a Receive (Rcv) task that handles incoming packets. Additional tasks may be used depending on the applications included in the AO, such as an audio task for the bare PC softphone.

IV. BARE PC SIP UA SOFTWARE DESIGN

The software design of the bare PC SIP UA is simple and modular. It manages data associated with log in, dialing, incoming and connected calls, media IP address and port, and STUN state [18]. The main data structures in the SIPUA object are phone_book, media, call_session, configuration, call_log, and via_headers. The call_log struct with some of its fields is shown in Figure 1. For each char array, an int variable is used to store its actual size. For example, int call_id_size stores the size of the char array call_id.

In addition, the SIPUA object defines numerous constants, variables, and arrays. These are used to store UA timers, counters, tags, authentication data, menu controls, and user account information. As in all bare PC applications, a TCB table [3], [11] is used to store information such as addresses and ports needed to process incoming packets.

The SIP UA code and methods are designed based on the various states the UA can be in. Since there is no OS or conventional protocol stack, the self-supporting SIP UA application itself sends and responds to messages using the necessary protocols. This requires that the UA manage memory, CPU and task scheduling without any OS support. All memory is real and mapped statically, i.e., there is no provision to load any modules dynamically at run-time,

2012, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

which significantly increases security of the SIP UA and prevents attackers from gaining access to other parts of memory or execute code other than the legitimate SIP application itself.

```
struct call_log
{
   int phone_state;
   int app_state;
   char call_id[70];
   char cseq[5];
   char password[20];
   char account_name[20];
   char account_domain[20];
   char account_ip[16];
   char account_port[6];
   char media_ip[16];
   char media_port[6];
   char call_tag[100];
   char callee[40];
   char callee_sip_ip_text[16];
   unsigned int callee_sip_port_hex;
   unsigned int callee_media_port_hex;
   int srtp; int srtp_match;
   int originator; struct configuration config;
   …
 };
```

Figure 1. Example of a UA Data Structure

Excluding methods for booting and loading the UA code from a USB flash drive (or other portable media), other “low-level” methods that are common to all bare PC applications, and the method that runs the Main task, the remaining methods of the SIP UA consist of those that initialize the UA, display menus and call state, and interact with the user via keyboard, and methods that implement key SIP functionality. The primary user data and SIP state are contained in char arrays and pointers, and the method parameters enable the data to be accessed and manipulated based on SIP messages that are received including REGISTER, INVITE, and BYE. Authentication is enabled by default, but can be turned off by the user prior to making a call.

In Figure 2, SIP UA method declarations appearing in the SIPUA.h file and their parameters are shown. It can be seen that method parameters reference user account, and IP address and port number information, as well as SIP/SDP headers and tags in incoming and outgoing messages. In addition, the local data structures referred to earlier are directly manipulated by these methods. The methods instantiate other objects such as DHCP, STUN, and SRTP [19] together with the usual RTP, UDP, IP, and Ethernet objects used for network communication. Keyboard input and screen output is done via special bare PC interfaces that are use by all bare PC applications. Although the bare PC SIP implementation is lean, it requires strict adherence to the SIP protocol specifications given in [15] in order to communicate with both OS-based and bare PC SIP servers and UAs. From a security viewpoint, the code is simpler and there are no hidden dependencies on external libraries or code other than that of the SIP UA application itself. The elimination of OS or kernel code and external dependencies enables only functionality that is essential to the SIP application to be implemented, which simplifies code analysis and detection of vulnerabilities.

Figure 3 shows the call flow relationships among the key methods in the UA. The method parameters and return types are omitted since they were specified in Figure 2. It can be seen that several methods in the UA are also used by the SIP server [3]. These include siphandler, sipsenddata, format_sip_response, parse_headers, copy_tag_line, create_packet, copy_sip_request, create_response_packet, and generate_sip_response, and several methods that are used to parse tags. The main difference between the SIP server code and the UA code is that the server also contains code to manage the database storing registration and authentication information for users. In contrast, the UA contains code to process SIP authentication (i.e., handle the SDP challenge/response messages), secure the VoIP media stream via SRTP, and provide the user interface via menus and keyboard input.

When the UA is booted, sipuainit and sipuser_init are used to initialize the UA and store user information respectively; generate_sip_response sets up memory for the packet to be sent and calls sipsenddata; sipsenddata in turn calls format_sip_response to construct the SIP message, and then calls the relevant methods in the UDP, IP, and Ethernet objects to construct the necessary protocol headers and send the packet; format_sip_response constructs the SIP message in the following cases: received INVITE, TRYING, RINGING, ACK, BYE, OK, UNAUTHORIZED, or PROXY_AUTH; or sent INVITE, RINGING, REGISTER, ACK, BYE, DECLINE, BUSY, or REGISTER_LOGOUT.

