



**NAVAL
POSTGRADUATE
SCHOOL**

MONTEREY, CALIFORNIA

THESIS

**FUNCTIONALITY MINIMIZATION ANALYSIS OF
ASTERISK FOR FUTURE USE IN SECURE
ENVIRONMENTS**

by

Jeffrey A. Wiley, Jr.

September 2009

Thesis Advisor:
Second Reader:

Cynthia E. Irvine
Mark Gondree

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2009	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE Functionality Minimization Analysis of Asterisk for Future Use in Secure Environments			5. FUNDING NUMBERS	
6. AUTHOR(S) Jeffrey A. Wiley, Jr.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) <p>Asterisk, the open-source PBX, supports various implementations of voice over Internet Protocol (VoIP), a popular alternative to public switched telephone networks (PSTN) that offers cost benefits and ease of management. The Monterey Security Architecture (MYSEA) is a distributed multilevel security (MLS) environment designed to provide secure, collaborative sharing of information. It does not currently support either real-time voice communications or voice mail. The purposes of this thesis are to determine high-level VoIP requirements and to build a minimized version of Asterisk that supports these requirements. This minimized version of Asterisk could then be ported to run within the MYSEA architecture.</p> <p>To achieve this goal, threats were enumerated and requirements were determined. Then the modules within Asterisk were minimized to eliminate unnecessary functionality while still supporting mechanisms required for voice communications and voice mail. Testing showed that voice calls could be placed and voice mail messages could be left and retrieved using the minimized Asterisk server.</p> <p>Asterisk's functionality was successfully minimized to meet the requirements determined through the VoIP analysis by reducing the number of modules used for the build. This work provides the groundwork for future implementations of VoIP and voice mailboxes provided by Asterisk within MYSEA.</p>				
14. SUBJECT TERMS Asterisk, MYSEA, Open-source PBX, VoIP, Voice over Internet Protocol, Voice mail			15. NUMBER OF PAGES 106	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**FUNCTIONALITY MINIMIZATION ANALYSIS OF ASTERISK FOR FUTURE
USE IN SECURE ENVIRONMENTS**

Jeffrey A. Wiley, Jr.
Civilian, Naval Postgraduate School
B.S., Brigham Young University—Hawai'i, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
September 2009**

Author: Jeffrey A. Wiley, Jr.

Approved by: Cynthia E. Irvine
Thesis Advisor

Mark Gondree
Second Reader

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Asterisk, the open-source PBX, supports various implementations of voice over Internet Protocol (VoIP), a popular alternative to public switched telephone networks (PSTN) that offers cost benefits and ease of management. The Monterey Security Architecture (MYSEA) is a distributed multilevel security (MLS) environment designed to provide secure, collaborative sharing of information. It does not currently support either real-time voice communications or voice mail. The purposes of this thesis are to determine high-level VoIP requirements and to build a minimized version of Asterisk that supports these requirements. This minimized version of Asterisk could then be ported to run within the MYSEA architecture.

To achieve this goal, threats were enumerated and requirements were determined. Then the modules within Asterisk were minimized to eliminate unnecessary functionality while still supporting mechanisms required for voice communications and voice mail. Testing showed that voice calls could be placed and voice mail messages could be left and retrieved using the minimized Asterisk server.

Asterisk's functionality was successfully minimized to meet the requirements determined through the VoIP analysis by reducing the number of modules used for the build. This work provides the groundwork for future implementations of VoIP and voice mail provided by Asterisk within MYSEA.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
	A. MOTIVATION.....	1
	B. PURPOSE OF CURRENT STUDY	2
	C. ORGANIZATION OF PAPER	2
II.	BACKGROUND.....	3
	A. VOIP.....	3
	1. Concept of Operation	3
	a. <i>General VoIP</i>	3
	b. <i>Voice Mail</i>	4
	2. Call Setup Protocol.....	4
	a. <i>Session Initiation Protocol</i>	4
	3. Call Media and Data Transfer Protocols	5
	a. <i>Real-time Transfer Protocol</i>	5
	4. VoIP Using SIP/RTP	6
	B. ASTERISK	7
	C. THE MONTEREY SECURITY ARCHITECTURE.....	12
	D. SUMMARY	12
III.	THREAT ANALYSIS	13
	A. INTRODUCTION.....	13
	B. CONCEPT OF OPERATION.....	13
	1. Voice Mail in MYSEA	13
	C. THREAT ANALYSIS.....	14
	1. Threats in VoIP.....	15
	2. Threats in SIP.....	16
	3. Threats in RTP	18
	4. Threats in Asterisk	19
	D. MITIGATIONS.....	20
	E. SUMMARY	21
IV.	ARCHITECTURE, REQUIREMENTS, AND SPECIFICATION SELECTION.....	23
	A. GOALS.....	23
	B. ASTERISK ARCHITECTURE	23
	1. Back-to-back User Agent	23
	2. Thread Architecture.....	24
	3. Software Architecture	24
	4. Asterisk Concepts	26
	5. Codec Background.....	27
	6. Asterisk Threads.....	28
	C. REQUIREMENTS.....	31
	1. System Requirements	32

2.	Asterisk Requirements.....	33
D.	MODULE SELECTION	34
1.	Asterisk Module Relevance	35
E.	SUMMARY	37
V.	EXPERIMENTATION.....	39
A.	INTRODUCTION.....	39
B.	EXPERIMENTATION	39
1.	Module Selection	40
2.	Build Process.....	41
C.	TESTING	42
1.	Voice-Call Asterisk Testing.....	44
a.	<i>Configuration Files</i>	44
b.	<i>Description of Tests</i>	47
2.	Voice Mail Asterisk testing	49
a.	<i>Configuration files</i>	49
b.	<i>Description of tests</i>	52
D.	ANALYSIS AND RECOMMENDATIONS	54
1.	Analysis of Minimal Asterisk	54
2.	Recommendations.....	54
E.	SUMMARY	55
VI.	FUTURE WORK AND CONCLUSION	57
A.	INTRODUCTION.....	57
B.	FUTURE WORK.....	57
1.	Real-time Voice Communication within MLS Context.....	57
2.	Voice Communications Originating Outside the MLS Network	57
3.	Voice Communications between MLS Enclaves.....	58
C.	CONCLUSION	58
	APPENDIX A: INSTALLATION PROCEDURES	59
	APPENDIX B: CONFIGURATION FILES	71
	APPENDIX C: TEST PROCEDURES	78
	LIST OF REFERENCES.....	84
	INITIAL DISTRIBUTION LIST	88

LIST OF FIGURES

Figure 1.	SIP-based VoIP Architecture.....	6
Figure 2.	Asterisk Architecture.....	26
Figure 3.	Configuration file extensions.conf.....	27
Figure 4.	Thread and channel for an incoming call (native bridge).....	30
Figure 5.	Thread and channel for an incoming call using transcoding.....	31
Figure 6.	Experimentation network topology.....	40
Figure 7.	Background threads in Voice Mail Asterisk	42
Figure 8.	Voice-Call and Voice Mail Asterisk's modules.conf	45
Figure 9.	Voice-Call Asterisk's extensions.conf file	46
Figure 10.	Voice-Call Asterisk's sip.conf file	47
Figure 11.	CLI sip show peers output	48
Figure 12.	Error associated with dialing extension "544"	49
Figure 13.	Error associated with dialing extension "100"	49
Figure 14.	Voice Mail Asterisk's extensions.conf file	50
Figure 15.	Voice Mail Asterisk's voicemail.conf file	51
Figure 16.	Voice Mail Asterisk's sip.conf file.....	52
Figure 17.	CLI command sip show peers	53
Figure 18.	Test Network Topology.....	59

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Call Features in Asterisk.....	8
Table 2.	Required Modules for Voice-Call Asterisk	35
Table 3.	Modules for Minimized Voice-Call and Voice Mail Asterisk	36
Table 4.	Asterisk Build Details.....	42
Table 5.	Voice-Call Asterisk Requirements Tests	43
Table 6.	Voice Mail Asterisk Requirements Tests	44

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS AND ACRONYMS

AES	Advanced Encryption Standard
AMI	Asterisk Management Interface
API	Application Programming Interface
ARP	Address Resolution Protocol
CDR	Call Detail Record
CLI	Command Line Interface
COTS	Commercial Off-the-Shelf
DNS	Domain Name System
DoD	Department of Defense
DOS	Denial of Service
GPL	General Public License
GSM	Global System for Mobile communications
HTTP	Hypertext Transfer Protocol
IAX	Inter-Asterisk Exchange
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPsec	Internet Protocol Security
LAN	Local Area Network
MGCP	Media Gateway Control Protocol
MLS	Multilevel Secure
MYSEA	Monterey Security Architecture
PBX	Private Branch Exchange

PSTN	Public Switched Telephone Network
RFC	Request for Comments
RTCP	Real-time Transfer Control Protocol
RTP	Real-time Transfer Protocol
SIP	Session Initiation Protocol
SSRC	Signaling Source
TDM	Time Division Multiplexing
TLS	Transport Layer Security
VoIP	Voice over Internet Protocol

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Cynthia Irvine, for her support during this process. I would also like to thank Thuy Nguyen, for the assistance she provided. I also express appreciation to Dr. Mark Gondree, for his technical assistance and helpful guidance.