As shown in Figure 3, when a packet is received, siphandler calls parse_headers to determine the type of the received SIP message, and then calls handle_session to retrieve or store state information, followed by generate_sip_response to construct the response; parse_headers in turn calls several methods to parse tags including copy_tag_line, parse_from_to_tag, and parse_call_id_tag; copy_sip_request, create packet, and create_response_packet are used to create SIP packets; parse_authenticate is used to parse an authentication challenge and compute an authentication response; route_packet method is used to determine whether the packet can be delivered directly or whether it needs to be sent to the default gateway; and shout_or_route is called directly from the main task for start-up, displaying menus, registering the user, and initiating calls. SIPUA also includes methods for displaying menus and submenus, switching screens, and quick dialing.

After shout_or_route is called as above and the user is registered, the user can initiate a new call. Incoming calls are handled as follows. When the Rcv task receives a UDP message containing a SIP Invite, the main task actives a SIP task. Then siphandler is invoked and the SIP responses (TRYING and RINGING) are sent by using the relevant methods in the SIPUA application as described above. When a SIP response message such as an ACK or 200 OK is received, the TCB entry is used to retrieve the information needed to process the message.

2012, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

Figure 4 shows the relationship between protocols and the Main, Rcv and SIP tasks in the SIP UA. Tasks handling audio processing [14] are not shown here. All upper layer protocols use UDP/IP/Ethernet and SRTP uses RTP. Incoming UDP packets are handled by the Rcv task regardless of the upper layer protocol. The SIP task handles SIP processing that is initiated by siphandler, and passes control back to the main task after sipsenddata terminates.

```
unsigned short get_destsipport (char *data_array, int size);
void sipuainit();
int route_packet(char *source_ip, char *dest_ip, char *subnet_mask, char *dest_mac);
void shout_or_route();
void siphandler(char *packet_array, int packet_size);
int create_packet (char *cat_array, char *data_array, int datasize);
int parse_headers (char *data_array, int size_array);
int parse_call_id_tag (char *data_array, char *tag_line, int start);
int parse_from_to_tag (char *data_array, int start, int end, char *account_name, int account_name_size, char *account_domain, int account_domain_size);
void generate_sip_response(char *data_array, int response);
void sipsenddata(char *cptrl, long InPtr, char *data_array, int response);
int format_sip_response(char *send_buffer, char *data_array, int response);
int create_response_packet (char *cat_array, char *data_array, int datasize);
int copy_sip_request (char *cat_array, char *data_array, int new_data_size, int datasize, int enddatasize);
int copy_tag_line (char *data_array, char *tag_line, int start, int end);
int get_ip(char *data_array, int start, int end);
char parse_sdp_ip(char *ip_address_string, int size, int octet);
void parse_sdp_media(char *data_array, int start, int end);
void parse_sdp_attribute(char *data_array, int start, int end);
int parse_crypto_tag (char  *data_array, int start);
int parse_encrypt_tag (char  *data_array, int start);
int text_ip(char  *data_array, int end, char * hex_ip);
int handle_session(char *address_name, int address_name_size, char *address_domain, int account_domain_size, int type);
int int_text(int i, char  *port_string);
int check_media();
void init_sdp_media();
void reset_values();
void sipuser_init();
void sipserver_init(int flag);
void parse_authenticate(char *data_array, int start, int end);
int parse_cseq_tag (char  *data_array, int start, int end);
int parse_expires_tag (char *data_array, int start, int end);
int parse_contact_tag (char *data_array, int start, int end);
void clear_all_menus(int type);
void header_label();
void footer_label(int type);
void middle_label();
void call_connected_menu();
void incoming_call_menu();
void selected_item(int type);
void button_press();
int wait_timer(int type);
void switch_screen();
void main_start_menu();
void sub_main_menu();
void sub_quick_dial_menu(int type);
void sub_phone_config_menu();
void sub_stun_menu();
void sub_ip_config_menu();
```

Figure 2. SIP UA methods and parameters

V. Bare PC SIP UA Implementation

The bare PC SIP user agent (UA) is integrated with the bare PC softphone enabling calls to be set up. Its operational characteristics are similar to those of a SIP UA in a conventional OS-based SIP softphone. However, the UA implementation is different due to the absence of an OS and a built-in protocol stack, and results in a UA with less overhead and better security. The UA can also directly communicate with a peer on a local network (without using a SIP server).