I am also deeply grateful to the Federal Cyber Service: Scholarship for Service program for making the pursuit of this degree possible.

Finally, I want to thank my dear, sweet wife, Starlyn Wiley, for her love and support through this growing experience.

This material is based upon work supported by the National Science Foundation, under grant No. DUE-0414102. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author, and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Department of Defense (DoD) missions require data and communications at various levels of classification. Separating these functions across physically separated networks dedicated to work at a single classification level is often inefficient, and a potential barrier to mission success; in such instances, the use of a multilevel secure (MLS) architecture may be beneficial. A MLS architecture maintains the separation of data at varying classification levels and enforces policy regarding access to that data. The Monterey Security Architecture (MYSEA) is a MLS architecture. In fact, MYSEA is a distributed MLS architecture, extending MLS capabilities to a local area network (LAN). MYSEA combines a small number of high-assurance components which enforce security policies and allows the use of open-source and commercial off-the-shelf (COTS) components. The services currently provided by MYSEA include e-mail and web-browsing capabilities.

Voice over Internet Protocol (VoIP) is a low-cost alternative to the public switched telephone network (PSTN) for real-time voice communication. Unlike the PSTN, VoIP does not need a dedicated connection; rather, it can use commonly available network infrastructures to provide services on a packet-switched network. Within the context of MYSEA, VoIP would be a beneficial service for providing real-time voice communication.

Asterisk, the open-source private branch exchange (PBX), is telephony software that contains a wide range of functionality supporting VoIP. Asterisk is flexible and allows different functionality to be added as it becomes necessary. This makes it a good choice for an integrated VoIP solution. Modifying Asterisk work within the MYSEA architecture, however, is no small task. This task is complicated by the size and complexity of the configuration chosen by default during the Asterisk build process. This leads to the following objective.

B. PURPOSE OF CURRENT STUDY

This research has the objective of determining high-level requirements and constructing an Asterisk-based VoIP system that meets these requirements. The ability of the system to meet these requirements will be validated by a set of well-defined experiments. We believe that our simplified Asterisk system may be helpful during the task of modifying Asterisk to provide VoIP services in the MYSEA MLS environment.

C. ORGANIZATION OF PAPER

Chapter II contains a discussion of background topics. These topics include a basic overview of VoIP and Asterisk, as well as general information about MYSEA. In Chapter III, a discussion of recognized VoIP threats is presented, helping to provide information concerning risks that need to be mitigated. Chapter IV introduces requirements and the specifics of a slimmed-down version of Asterisk as well as a discussion about Asterisk's architecture. Chapter V describes the experiments and tests which illustrate that the Asterisk configuration meets the VoIP and voice mail requirements. Finally, Chapter VI presents conclusions and suggests future work.

II. BACKGROUND

A. VOIP

Use of the Voice over Internet Protocol (VoIP) represents a shift for voice communication from standard circuit-based to packet-based communication. Traditionally, telephone communication has consisted of a direct circuit between two ends allowing real-time communication for the sending and receiving parties. Since the birth of internetworking and the spread of the Internet, packet-based communication has become the norm. Through the improvement of these technologies in such facets as latency reduction, throughput, and voice quality, it has become possible to use packet-based networks to send voice communications of reasonable quality. Although VoIP sends data using the Internet Protocol, higher level transport functions, such as session negotiation, media transport, and session control, are achieved using various technologies, for which there is no single standard.

1. Concept of Operation

a. General VoIP

The basic scenario for a VoIP call is from one personal computer using a software-based phone (softphone) to another computer with a similar configuration. Given this situation, suppose Alice needs to call Bob. Using her softphone, Alice would dial some identifier associated with Bob to cause his softphone to ring. Instead of a circuit connecting Alice and Bob directly, packets go from Alice to Bob via any number of routes between the two computers. At both ends, the users have a means of speaking and listening—such as using a microphone and speaker, or a headset—in which the two are integrated. When Bob becomes aware of an incoming call, he pushes a button to answer the

phone and the call begins. While this description is straightforward, there are a number of underlying operations taking place that are not perceived by the user.

b. Voice Mail

Continuing with the previous scenario, suppose that Bob is unable to answer his softphone to receive Alice's call. After a predetermined amount of time, the call is directed to the voice mail system, which automatically answers the call and prompts Alice to leave a message. After Alice leaves her message and the call is complete, this message is saved in Bob's personal voice mailbox. An indicator on Bob's phone informs him that a message has been received. Bob can then dial into the system and access his mailbox to retrieve the message Alice left for him. Bob may then perform a number of different actions with this message, including saving it, deleting it, or forwarding it to another mailbox.

2. Call Setup Protocol

In much the same manner that traditional telephony needed a switchboard and an operator to help establish circuits for communication, VoIP requires a similar logic to manage a call. The creation of a connection to Bob from Alice is managed using a collection of protocols and technologies. Although these protocols are standards-based, there are multiple protocols from which to choose. We have chosen to focus on a strategy using the Session Initiation Protocol. This strategy is particularly promising for our future applications, as determined from a survey and comparison performed by previous research [1].

a. Session Initiation Protocol

Session Initiation Protocol (SIP) has evolved through a number of revisions and updates since it was first proposed, in 1999, as a standards-track document for consideration by the Internet Engineering Task Force (IETF). From the latest revision of the protocol's request for comments (RFC): "SIP is an

agile, general-purpose tool for creating, modifying, and terminating sessions that works independently of underlying transport protocols and without dependency on the type of session that is being established” [2]. As stated, SIP is involved with the creation and termination of a session. It handles the signaling data used to prepare for communication, but does not actually handle the transport of voice media. SIP is not the only technology needed for SIP-based VoIP communications. We explore media transport next.

3. Call Media and Data Transfer Protocols

Many VoIP specifications consist of at least two different protocols, one for call setup and one for the actual transmission of the call. A standard SIP-based implementation of VoIP typically follows this pattern by using the Real-time Transfer Protocol (RTP) to send the contents of a voice call.

a. Real-time Transfer Protocol

First proposed as a standard in 1996, RTP is a flexible protocol that is suited for any type of real-time data traveling over a network. From the protocol’s RFC: “[RTP] provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services” [3]. This versatility makes it a prime choice for use with VoIP and other types of communication. Its generality affords implementers the choice of communicating a variety of media and does not limit communication to voice data.

RTP uses a clearly-defined header, with fields that correspond to the particular media the protocol is handling. For VoIP, this is typically an audio format. The RTP standard also defines the Real-time Transport Control Protocol (RTCP), a control protocol that helps manage the RTP connection and data transfer quality. Thus, a single RTP channel actually uses two ports: one for data,

and one for control. The RTP portion (the actual media) is always carried on the lower, even-numbered port, while the next, odd-numbered port, carries the RTCP portion (the control data).

4. VoIP Using SIP/RTP

The following are the details of a SIP-based implementation of VoIP. When Alice makes a call to Bob, her softphone contacts a VoIP proxy using SIP. If Bob is connected and registered to the VoIP proxy, it sends him an invitation on behalf of Alice. His softphone receives and handles this invitation by ringing, and perhaps, displaying caller ID data. The SIP proxy also sends a message to Alice's phone, indicating that an invitation has been sent to Bob. As Bob answers the call, his softphone sends signaling data, including what type of audio format it supports, to the VoIP proxy, to be forwarded to Alice's softphone. The VoIP proxy is an intermediary throughout the SIP connection, during which the softphones negotiate a new, ephemeral port over which to communicate voice data via RTP. The voice media travels directly between Alice and Bob over RTP. This simple SIP-based VoIP session is illustrated in Figure 1.

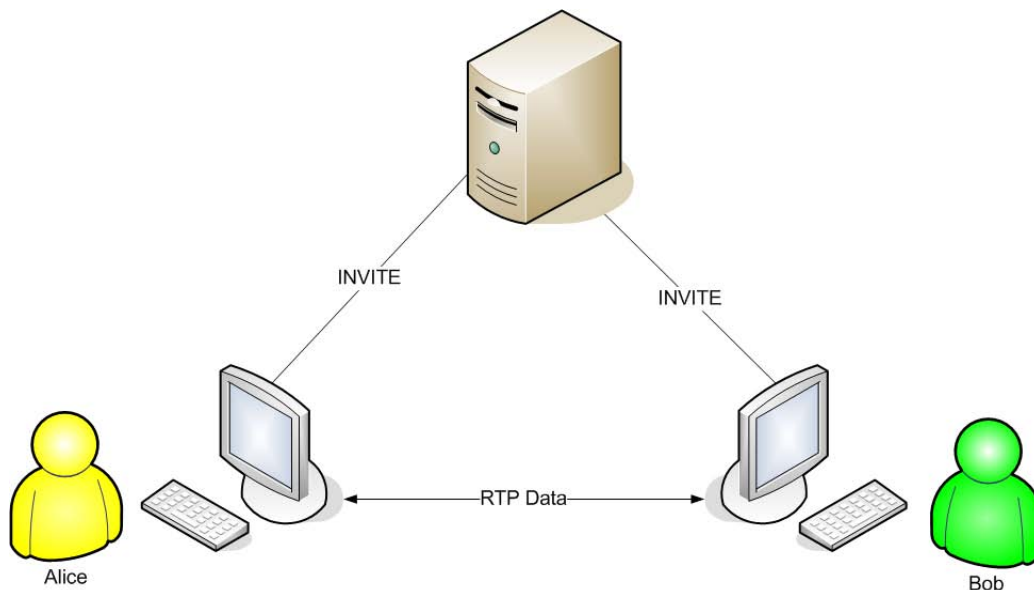


Figure 1. SIP-based VoIP Architecture

B. ASTERISK

Asterisk® [4] is a software project implementing a private branch exchange (PBX) system. Asterisk was created by Mark Spencer in 1999. It is the main product of Digium, Inc. [5], a company originally founded as Linux Support Services, L.L.C. by Spencer in 1999, later renamed and incorporated in 2002. Asterisk is released under a dual software license model, allowing licensees to choose either the GNU General Public License (GPL) or an alternate, proprietary license negotiated with Digium. Asterisk has grown and developed since its inception, and now touts a wide range of telephony features and support for various protocols and technologies. With appropriate hardware, Asterisk can connect to both standard circuit-switched telephone networks and packet-switched computer networks. Currently, Asterisk is one of the most popular PBX products available, claiming over 80% of the open-source PBX market [6].

Using a foundation of programs written in C, Asterisk is configured using a series of text files using a simple, yet flexible and extensible, language. The actions a PBX performs when a particular extension is dialed is defined in its “dialplan.” For Asterisk, the dialplan is defined in the extensions.conf file. Asterisk’s dialplan may take advantage of a range of custom applications, scripting languages, and external databases.

The Asterisk Project’s development community constantly adds to the list of features available within Asterisk. Table 1 summarizes the features available in the version of Asterisk (v1.6.1.0) under consideration in this work. For more information on these features, we refer the reader to [7], [8], [9]. Of most interest to us is the fact that Asterisk supports the features relevant to the current study, SIP-based VoIP and voice mail. The main goal of this study is to reduce the unnecessary logic and complexity of our PBX software (i.e., remove those features that are irrelevant to us), identifying those features integral to our functional requirements and their dependencies.

Table 1. Call Features in Asterisk

Number	Feature	Description
1	ADSI On-Screen Menu System	Analog display services interface (ADSI) allows Asterisk to provide menu items and customized information, such as name lookups, to appear on the screen of a telephone (customer premises equipment, or CPE) device connected to the network.
2	Alarm Receiver	Allows Asterisk to interface with some fire and burglar alarms.
3	Append Message	Append a voice mail message to an e-mail.
4	Authentication	IAX method of authentication using plaintext, md5, RSA methods to limit access to server.
5	Automated Attendant	Allows Asterisk to create an interactive session with a caller to direct an incoming call in various ways. Voice mail is an example of this.
6	Blacklists	Ability to explicitly deny calls based on a list of numbers.
7	Blind Transfer	Transfer of a call to a recipient without notifying the recipient, also unsupervised transfer.
8	Call Detail Records	Calls within or to the Asterisk server are able to store various types of information about a call. Types of information that can be stored includes call duration, caller ID of the incoming call, and destination of the call. It is possible to configure what information will be saved, thus allowing the administrator to determine what types of information are to be stored. This is an extended form of logging and auditing.
9	Call Forward on Busy	Allows configuration of the system to forward an incoming call to another phone number or extension if the line is busy.
10	Call Forward on No Answer	Allows configuration of the system to forward an incoming call to another phone number or extension if the line is not answered.
11	Call Forward Variable	Ability to forward a call based on different results such as time of day, caller ID, or an extension's status (e.g. busy).
12	Call Monitoring	Records call of a given agent (user) within the system. Allows monitoring incoming, outgoing, or both.
13	Call Parking	Sets up a "parking lot" to which calls can be transferred. This allows a call to be placed on hold, another party to be notified that someone is on hold in the "parking" area, and then dial the corresponding extension to "pick up" the call.
14	Call Queuing	Creates a queue for incoming calls, especially in a situation where there are more calls than extensions (e.g. a helpdesk). This also allows Asterisk to have control over whose phone (extension) rings next and can be used with other functions such as music on hold.
15	Call Recording	Allows calls to be recorded.
16	Call Retrieval	Paging for the correct person to pick up the call.
17	Call Routing (DID & ANI)	Allows the Asterisk server to ring several numbers at which the user may be available when a Direct Inward Dialing (DID) number is dialed. Automatic Number Identification (ANI) allows an incoming number to be recognized and handled accordingly.

18	Call Snooping	Allows calls to be heard based on a channel or an extension.
19	Call Transfer	Allows several ways to transfer a call to another extension or number.
20	Call Waiting	Allows a call to come into an extension that may be currently on a call. A sound is heard to indicate an incoming call to the extension owner who is then able to switch over to the incoming call.
21	Caller ID	Defines how an incoming call's number is to be displayed.
22	Caller ID Blocking	Ability to block caller ID from appearing when a number is called.
23	Caller ID on Call Waiting	Uses the phone's display to show the caller ID of an incoming call during an active call.
24	Calling Cards	Allows Asterisk to act like a calling card service: in order to make a call, there must be sufficient funds on the account. Asterisk examines the account, based on a PIN number to determine if enough credits are available for the call to be completed.
25	Conference Bridging	Allows the creation of conference calls.
26	Database Store / Retrieve	Allows Call Detail Records and voice mail to be stored in a database.
27	Database Integration	Allows information to be stored in a third-party database and accessed or changed during a call.
28	Dial by Name	From an on-screen menu (available on most softphone and some hard phones) a user is able to dial another user based on his or her name. Users can be looked up from a directory on the system. Additionally, suppose there is only one user named Alice: she would be able to be dialed by her extension or by entering <i>alice</i> as her extension.
29	Direct Inward System Access	Allows outside callers to call inside the system as if they were placing a call from a system-internal phone.
30	Distinctive Ring	Allows the phone to ring in a specific manner based on desired criteria.
31	Distributed Universal Number Discovery (DUNDi™)	DUNDi [10] is a peer-to-peer system for locating Internet gateways to telephony services. Unlike traditional centralized services (such as ENUM), DUNDi is fully distributed with no centralized authority whatsoever.
32	Do Not Disturb	Allows an extension to be set to "Do not disturb". Under this setting, the phone does not ring, rather it is automatically handled (e.g., by voice mail or forwarded to another extension).
33	E911	Allows users to dial 911 emergency services.
34	ENUM	ENUM is the common name for a collection of technologies which enable ordinary telephone numbers (E.164 numbers) to be mapped to an IP address, using special Domain Name System (DNS) record types.
35	Fax Transmit and Receive (3rd Party OSS Package)	Using a third party application, allows transmission and receipt of faxes; without this additional application, it is only possible for Asterisk to transmit—but not originate or receive—a fax.
36	Flexible Extension Logic	Ability to set and enable various features for different extensions.

37	Interactive Directory Listing	Allows the use of a directory from the system that can display on the screen of various phone devices.
38	Interactive Voice Response (IVR)	An automated voice system that allows callers to navigate a phone system and be directed to the correct extension by pressing a series of numbers on a touch-tone phone. (i.e. Push 1 for sales, push 2 for support, etc..)
39	Local and Remote Call Agents	People can log on to the PBX system from any phone using a Login ID, allowing them to receive and place calls.
40	Macros	A set of configurable functions available for use within the dialplan, allowing dynamic or complex functionality to be expressed simply.
41	Music On Hold	Music plays for the incoming party when they are placed on hold.
42	Music On Transfer:	Music plays for a party while the call is transferred.
42a	<ul style="list-style-type: none"> Flexible Mp3-based System 	Allows use of audio encoded with the MP3 format. This requires the possession of a license for the music being played.
42b	<ul style="list-style-type: none"> Random or Linear Play 	Plays music in a random order or linearly.
42c	<ul style="list-style-type: none"> Volume Control 	The volume of the MP3 music can be controlled.
43	Predictive Dialer	A predictive dialer is an outbound call processing system designed to maintain a high level of utilization and cost efficiency in the contact center. The dialer automatically calls a list of telephone numbers, screens the unnecessary calls such as answering machines and busy signals, and then connects a waiting representative with the customer.
44	Privacy	Requires the caller to enter her phone number if Caller ID information is not sent.
45	Open Settlement Protocol (OSP)	OSP is a protocol standard [11] for securely routing and accounting for inter-domain VoIP calls. OSP is not a VoIP protocol, but rather a standard for managing the billable exchange of VoIP sessions between IP networks. OSP is used by wholesale VoIP carriers to establish secure point-to-point peering between source and destination networks.
46	Overhead Paging	Also called intercom, this allows a centrally-located speaker to be "dialed into", for making announcements. Frequently seen in retail stores, car dealers, factory floors and other non-office type situations.
47	Protocol Conversion	Allows call endpoints to use different protocols and supports the conversion of one protocol to another.
48	Remote Call Pickup	Allows a user to call into the server and answer a call.
49	Remote Office Support	A remote office could be set up that interfaces with the local Asterisk server but provides a small network at a separate location.
50	Roaming Extensions	Phones can quickly be configured with an extension. This may allow a temporary employee to be able to work at any desk with a phone while keeping a single extension.
51	Route by Caller ID	A call can be routed to a specific recipient (extension)