A. *UA Operation/User Interface*

As in the case of the bare PC SIP server, only two tasks Main and Rcv are needed for the UA, and arriving SIP messages and responses are processed in a single thread of execution as described earlier. When the UA is booted, if an IP address for the UA has not been preconfigured, the UA sends out a request for an IP address and obtains one using DHCP. If this is a private address, the UA is behind a NAT and uses STUN [18] to learn its public IP address and port. In this case, the UA first sends a DNS request and obtains the IP address of a public STUN server. The bare PC STUN implementation is described in more detail below.

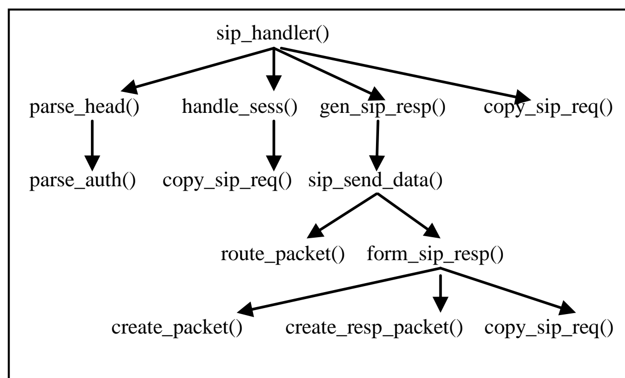

Figure 3. UA call flow relationships

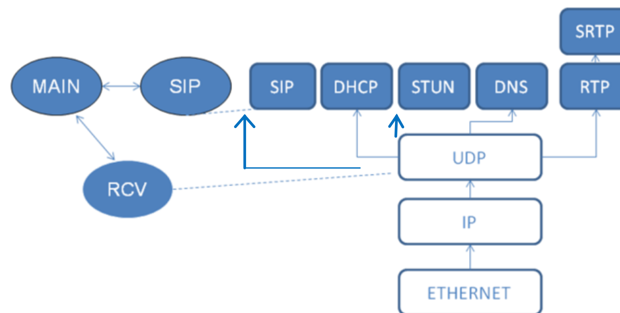

Figure 4. UA protocol/task relationships

After UA completes the initialization process it displays the main login menu, which enables the user to login-in to a particular SIP server or to communicate directly with a peer as noted earlier. In case SIP server login is selected, the UA sends a SIP Register request to the server after performing a DNS resolution if needed. Once the 200 OK messages are

2012, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

received from the SIP server, the UA displays a "main menu" screen as in Figure 5. The menu has several options, which enables the user to see the IP configuration information from DHCP, and NAT mappings from STUN that show the external IP address and internal/external SIP and RTP ports for the softphone. Such information is useful to troubleshoot connectivity problems. In addition, a separate option shows call status and connectivity information, and indicates whether security is on. A "quick dial" option for selecting specific users is also available.

Figure 5. UA Main Menu Screen.

The essential UA functionality contained in the SIPUA object consists of about 3000 lines of C++ code. This object is supplemented by 1) objects for cryptographic and other algorithms needed for key establishment (HMAC, SHA-1, MD5, AES, and Base64); 2) objects implementing the essential elements of the necessary auxiliary protocols (STUN, DHCP, and DNS); and 3) objects needed by the bare PC softphone including the Ethernet, IP, and UDP objects, the RTP, audio, and G.711 objects that handle voice data processing, recording, and playback on the bare PC softphone, and the SRTP object [19] that provides VoIP media security. The SRTP protection is optional and can be turned off. However, the additional overhead due to using SRTP is low [20].

B. *User Agent Client and User Agent Server*

The bare UA consists of two independent components: the SIP user agent server (UAS) and SIP user agent client (UAC). The UAS is operationally similar to the bare PC SIP server with respect to its handling of SIP packets. For example, it listens for call requests and its actions are activated by the Rcv task when a packet arrives as discussed earlier for the case of the SIP server. The UAC can be activated by keyboard input. The UA functionality is contained in a SIPUA object that is responsible for processing SIP messages and SDP tags, displaying the SIP UA interface, and interacting with the user. The SIPUA object is integrated in a single AO with several other objects needed to implement the UA.

C. *STUN/DHCP/DNS/SRTP*

The public IP address and port learned from the public STUN server is used in SIP Invite requests to enable the peer to communicate with the UA behind the NAT. The bare PC SIP UA sends out multiple STUN messages to find the external port for its voice channel over RTP. Since the signaling channel is proxied through the SIP server, STUN is not needed to discover the external SIP signaling port. After the bare PC client is booted, STUN messages for the media channel are sent every 30 seconds until the SIP UA establishes the call. The Invite message contains the last known media channel external port number. Since the NAT binding may change, the UA sends voice packets to the destination host using a sequence of consecutive ports. The UA stops sending on the other ports once voice packets are received on a particular port.