		based on the result of the Caller ID information.
52	SMS Messaging	Ability to send SMS messages to mobile phones.
53	Spell / Say	Can "read" letters and numbers to a caller, for example reading back the current time.
54	Streaming Media Access	Allows streaming of media such as mp3s directly into the phone system, useful for hold messages.
55	Supervised Transfer	Similar to blind transfer, allows a person to transfer a call to another extension by first announcing that call to the transferred extension. Useful in situations like "I have Joe Carseller on the phone, do you want to talk to him?"
56	Talk Detection	There are a few modules which allow various types of noise detection on certain channels; talk detection can be used to trigger an event, for example, play an automated message when somebody says "hello."
57	Text-to-Speech (via Festival)	Asterisk may read text via the Festival voice synthesis suite. For example, this could be used to read a web page containing the current weather or an e-mail.
58	Three-way Calling	Supports standard 3-way calling, which allows another person to be added to a standard conversation between two users.
59	Time and Date	Upon calling a specified extension the current time and date information is played on the line.
60	Transcoding	Allows Asterisk to bridge calls that use different audio formats.
61	Trunking	Logically grouping together multiple phone lines for outbound dialing; typically seen in medium to large deployments. The PBX can be configured to auto-select an available line for outbound dialing rather than the user having to choose which line to dial out on.
62	VoIP Gateways	Asterisk can act as a bridge between VoIP telephones and the PSTN; additionally, Asterisk can be used to route calls to some third-party VoIP gateways (e.g. Vonage).
63	Voice mail:	Allows storage and retrieval of messages on a given extension.
63a	<ul style="list-style-type: none"> Visual Indicator for Message Waiting 	Creates a visual cue on the phone to indicate a message is in voice mail (e.g. a blinking light).
63b	<ul style="list-style-type: none"> Stutter Dialtone for Message Waiting 	A different dial tone is heard if the phone from which a call is being made has a message in the mailbox.
63c	<ul style="list-style-type: none"> Voice mail to e-mail 	Asterisk's native voice mail can send an e-mail to the voice mail recipient, and can optionally attach a WAV file of the entire message.
63d	<ul style="list-style-type: none"> Voice mail Groups 	Can be configured to leave a voice mail in multiple mailboxes or have a single shared voice mail for several users.
63e	<ul style="list-style-type: none"> Web Voice mail Interface 	A way to retrieve voice mail via the internet or a web client by connecting to an internal page.
64	Zapateller	Intended to block telemarketers, Asterisk plays a special tone recognized by some telemarketing centers as an indicator the number they have dialed has been disconnected.

C. THE MONTEREY SECURITY ARCHITECTURE

The Monterey Security Architecture (MYSEA) is a high assurance multilevel security environment used for the research and advancement of high assurance data processing and distributed multilevel security. MYSEA leverages evaluated, high-assurance products to allow untrusted, unevaluated, commercial off-the-shelf (COTS) products to operate in a multilevel secure (MLS) environment. Using relatively few trusted components, assurance is maintained, and the architecture is able to support applications operating concurrently, at different classification levels.

One of the goals of MYSEA is to “facilitate experimentation with . . . Secure collaborative information sharing” [12]. Currently, MYSEA has support for several services, such as e-mail and web browsing. “Secure multilevel VoIP” is a novel concept that has been considered in the context of MYSEA [1]. This work extends efforts towards this goal. We hope a future implementation of MLS VoIP may complement existing MLS services in MYSEA, providing a type of interactive and collaborative service that adds dimension to e-mail.

D. SUMMARY

Asterisk is an open-source PBX for VoIP with an active development community and an ever-expanding list of features. Its support for the SIP and RTP protocols makes it a natural candidate for future work investigating VoIP for MYSEA, following the guidance and methodology established by Tse [1]. We have provided some general background information on VoIP, Asterisk and MYSEA, to motivate the current research: feature elimination in Asterisk, in preparation for its use to provide VoIP services in a high-assurance environment like MYSEA. Next, we will explore some common security issues relevant to VoIP implementations and Asterisk.

III. THREAT ANALYSIS

A. INTRODUCTION

This chapter provides an overview of issues that threaten the security of VoIP communication. It commences with a concept of operation to help illustrate the type of VoIP implementation that may be possible within MYSEA. Then, we review threats common to many VoIP implementations and security issues facing VoIP protocols. This section also discusses the security issues specifically relevant to Asterisk and how they are addressed by the Asterisk community. The chapter concludes by reviewing mitigations to some of these threats.

B. CONCEPT OF OPERATION

Our concept of operation for voice mail in a multilevel secure environment retains many of the features of standard voice mail. The difference between these two are attributable to the multilevel secure system's enforcement of MLS policy, as described by the Bell-LaPadula [13] Model for confidentiality and the Biba Integrity Model [14]. These models describe constraints on information flow and the resulting mandatory access control policy (e.g., a secret subject shall not write to an object classified below the secret level).

Ideally, a minimized version of Asterisk that supports voice phone calls as well as voice mailboxes can be ported to the MYSEA architecture. This minimized version of Asterisk may have functionality added as needed, but is otherwise limited at build time to some narrow set of functionalities. The following is the concept of operations for VoIP in MYSEA from which we will extract requirements and define this narrowed set of functionalities.

1. Voice Mail in MYSEA

Alice logs in at the secret level and needs to relay an important message to her coworker, Bob. Bob is currently logged in at the secret level and is already

using his phone on another call. Thus, Alice must leave a message. When Alice uses her softphone to leave a message, she hears a generic prompt to leave a message at the extension she has called. After Alice leaves a message, an indicator on Bob's softphone alerts him to the message waiting in his mailbox. In this example, no classification boundaries are crossed and voice mail operates as in a single-level system.

Now, suppose Alice logs on at the secret level but Bob, unbeknownst to Alice, is currently logged in at the unclassified level. Since the mandatory confidentiality policy as reflected in the Bell-LaPadula model prevents Alice from writing down, she is again presented with generic instructions to leave Bob a message. In particular, Alice does not know the level of the session at which Bob is currently operating. Alice's message for Bob is left at her session level, regardless of Bob's current level. Bob, however, will not be able to retrieve this mail (nor will he be notified that he even has a message waiting for him) until he negotiates another session whose level dominates secret. The message is stored in a mailbox dedicated to Bob at the secret level. The messages are saved by Asterisk within the file system of a multilevel operating system that enforces the rules of the Bell-LaPadula model. This way, the message cannot be saved to a file whose level does not dominate the sender's classification level. In short, the message is saved at the secret level and can only be retrieved by Bob when he is working at a session level that is secret or higher.

C. THREAT ANALYSIS

Since our concept of operation is VoIP in MYSEA, we inherit the many strong security features of MYSEA. In particular, this includes IPsec, the identification and authentication of users and processes, and confinement to a specific security level [15], [16], [17]. It is, however, important to have knowledge of the basic threats to VoIP and its underlying protocols within a more general context. This knowledge helps designers to articulate security requirements that prevent possible insecure implementations or the possibility of configuring VoIP

in a manner that makes it insecure in spite of the features available within the MYSEA architecture. A brief discussion of the threats specific to VoIP and Asterisk is presented next.

1. Threats in VoIP

In the absence of any extra measures, VoIP has a number of vulnerabilities against which attacks can be mounted. Some of the basic threats against VoIP include unauthorized use of telephone services, attacks taking advantage of poor phone configurations, and malicious impersonation of various parts of the VoIP infrastructure.

A potentially expensive threat against VoIP is the unauthorized use of telephone services, such as long distance calls. By calling into a number that is part of a VoIP system, an attacker may use various techniques to make a call without paying for the service. Such a call would originate from another system to which the attacker has connected, either legitimately or illegitimately. By using a certain sequence of menu choices within an automated attendant menu, an attacker could gain the ability to make a long-distance phone call. The end result might be that the call is placed at the expense of the recipient. This can be a costly experience and, if the attacker is conservative in her use of the services, might continue undetected for a long time, especially in the absence of regular auditing [18].