The bare PC SIP UA also needs to send DHCP messages to automatically obtain an IP address and other essential configuration information at start-up. Since there is no OS and no built-in protocol stack on the bare PC softphone, a lean implementation of DHCP is used. The DHCP messages follow the typical DHCP call flow (Discover, Offer, Request, and Ack). The softphone can also send DNS requests to resolve the domain name of the SIP or STUN server. As noted earlier, the implementation of the DHCP and DNS protocols have only the minimal features needed by the bare PC SIP softphone.

The bare PC SIP UA is also integrated with SRTP. The implementation and performance of SRTP on a bare PC softphone are presented in [20]. SRTP allows the UA to communicate securely with conventional SIP UAs that are SRTP capable. The bare PC softphone AO includes implementations of SHA-1, MD5, HMAC, and AES in counter mode, which are used by SRTP. The bare PC SRTP implementation also supports addition of a recommended authentication tag to the end of the RTP packet. The UA currently implements the SDP Offer/Answer model via SDES for key exchange. This method is used by several conventional SRTP clients. The keys used to generate the session keys are Base64 encoded by the bare PC softphone prior to transmission. Since this approach for transmitting keys is not secure, a more secure alternative is described in Section VIII.

D. *Protocol Intertwining*

An essential component of the bare PC SIP UA application is code that intertwines the necessary network protocols with the SIP UA application code. Since there is no protocol stack and a distinction between user space and kernel space as in a conventional OS-based system, it is necessary to provide methods that can directly instantiate objects of other protocols and invoke their methods. To illustrate this idea, we consider two methods in the SIPUA object: generate_sip_response and sipsenddata in Figs. 6 and 7. Comments and code lines unessential to the present discussion are not shown.

In Figure 6, the SIPUA object instantiates the bare PC Ethernet object EtherObj. This enables the UA to access the Ethernet buffer directly, which significantly reduces overhead. It can be seen that addresses and memory in the bare PC are directly manipulated by the SIPUA. Such "hardcoding" makes the SIP UA code more secure: it is more difficult for an attacker to perform standard buffer manipulations since different bare PC SIPUAs can use a

2012, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

different memory layout. The last action of the generate_sip_response method is to invoke sipsenddata.

In Figure 7, the code for the sipsenddata method is shown. This code illustrates how the SIP UA directly instantiates the necessary protocol and "lower layer" objects: DHCP, UDP, IP, and Ethernet. The method calls route_packet, which calls ARP to find the MAC address, then calls format_sip_response to create the correct SIP response (depending on the current state determined by a received SIP message such as an INVITE), and finally sends the data via a call to UDP. Again, it can be seen that header sizes are "hardcoded", which improves security, and that direct invocation of the necessary protocols eliminates the overhead of interlayer communication and context switching that would be present in a conventional OS-based system.

VI. TESTING

Operational tests of the bare PC SIP server and SIP softphone implementations with and without authentication and SRTP security were conducted using Dell GX-260 desktops with Intel Pentium 4 2.4 GHz processors, 1.0 GB RAM, and a 3COM Ethernet 10/100 PCI network card. The test network consists of a dedicated LAN within the Towson University network, and an external network connected through an ISP as shown in Figure 8. The bare PC SIP server and user agents were first tested within the dedicated LAN. Testing was performed to verify 1) correct operation between the bare PC SIP server and bare PC SIP softphones; 2) interoperability of bare PC SIP softphones with the OpenSer v3.0.0 server [21]; 3) interoperability of the bare PC SIP server with Linphone 2.1.1 [22] and Snom360-5.3 softphones [23]; and 4) interoperability of bare PC SIP softphones with the Linphone and Snom softphones.

```
void SIPUAObj::generate_sip_response(char *data_array, int response)
{
EtherObj EO;
char *sip_send_buffer;
long *p1;
long cb;
char ca;
long InPtr = 0;
long x = 0;

p1 = &cb;
sip_send_buffer = &ca;

x = EO.DownListPointer + EO.SendInPtr * 32 + 8 -   ADDR_OFFSET;

p1 = (long*)x;
sip_send_buffer = (char*) *p1;

InPtr = EO.SendInPtr;
EO.SendInPtr++;
if (EO.SendInPtr == EO.SndLstSize)
{
  EO.SendInPtr = 0;
}
sip_send_buffer = sip_send_buffer + 14 + 20 + 8 - ADDR_OFFSET;

sipsenddata(sip_send_buffer, InPtr, data_array, response);
}
```

Figure 6. Code to generate a SIP response

```
void SIPUAObj::sipsenddata(char *cptr1, long InPtr, char *data_array, int response)
{
static int jj = 0;
EtherObj EO;
UDPObj udp;
DHCPObj dhcp;
IPObj ip;
char *send_buffer;
char c1;
send_buffer = &c1;
int retcode = 0;
int SIPPack_size=0;

//call route packet to get proper mac
route_packet(dhcp.ip_data.ip_address, SIPDestIP, dhcp.ip_data.subnet, SIPDestMac);

send_buffer = cptr1;

//format_sip_response creates the SIP response packet
SIPPack_size = format_sip_response(send_buffer, data_array, response);

send_buffer = send_buffer - 8; //8 byte UDP header
retcode = udp.FormatDHCPUDPPacket(send_buffer, SIPPack_size, udp.SIPSourcePort, udp.SIPDestPort, 0);
SIPPack_size = SIPPack_size + 8;
send_buffer = send_buffer - 20; //20 byte IP header
retcode = ip.FormatIPPacket(send_buffer, SIPPack_size, SIPDestIP, SIPDestMac, UDP_Protocol, 0);
SIPPack_size = SIPPack_size+20;
send_buffer = send_buffer - 14; //14  byte ethernet header
retcode = EO.FormatEthPacket(send_buffer, SIPPack_size, IP_TYPE , SIPDestMac, InPtr, 0x04, 0);
SIPPack_size = SIPPack_size+14;
}
```

Figure 7. Code to send a SIP message

Similar tests were conducted over the Internet by establishing calls between a softphone on the external network and another on the dedicated LAN when the SIP servers are connected to the LAN. These tests also served toverify that the UA and the lean DHCP, STUN, and DNS implementations on the bare PC SIP softphone work correctly when it is connected to the Internet. In particular, the bare PC STUN implementation was found to be adequate for connecting between clients behind NATs on the dedicated test LAN and on an ISP network.

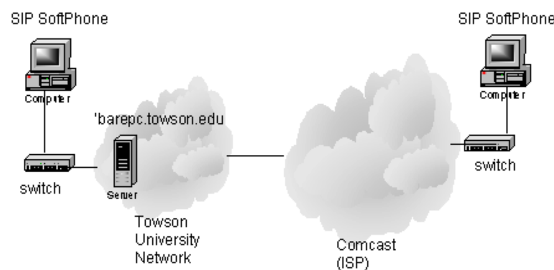

Figure 8. Test Network.

2012, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

VII. SIP UA PERFORMANCE

To evaluate performance of the SIP UA, experiments were conducted using a 100 Mbps Ethernet test LAN consisting of two SIP UA client machines and a SIP server. All machines were Dell GX260 with Intel Pentium 4 (2.4 GHz), 1.0 GB RAM, and 3COM Ethernet 10/100 PCI network cards. The OS-based (non-bare) machines ran Linux CentOS 5.6 (2.6.18). The non-bare SIP server and SIP UA used were OpenSER 1.3.4 [21] and Linphone 2.1.1 [22] respectively.

Times to complete each of the four SIP message exchanges corresponding to REGISTER, INVITE, SESSION OK and BYE as shown in Figure 9 were measured by capturing the relevant packets using Wireshark 1.5.0 [24]. Each exchange was repeated several times to ensure that the measurements were stable.

The results are shown in Figs. 10-12. In Figs. 10 and 11, Linux SIP UAs are connected to a bare PC or Linux SIP server. The times for completing REGISTER in each case are seen to be highest compared to the other SIP exchanges. In Figs. 12 and 13, bare PC SIP UAs are connected to a bare PC or Linux SIP server. The times to complete all exchanges except for SESSION are much smaller for the bare PC SIP UA than for the Linux SIP UA. For both UAs, it does not make much difference whether a Linux or bare PC SIP server is used. For the REGISTER and INVITE exchanges in particular, it can be seen that the time for the bare PC SIP UA is less than 1 ms, whereas the Linux SIP UA averages 67 ms and 15 ms respectively. As expected, the BYE exchange takes negligible time for both SIP UAs. The excessive time taken by the bare PC SIP UA for the SESSION OK exchange (compared to the Linux SIP UA) is due to processing the media stream prior to sending the ACK. This time could be reduced by only processing SIP messages in this case as done by the Linux SIP UA.