Another vector of attack against VoIP is the phone device itself. This type of attack focuses on the configuration of a VoIP handset. It is possible to retrieve the configuration file from a phone, giving an attacker information that may be used to further attack or infiltrate a VoIP network [19]. In some cases, it is also possible for an attacker to create her own configuration file and send it to other handsets, across the network. For some phones, a malicious configuration file could cause the phone to direct the VoIP system to record calls, unbeknownst to either caller. Using this ability, an attacker could cause a phone to connect to a rogue server under her control, to intercept and record the call [19].

In addition to using a malicious configuration file that redirects a phone to connect to a rogue VoIP server, it is quite trivial for an attacker to impersonate a legitimate VoIP server. The attacker could accomplish this by building a rogue Asterisk server and monitoring network traffic. In the case of a SIP server, when the attacker sees an INVITE message sent by a user agent on the network, the rogue server could respond with its own messages. These spoofed messages would make the user agent think that it needs to authenticate with the rogue server. In the absence of any authentication, the attacker is given access to the legitimate party's calls and possibly other information. Of course, similar attacks exist for VoIP protocols other than SIP.

2. Threats in SIP

Many of the threats to SIP result from the fact that the protocol data, by default, is sent in clear text. Passwords and other potentially sensitive data can be captured over the network quite trivially. Known threats include lifting data out of the clear text protocol, offline password cracking, Man-in-the-Middle attacks, registrar and proxy spoofing, and denial of service attacks.

Since information is sent in the clear through SIP signaling channels, it is trivial to capture data from the packets being transmitted. The types of data valuable to an attacker that can be captured from these packets include usernames, passwords, and sequence numbers. This data has value to an attacker as it may be used in other exploits and in multi-stage attacks.

The RFC for SIP suggests a way to perform (one-way) authentication at the start of a SIP session [2], [20]. Following the HTTP digest authentication scheme, the RFC suggests hashing parts of authentication messages to create a digest, instead of sending authentication data in the clear. The messages that are hashed include the username, realm (i.e., SIP domain), password, and challenge-response data. The digest is a token that can be computed by either party based on pre-shared (e.g., password), and recently communicated, data, allowing a user to authenticate without sending her password in the clear. There

are, however, well-known attacks to extract messages from digests when the search space is small, e.g., using brute force password cracking or rainbow table lookups [19], [21]. Since everything except the password was provided in the clear, at some earlier point during the SIP conversation, the search space is very small and these techniques are quite applicable. Making the situation worse, many passwords used for voice mail and other telephony services often consist of only 4-6 characters, usually limited to numbers. This further reduces the search space and increases the speed with which an offline password lookup or brute force attack could be performed.

As with many protocols lacking mutual authentication, SIP is susceptible to Man-in-the-Middle attacks. Through ARP cache poisoning or DNS spoofing, an attacker can redirect and capture all of the data from the softphone and create her own connections to a SIP server [19], [22], [23]. This could potentially provide the infiltrator with passwords or even the data stream from a call (for more details, see Section 3). Spoofing proxy servers and SIP registrars allows attackers to record and change whole conversations [19]. For example, an unintended party who is spoofing a legitimate registrar could submit a forged response to a SIP REGISTER message [19]. When the unsuspecting user makes a call, it will be routed to the attacker's server instead, allowing the call to be intercepted.

Finally, another well-known class of attack to which SIP is susceptible is the denial of service (DOS) attack. Abusing the basic signaling messages of the SIP protocol, a call can be forcibly ended or even prevented. By gaining just a few bits of information from a SIP session, such as an IP address and call identification, a BYE message can be forged and sent on behalf of a user currently in a call. When the party at the other end of the call receives this message, the recipient's softphone acts as if the call is complete and the connection will be torn down. Similarly, if a user is trying to make a call, an

attacker can forge a CANCEL message that appears to originate from the caller. This message tells the callee's softphone to cancel the attempt to initiate a session [19].

Although this is not an exhaustive list of all of the attacks possible against SIP, it suggests that attention is warranted when configuring SIP as a mission-critical application. Many of these attacks can be mitigated and prevented, or have their likelihood of occurring greatly diminished (see Section D).

3. Threats in RTP

The Real-time Transfer Protocol (RTP) is susceptible to a number of attacks. These attacks include spoofing, hijacking, denial of service, and traffic manipulation. Since data is not encrypted, an attacker can eavesdrop on a conversation or inject into it her own audio. The potential loss of integrity and confidentiality makes these attacks against RTP especially severe.

Since RTP transfers the media stream of a conversation, a significant portion of a conversation's stream may need to be captured for an attacker to have any useful data. By using a Man-in-the-Middle attack, however, a perpetrator could easily capture the entire stream of a conversation. Without encryption, the captured audio simply needs to be played back and the attacker has access to the entire conversation [19].

Though slightly more complex, voice injection uses the same Man-in-the-Middle attack to allow the data to be modified by the attacker. The attacker can use captured packets to extract sequence numbers and signaling source values (SSRC) from the conversation. These allow the attacker to spoof packets and send a prerecorded audio message [19]. In this manner, the conversation may receive messages that were not actually intended by either legitimate call participant.

4. Threats in Asterisk

Both the Makefile and README [24], included with Asterisk's source code, encourage those interested in running Asterisk to read the section of Asterisk's documentation about security before continuing. At less than two pages in length, this section of the Asterisk documentation begins with a warning regarding the topic which is the main focus of Asterisk security: the potential unauthorized use of phone services by attackers, resulting in hefty phone bills [25]. This document breaks down Asterisk security into two areas: network security and dialplan security.

Network security from the standpoint of Asterisk entails limiting access to an Asterisk server. The documentation encourages use of SSH or VPN solutions for access to the management portion of the Asterisk server, if remote management is enabled. Also, since certain ports are opened by default for various channels, like SIP, the documentation encourages using the configuration files for these protocols to explicitly configure Asterisk to permit or deny access to the port on a per-user basis. This section of the documentation explicitly mentions that Asterisk (v1.6.1.0) does not yet support encryption for SIP [25].

Dialplan security in Asterisk is intended to prevent the use of unauthorized phone services. Several measures are suggested to prevent this. First, if one uses the default extensions.conf file, the unused and unnecessary contexts should be removed. Second, dialplan contexts should be used to keep incoming calls out of contexts that allow outgoing and long-distance calls. Otherwise, if an attacker is able to be transferred to a line that can access a context with privileges to call long distance, the attacker may be able to incur charges for the owner of the system. This does not prevent an outside party from calling a user with access to long-distance calling; rather, it refers to a separation of privilege between trusted parties (e.g., users with a valid internal line) and untrusted

parties (e.g., individuals calling into the system). By following this guidance, the potential for an attacker to make unauthorized calls is reduced.

In addition to these threats, Asterisk is occasionally faced with security issues due to exploitable flaws in the code. In order to respond to known problems and mitigate their possible exploitation, the Asterisk community publishes security advisories [26]. These advisories provide detailed information about each problem and, whenever possible, provide patches or guidance on problem mitigation.

D. MITIGATIONS

As mentioned earlier, guidance for securing Asterisk focuses on securing the dialplan to prevent the unauthorized use of telephone services. A number of tutorials and other documents have been written to help Asterisk administrators secure their dialplan [27], [28]. This problem can also be mitigated by ensuring there is no connection from the system to another telephony infrastructure. In particular, if Asterisk is on an isolated local area network and is not able to make outside calls, the threat is essentially eliminated.

Asterisk can be configured to ensure that SIP passwords are not sent in the clear over the network. When configured this way, the cryptographic hash of the password is sent. However, as previously described, these hashes can be captured on the network and the passwords can be extracted using well-known techniques. One mitigation for this is the use of Secure SIP (SIPS) [2], in which transport layer security (TLS) protects the SIP conversation. As of this writing, a stable version of SIPS for Asterisk remains a work in progress [29], [30], [31].

Another security feature that is the subject of active development for Asterisk is secure RTP (SRTP) [32]. The goal of SRTP is to provide protection from eavesdropping and audio injection. The most intuitively valuable part of the conversation, voice data, is sent unencrypted when RTP is used to distribute the

media. With the use of SRTP, voice media is protected using strong encryption, such as the advanced encryption standard (AES) algorithm.

E. SUMMARY

Numerous attacks can be mounted against VoIP, targeting a range of weaknesses, from protocol issues and telephony hardware problems to bugs in Asterisk's implementation. Asterisk has a community actively engaged in implementing the latest secure features and improvements of protocols, including SIPS and SRTP. Overall, knowledge of these threats helps to highlight the technology layers wherein a VoIP implementation may need additional protection.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ARCHITECTURE, REQUIREMENTS, AND SPECIFICATION SELECTION

A. GOALS

Asterisk contains a large number of functions unnecessary for the future goal of creating a VoIP implementation for use in a multilevel secure context. This is evidenced in our summary of Asterisk's features (Table 1, Chapter II). These extra features, however, ultimately translate into performance overhead and unnecessary logic, in which vulnerabilities may exist. In order to minimize the necessary functions, Asterisk's core functionality and modules must be identified. The superfluous functionality may then be eliminated. First, we summarize the design and architecture of Asterisk's implementation. Then, we define the requirements for our target Asterisk system, capturing the functionality required by our concept of operation from Chapter III.