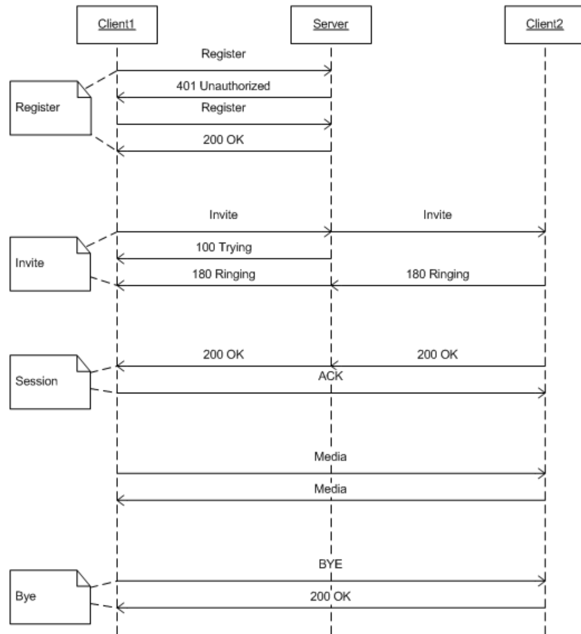

Figure 9. SIP message exchanges

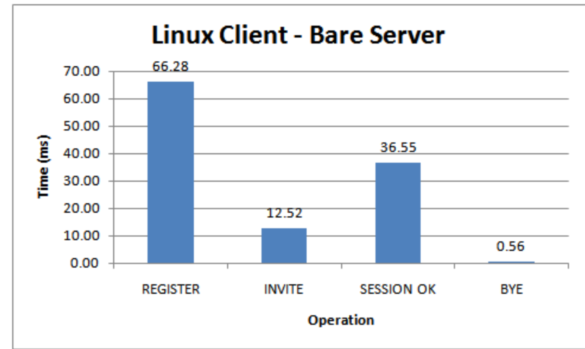

Figure 10. SIP message processing time (Linux client/bare server)

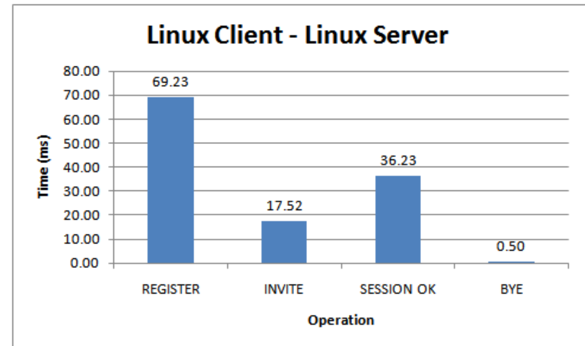

Figure 11. SIP message processing time (Linux client/Linux server)

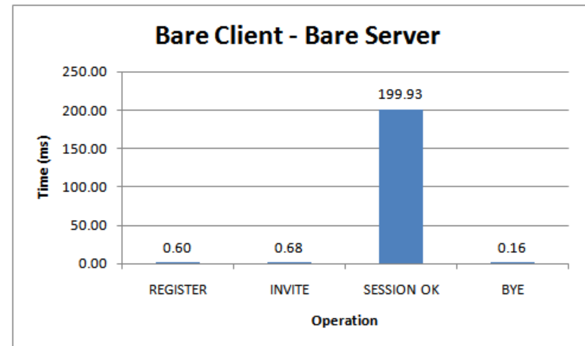

Figure 12. SIP message processing time (bare client/bare server)

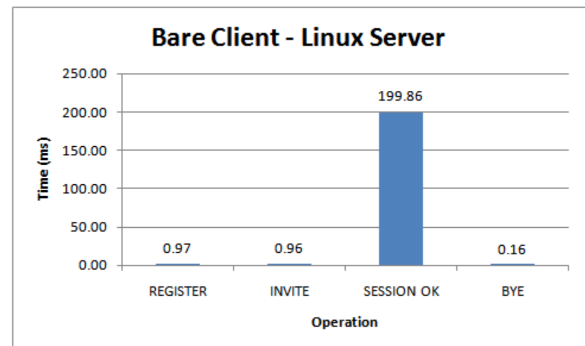

Figure 13. SIP message processing time (bare client/Linux server)

2012, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

VIII. SECURE KEY EXCHANGE

Security is an important aspect of VoIP. First, the caller and callee must be authenticated; and second, the voice packets must be encrypted and their integrity needs to be ensured. SRTP provides a standardized convenient method to achieve these requirements. Since SRTP does not mandate a specific key exchange method, a variety of methods are used in practice. A common approach used by many softphones is to use TLS to protect the SRTP key exchange. However, TLS key establishment is expensive and the handshake requires TCP. This means significant overhead and delay may be incurred prior to the beginning of a call before voice packets secured by SRTP can be exchanged. A low overhead alternative is to send Base64-encoded SIP/SDES messages to exchange the SRTP keys [20]. Unfortunately, this method is not useful from a security viewpoint. We describe below a simple modification using SIP/SDES that significantly enhances security without incurring the penalty of TLS.

The proposed key exchange technique is an adaptation of a technique that is frequently used for establishing a shared encryption key between clients when a trusted key server is available. We assume that the local SIP servers S1 and S2 associated with the caller and callee are trusted, and that S1 and S2 pre-share secrets s1 and s2 respectively with the caller and callee. For convenience, we also assume that these secrets are the respective passwords used for SIP registration and authentication by the caller and callee (or derived from the passwords), and that the caller and callee share the same SIP server S i.e., S1=S2=S. The technique is easily extended when the caller and callee have different SIP servers S1 and S2 (provided that S1 and S2 have a secure pre-established channel or an IPsec tunnel between them to encrypt the payload of any messages exchanged between them).

At most one new message is required when using this technique to secure the SRTP key exchange. It works as follows. Whenever a SIPUA initiates a call with authentication, the usual challenge/response technique requires that a nonce be generated by the SIP server and sent to the SIPUA in order to verify that the SIPUA has the correct password/secret (see Section IV). If two nonces instead of one are sent by the SIP server, the second nonce can be used as a seed for the TLS PRF (pseudorandom function) [25] in order for the SIP server and SIPUA to generate a common AES encryption key of length 128, 192 or 256 bits for encrypting SIP/SDES messages. The TLS PRF uses the following data expansion function:

P_hash(secret, seed) = HMAC(secret, A(1)||seed) || HMAC(secret, A(2)||seed) ||…

Here || denotes concatenation. HMAC can use any one-way hash function such as MD5, SHA-2 or SHA-3, and A(i)=HMAC(secret, A(i-1)) for $i \geq 1$ with A(0)=seed. For a 128-bit AES key, the TLS PRF needs to be iterated only once with either an MD5 or SHA-2 hash; for a 256-bit key, at most two iterations are needed depending on the hash size. This technique enables the SIP server and the SIPUA at the caller and callee respectively to compute common AES keys k1 and k2 of desired lengths. The SIP server can now exchange messages with the caller and callee that are encrypted with the keys k1 and k2 respectively. In particular, the server can send a message encrypted with the key k2 that transfers the key k1 to the callee. The key k1 can then be used by the caller and callee to encrypt the SIP/SDES SRTP key exchange over UDP/IP without incurring the overhead of a TCP connection or an expensive TLS handshake. Since implicit authentication based on knowledge of a pre-shared key is used instead of verifying a certificate as in TLS, the proposed technique trades off reduced security for less overhead.

In effect, the single message used to authenticate a SIPUA also serves to transfer the extra nonce for generating an AES key. The only additional cost is the cost to transfer an extra nonce within this message, which should be insignificant, and the cost of the extra message to the callee to transfer the key k1. For each call, a different AES key is generated since the nonce will be different. If the caller and callee have different SIP servers S1 and S2, it will be necessary for S1 to securely transfer the key k1 to S2 so that it can be relayed to S2. This can be done via IPsec or TLS as usual, or by use of preshared keys. In a local network, two bare PC SIP UAs can directly establish a secure VoIP call after authentication without using a SIP server. In this case, authentication is based on a preshared secret, and the preceding technique can be used for the caller and callee to generate a common AES key to encrypt the SRTP key exchange between them.

IX. CONCLUSION

We described the design, implementation, and performance of a bare PC SIP UA, which provides essential SIP functionality with less overhead and better security than a conventional OS-based SIP UA due to the absence of an OS. The underlying bare PC system enables the bare PC SIP UA to benefit from simple tasking, lean protocols, and efficient data handling.

To illustrate SIP implementation for VoIP on a bare PC, we discussed the software design and provided code snippets for sample data structures and methods of the bare PC SIP UA. In particular, we provided details about the data structures used to store information associated with the user and for setting up and managing calls. We then examined the key methods in the SIPUA and the call flow relationships among these methods.

The tests conducted show that the bare PC SIP UA can interoperate with both bare PC and OS-based SIP UAs and SIP servers. In addition, we conducted experiments to compare the performance of the bare PC SIP UA and a compatible Linux SIP softphone with respect to key SIP operations. The experimental performance results show that regardless of whether a bare PC SIP server or OS-based SIP server is used, the time for the Register and Invite operations using a bare PC SIP UA are significantly less than the time for these operations using a Linux SIP softphone.

To enhance VoIP security, we proposed a simple modification to the technique used for SIP authentication to generate an AES key for encrypting the SIP/SDES SRTP key exchange. The modification does not require any new

2012, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

messages and does not incur the overhead of a TCP/TLS exchange.