B. ASTERISK ARCHITECTURE

The following section summarizes those architectural components of Asterisk useful to understanding the basic process by which the software implements SIP-based VoIP. This discussion will be helpful in understanding the dependencies identified later, in our minimal Asterisk system.

1. Back-to-back User Agent

While Asterisk may be configured to act as a SIP proxy (Figure 1, Chapter II), its default behavior is as a "Back-to-back User Agent" (B2BUA). As a B2BUA, Asterisk acts as a user agent from the perspective of both users involved in a conversation. This results in two simultaneous conversations within Asterisk: one between the call originator, and one with the call recipient. In this manner,

Asterisk stands between the callers throughout all communication. In particular, the RTP portion (voice data) passes through Asterisk, and not directly between the two callers.

2. Thread Architecture

Asterisk is a multi-threaded program. Upon startup, a main Asterisk process starts a number of threads, each associated with a module or a specific functionality. It may be possible to configure an installation with a minimal number of threads. For us, a minimal set of threads would necessarily include those used to manage core PBX functions and calls over a SIP channel.

3. Software Architecture

Asterisk's software architecture is modular and extensible. Asterisk was designed to encourage its development community to contribute to its features, through the creation of modules. These modules are made accessible to the other parts of Asterisk by conforming to one of several module application programming interfaces (APIs). Standardized APIs allow new modules to be integrated easily into Asterisk, expanding its support for new codecs, file formats, communication protocols, and dialplan functions. The essential APIs of Asterisk are discussed below.

The Channel API defines the interfaces each channel technology must implement and make available. These interfaces, in effect, hide technology-specific details from the PBX and from other modules. Thus, incoming calls can be handled in a technology-independent fashion. For example, calls are transferred or bridged, with little difference to Asterisk's PBX subsystem, whether they originate from an Inter-Asterisk eXchange (IAX) trunk or over H.323. As a consequence, as new protocols and standards for VoIP telephony are developed, the channel technologies supported by Asterisk may be extended with little modification to the rest of its code base.

The Codec Translator API defines those interfaces each codec module must implement and make available. The modules enable Asterisk to transcode audio between channels when the clients fail to negotiate a common audio format during the call. When transcoding, Asterisk converts audio data from one format to another, allowing the two clients to communicate. For example, a user on a mobile phone may call a user on the public switched telephone network (PSTN) through Asterisk, and the PBX will bridge the call, acting as the middleman by transcoding the audio between G.711 and GSM. These clients would otherwise be unable to communicate, if Asterisk did not operate as a B2BUA and transcode their audio.

Asterisk also provides a File Format API to read and write files of different types. These modules are invoked, for example, when a file is written after leaving a voice mail message. In this case, the caller speaks and Asterisk collects and saves the incoming media. Asterisk will save this file using a format, such as GSM or wav, chosen according to its configuration. Similarly, saved messages must be opened and read by Asterisk, to be played back to the user.

Asterisk's Application API provides those interfaces that allow applications (or "apps") to register with the various subsystems of Asterisk during start-up. For example, an application might make itself available as a function callable during dialplan execution (a dialplan function), accessible via Asterisk's management interface (an AMI function), or via the command line interface (a CLI function). In essence, the Application API allows third-party "apps" to be integrated into Asterisk's various subsystems.

In addition to the functionality provided by each module, there are also many Asterisk subsystems that are configurable and feature-rich. Some of these systems are singular in purpose, like the "loader" (which reads in modules during start-up) and the "PBX core" (which handles the majority of the PBX functionality). Other systems, however, provide very general functionality that is useful throughout Asterisk, like the scheduler, the I/O polling manager, the thread

manager, the call detail record (CDR) handler, and the logging subsystem. The relationships among Asterisk's subsystems and modules are suggested in Figure 2.

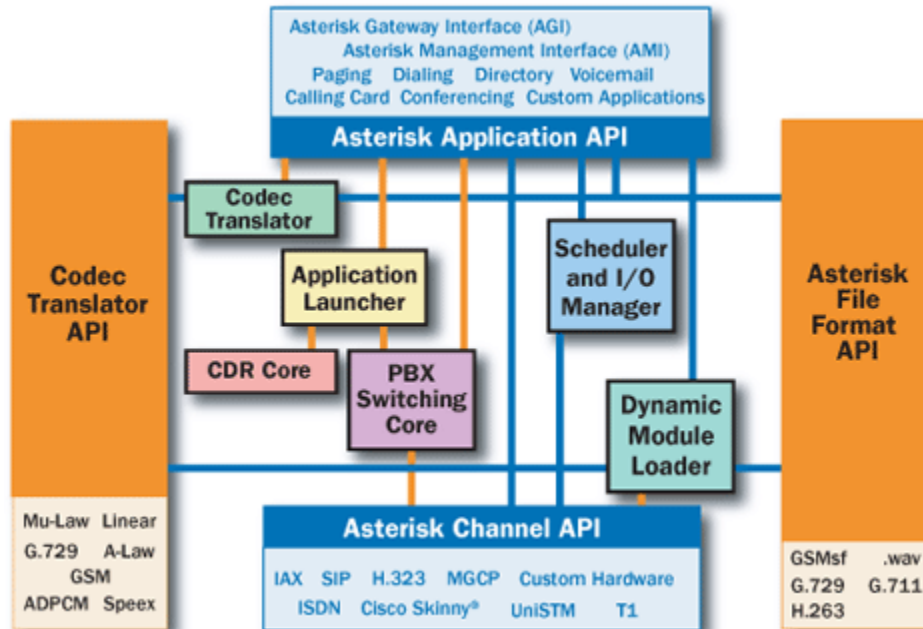


Figure 2. Asterisk Architecture¹

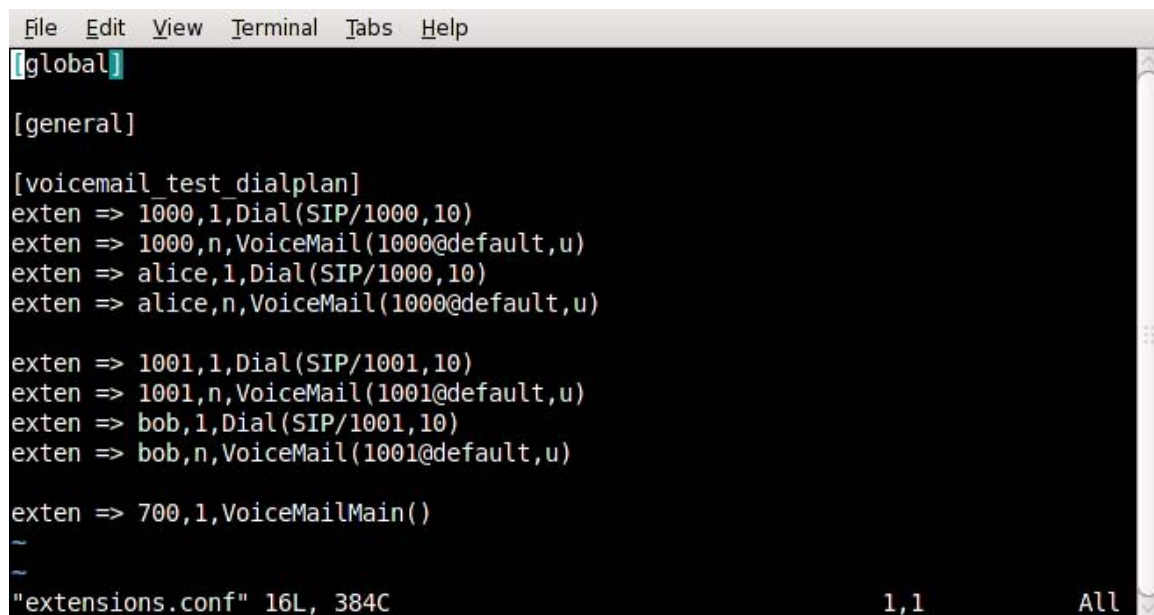
4. Asterisk Concepts

A channel refers to the connection over which a VoIP conversation occurs. Gonçalves provides a good definition of an Asterisk channel as follows, “A channel is the equivalent of a telephone line, but in digital format. It usually consists of an analogic or digital (TDM) signaling system or a combination of codec and signaling protocol (e.g., SIP-GSM, IAX-ulaw)” [9]. A number of signaling protocols are available to Asterisk, such as SIP, MGCP, or H.323, via its channel modules [32]. The subsystem managing the protocol’s logic is said to “drive the channel,” and is called the channel driver. Asterisk can also access a

¹ Image reproduced with the permission of Digium, Inc. [34].

variety of audio codecs, including GSM, Speex, and μ -Law, via its codec modules [32]. Our minimized Asterisk systems exclusively use SIP-GSM channels.