The bare PC UA provides essential SIP functionality with better performance than OS-based SIP UA, and immunity from OS-based attacks due to the absence of an OS. Bare PC SIP softphones can thus be used for improving performance in VoIP networks with OS-based SIP servers and softphones, or for building secure communication systems consisting of only bare PC SIP servers and UAs.

REFERENCES

[1] A. Alexander, A. L. Wijesinha, and R. Karne, "Implementing a VoIP SIP Server and User Agent on a Bare PC," 2nd International Conference on Future Computational Technologies and Applications (Future Computing), pp. 8-13, 2010.

[2] A. Alexander, A. L. Wijesinha, and R. Karne, "A Study of Bare PC SIP Server Performance," 5th International Conference on Systems and Network Communications (ICSNC) pp. 392-397, 2010.

[3] A. Alexander, R. Yasinovskyy, A. L. Wijesinha, and R. Karne, "SIP Server Implementation and Performance on a Bare PC," 5th International Journal on Advances in Telecommunications, vol. 4, no. 1 & 2, 2011.

[4] L. Chen, and C. Li, "Design and Implementation of the Network Server Based on SIP Communication Protocol," World Academy of Science, Engineering and Technology, pp. 138-141, 2007.

[5] S. Zeadally and F. Siddiqui, "Design and Implementation of a SIP-based VoIP Architecture," AINA, 2004.

[6] A. Singh, A. Acharya, P. Mahadeva, and Z-Y, Shae, "SPLAT: a unified SIP services platform for VoIP applications," International Journal of Communication Systems, vol. 19, no. 4, pp. 425-444, 2006.

[7] P. Zave, E. Cheung, G. W. Bond, and T. M. Smith, "Abstractions for Programming SIP Back-to-Back User Agents," IPTComm, 2009.

[8] S Siddique, RK Ege, SM Sadjadi, "X-Communicator: Implementing an advanced adaptive SIP-based User Agent for Multimedia Communication," SoutheastCon, pp. 271-276, 2005.

[9] K. J. Kim, Y, Jang, J. W. Chung, and J. H. Seo, "Design and implementation of SIP UA for a manufacturing network," International Journal of Advanced Manufacturing Techniques, vol. 28, no. 7-8, pp. 822-826, 2006.

[10] K. Singh and H. Schulzrinne, "Peer-to-Peer Internet Telephony using SIP," International Workshop on Network and Operating System Support for Digital Audio and Video, pp. 63-68, 2005.

[11] L. He, R. Karne, and A. Wijesinha, "The Design and Performance of a Bare PC Web Server," International Journal of Computers and Their Applications, vol. 15, pp. 100 - 112, June 2008.

[12] G. Ford, R. Karne, A. Wijesinha, and P. Appiah-Kubi, "The Peformance of a bare machine email server," 21st International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD, pp.143-150, 2009.

[13] G. H. Khaksari, A. L. Wijesinha, R. K. Karne, L. He, and S. Girumala, "A Peer-to-Peer bare PC VoIP Application," IEEE Consumer and Communications and Networking Conference (CCNC), pp. 803-807, 2007.

[14] G. H. Khaksari, A. L. Wijesinha, R. Karne, Q. Yao, and K. Parikh, "A VoIP Softphone on a Bare PC," Embedded Systems and Applications Conference (ESA), 2007.

[15] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, 2002.

[16] R. K. Karne, K. V. Jaganathan, T. Ahmed, and N. Rosa, "DOSC: Dispersed Operating System Computing," 20th Annual ACM Conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA), Onward Track, pp. 55-61, 2005.

[17] R. K. Karne, K. V. Jaganathan, and T. Ahmed, "How to Run C++ Applications on a Bare PC?" 6th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp. 50-55, 2005.

[18] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," RFC 5389, 2008.

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

[20] A. Alexander, A. L. Wijesinha, and R. Karne, "An Evaluation of Secure Real-Time Protocol (SRTP) Performance for VoIP," 3rd International Conference on Network and System Security (NSS), pp. 95-101, 2009.

[21] Kamailio (OpenSER) SIP server,[Online].Available: http://sourceforge.net/projects/openser Accessed: November 12, 2012.

[22] Linphone, [Online]. Available: http://www.linphone.org Accessed: November 12, 2012.

[23] Snom VoIP phones, [Online]. Available: http://www.snom.com Accessed: November 12, 2012.

[24] Wireshark, [Online]. Available: http://www.wireshark.org Accessed: November 12, 2012.

[25] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, 2008.

2012, © Copyright by authors, Published under agreement with IARIA - www.iaria.org