In Asterisk, the PBX's dialplan is defined in the *extensions.conf* configuration file. Recall, the dialplan determines what actions are taken when an extension is requested. In our example dialplan (Figure 3, below), when extension 1000 is requested, Asterisk invokes the Dial() function, to call an extension (1000) on a particular channel (SIP). Additionally, our dialplan states that, after a predetermined time (10 seconds), if the Dial() function fails to connect, Asterisk should invoke the VoiceMail() function.



```
File Edit View Terminal Tabs Help
[[global]]
[general]
[voicemail_test_dialplan]
exten => 1000,1,Dial(SIP/1000,10)
exten => 1000,n,VoiceMail(1000@default,u)
exten => alice,1,Dial(SIP/1000,10)
exten => alice,n,VoiceMail(1000@default,u)

exten => 1001,1,Dial(SIP/1001,10)
exten => 1001,n,VoiceMail(1001@default,u)
exten => bob,1,Dial(SIP/1001,10)
exten => bob,n,VoiceMail(1001@default,u)

exten => 700,1,VoiceMailMain()
~
~
"extensions.conf" 16L, 384C 1,1 All
```

Figure 3. Configuration file extensions.conf

5. Codec Background

A codec is an algorithm that encodes and decodes audio/video data. With respect to the current discussion, Asterisk codecs encode and decode voice data. For our minimized system, we chose to configure Asterisk to use a single codec: the Global System for Mobile communications (GSM) codec. The GSM codec is popular for use in mobile communications, provides decent sound

quality, low processing impact for conversion and it has no royalty fees [33]. The choice to support only a single audio codec in our systems—our voice mail audio files and clients all use the GSM format—reduces the complexity of our Asterisk systems and helps to avoid the heavy performance cost associated with transcoding during operation [35].

6. Asterisk Threads

Asterisk starts a number of threads during its initialization process. The number and purpose of threads vary depending on the configuration, modules, and settings of Asterisk. One of the primary goals of this research is to minimize the number of running threads on an Asterisk system. In order to accomplish this, it is important to have an idea of some of the threads and the actions they perform.

Upon startup, a number of subsystems start and modules are loaded, many of which use persistent threads to manage specific background tasks. A typical installation of Asterisk has more than 30 background threads handling tasks as diverse as monitoring I/O, handling scheduled tasks, and managing specific channels. We will mention the behavior of some notable background threads present in our minimized Asterisk system (Figure 7, Chapter V). The first of these is the *core event dispatcher* thread², which manages generic internal Asterisk events. Next, the *listener* thread begins, which manages switching Asterisk between daemon and console modes. Then, the *logger* thread begins, which initializes and manages Asterisk's logging capabilities. The *do_devstate_changes* thread then starts to manage the state (not in use, ringing, busy) of extensions and channels. A thread² managing the PBX core is then started. The *do_parking* thread then starts, which is involved in call transfers, "call parking", and related features. The *do_monitor* thread is the SIP channel driver, investigated in more detail later. The next thread² manages certain voice

² This thread is created by a subsystem in Asterisk that handles some thread management. It is listed in Figure 7 as a *tps_processing_function* thread.

mail tasks. The final thread to be loaded is the *monitor_sig_flags* thread, which responds to events registered with the signal handlers in Asterisk.

The SIP channel driver thread (*do_monitor*) polls the default UDP port for SIP traffic and manages open “SIP conversations” on other ports. After handling the portions of the conversation related to registering, authenticating, etc, this thread starts the *pbx_thread* to handle the rest of the call. The *pbx_thread* executes the dialplan, which results in a wide range of possible activities that includes using applications, reading and writing media to the channel, etc.—a nearly unlimited number of scenarios can follow. One of the more interesting scenarios occurs when an outgoing call is requested. In this case, the *pbx_thread* creates an outbound channel to the recipient, and bridges these channels when the recipient answers the call. When two user agents involved in the call request to use the same codec and connect via the same channel technology, Asterisk is able to bridge the call using a “native bridge.” During a native bridge, no audio transcoding is required during the conversation and the data transfer occurs directly in the channel driver (Figure 4). When clients use different codecs, however, the *pbx_thread* must perform audio transcoding on their behalf. In this scenario, the *pbx_thread* bridges the channels and converts the audio from one format to another (Figure 5). If the outgoing call is not answered, the dialplan directs the PBX what steps to take next. When the call completes, the *pbx_thread* terminates.

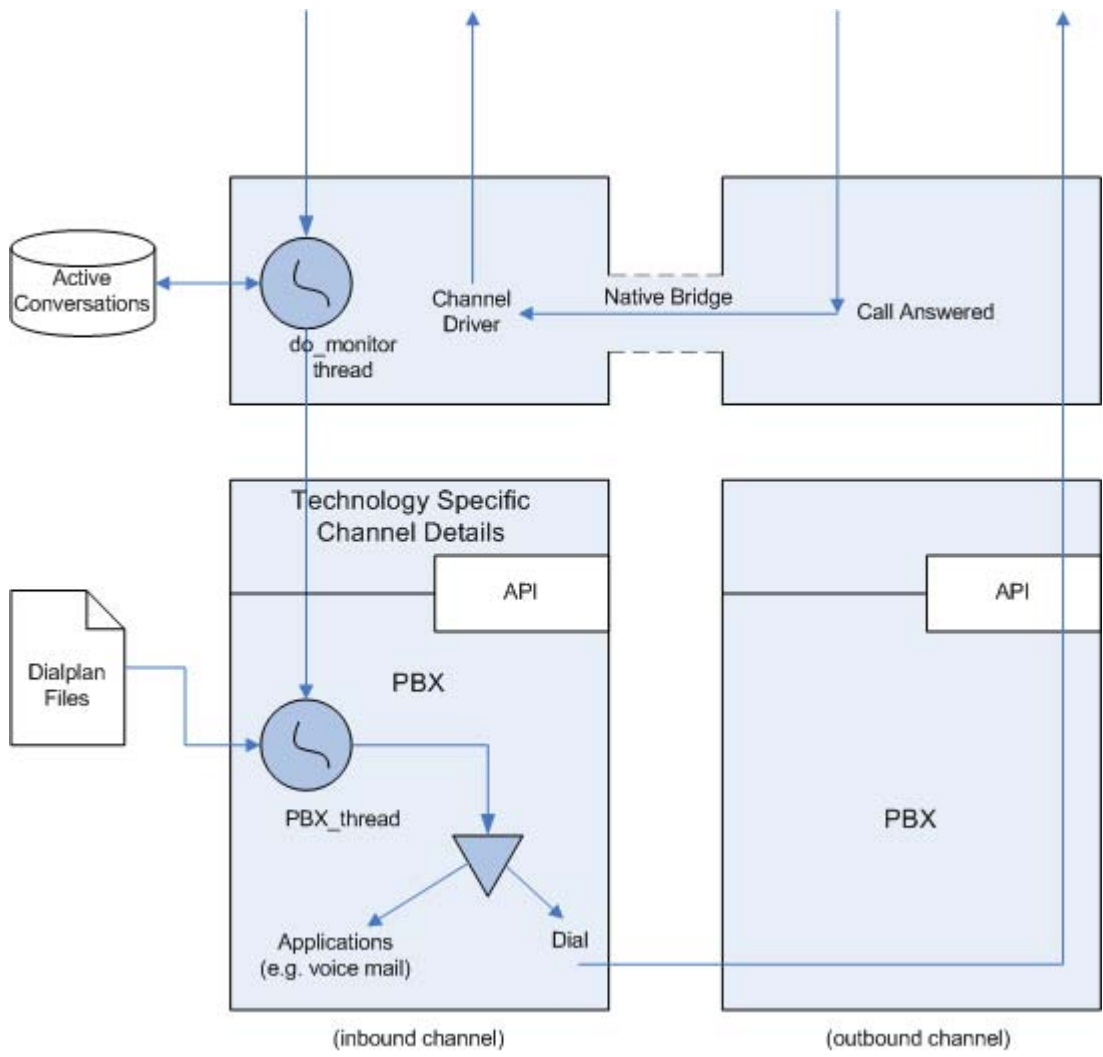


Figure 4. Thread and channel for an incoming call (native bridge)

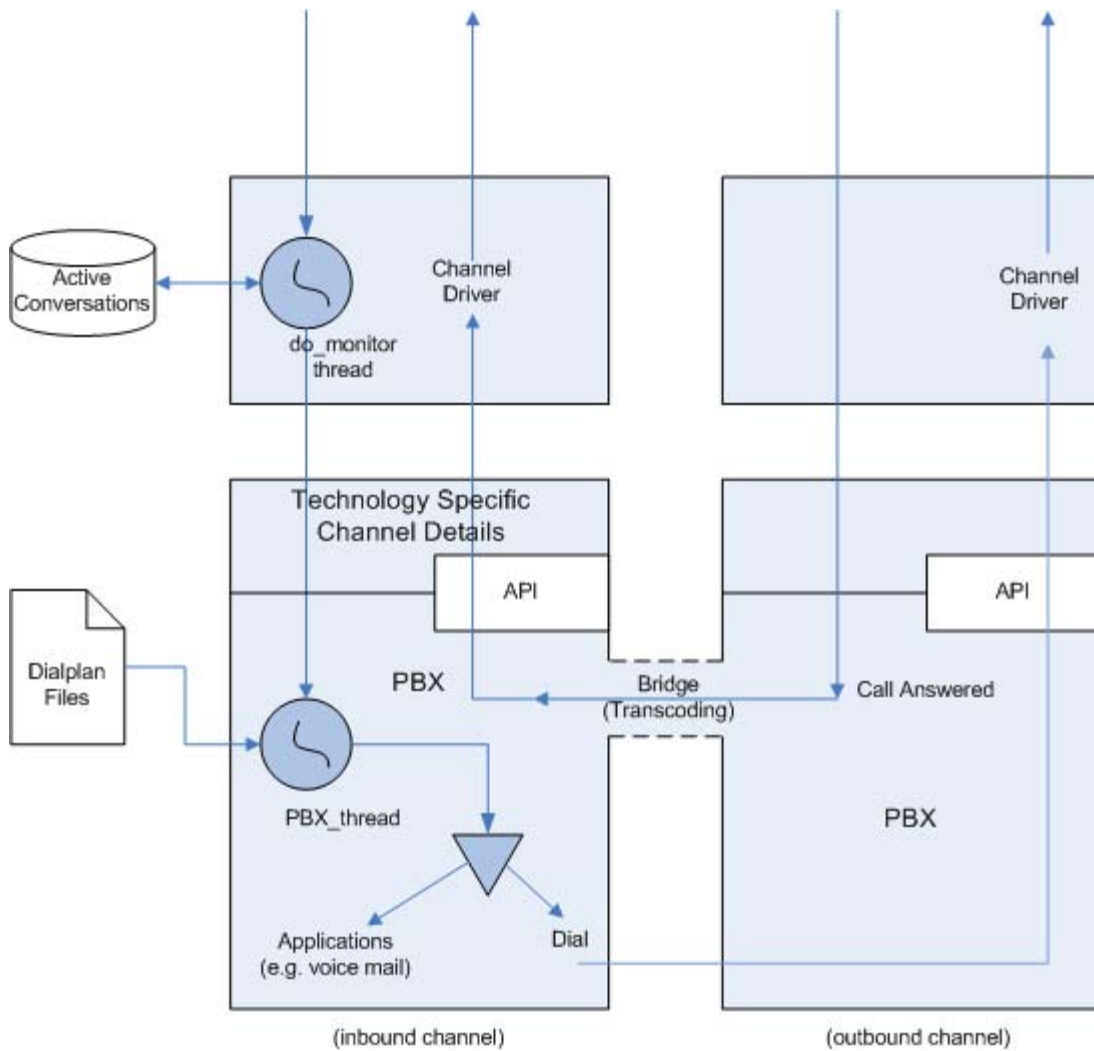


Figure 5. Thread and channel for an incoming call using transcoding

C. REQUIREMENTS

The idea of a “minimal” Asterisk system only makes sense relative to a set of requirements satisfied by the system. Our systems are minimal in the sense that any configuration selecting fewer modules or installation options results in a system that fails to meet these requirements. In particular, for our system, modules were added during Asterisk’s installation until our requirements were met. Using this methodology, an Asterisk server with few additional capabilities resulted.

In this section, we develop two sets of requirements, each describing a set of abilities required in our target concept of operations (Chapter II). The first set of requirements pertains to support for SIP-based phone calls. We call a minimal Asterisk system meeting these requirements a “Voice-Call Asterisk” system. The next set of requirements includes some additional support for voice mail. We call a minimal Asterisk system meeting these requirements a “Voice Mail Asterisk” system. These two sets of requirements are described in detail next.

1. System Requirements

The following are functional requirements for any Voice-Call Asterisk system. These pertain to a system in which callers may have VoIP conversations with other system users, where Asterisk uses SIP as its signaling protocol.

- A caller shall be able to contact another user by entering that party’s extension (SIP identifier) using a softphone.
- Asterisk shall be able to determine valid extensions and send the necessary information for the destination party’s softphone to ring.
- Using SIP, the users’ softphones shall be able to successfully negotiate an audio codec to be used for the voice portion of their communication.

In addition to the above functional requirements, the following items are necessary for any Voice Mail Asterisk system. These requirements are the minimum for a working voice mail system. Some features may be administratively controlled (e.g., mailbox size), but this is to be limited by the policy of the individual implementation. The administrative controls we have chosen are the default settings from the samples shipped with the version of Asterisk under consideration (see Appendix B, voicemail.conf).

- The dialplan shall be able to direct an unanswered call so that it is handled by the voice mail system.

- The voice mail system shall be able to provide a caller with an option to leave a message for the intended party.
- The voice mail recipient shall be able to receive a notification indicating that a voice mail has been received.
- The voice mail recipient shall be able to call an extension that connects to the voice mail system.
- The voice mail recipient shall have the ability to retrieve voice mail messages from a personal account within the voice mail system.
- The voice mail recipient shall be able to manage personal messages, including saving or deleting them.

2. Asterisk Requirements

To support these functional requirements, Voice-Call Asterisk must be able to do the following:

- Understand the session initiation protocol
- Handle SIP INVITE messages
- Dial other users (forward the INVITE message to the recipient)

Additionally, to support the voice mail functional requirements in the previous section, Voice Mail Asterisk must be able to meet the following requirements:

- Send notifications when a voice message is recorded
- Answer a call if it is not answered by the intended party
- Use format translators to read and play back the static sound files that make up the menus, selections, and information of the voice mail system

- Use an audio codec to save the voice mail into a file following a predetermined format
- Store voice mail for later retrieval by the receiving party
- Play voice messages back to the recipient during subsequent retrieval

Due to our decision to support the SIP signaling protocol, Asterisk needs to have the capability to receive and process SIP requests. As SIP INVITE requests are made, the codecs and RTP ports are negotiated. For Voice-Call Asterisk, the configuration has been simplified through the choice to use a single, system-wide audio codec. Since our softphones use the same codec, Asterisk will create a native bridge during the calls and does not need to transcode audio data.

As will be seen in the next section, Voice Mail Asterisk contains those modules needed to meet this set of requirements. Adding voice mail functionality increases the number of Asterisk modules required, compared to Voice-Call Asterisk. In Voice-Call Asterisk, the system does not interact directly with a call other than to set up, bridge, and tear down the call. To add voice mail support to the system, Asterisk now needs to perform additional tasks, including answering calls, sending voice data over the network, and receiving voice data.

D. MODULE SELECTION

Initially, trial systems meeting our requirements were created based on what could be gleaned from Asterisk's documentation and module names. Essentially, much of the process was an ad-hoc search for the "core modules" supporting our requirements. The outcome of our search for relevant modules is discussed next. We describe the modules ultimately chosen for our minimized systems and discuss their specific relevance in the next section.

1. Asterisk Module Relevance

Modules we identified as necessary for any Voice-Call Asterisk system are listed in Table 2. A number of these modules bring key components to Voice-Call Asterisk, while others are needed to satisfy dependencies. The `pbx_config` module allows the use of user extensions from a static configuration file [36]. The `chan_sip` module allows Asterisk to register SIP clients and perform necessary SIP interactions. This module requires the `chan_local` module, which is used internally by Asterisk for creating and managing channels. The `app_dial` module is needed to link one party to another. It enables the `Dial()` function to be used within the dialplan. When used, this function directs Asterisk to dial some extension on behalf of an incoming call. Appendix A provides details concerning configuring and installing our minimized Voice-Call Asterisk system.

Table 2. Required Modules for Voice-Call Asterisk

Module Type	Module Name	Why it is necessary	Dependencies
Application	<code>app_dial.c</code>	Allows an extension to be reached when it is dialed. Enables the <code>Dial()</code> function for use within the dialplan.	
Channel Driver	<code>chan_local.c</code>	Used internally by channel driver modules.	
Channel Driver	<code>chan_sip.c</code>	Enables Asterisk to use the Session Initiation Protocol.	<code>chan_local.c</code>
PBX Module	<code>pbx_config.c</code>	Enables the use of extensions from the static file <code>extensions.conf</code>	

Our Voice Mail Asterisk system requires four additional modules to meet its requirements (Table 3). First, the `app_voicemail` module needs to be enabled. A number of other modules are necessary for the `app_voicemail` module to function correctly. For example, this module depends on a resource module called `res_smdi`. This allows Asterisk to interact with the various capabilities of phones and the voice mail system, such as setting a message waiting indicator on a phone. Asterisk provides an archive of royalty-free recordings in the `CORE-SOUNDS-EN-GSM` file (see Appendix A for details on this sounds archive).

When these are installed, the voice mail system uses the recordings to provide incoming callers directions on how to leave and retrieve voice mail messages. The GSM codec is needed to allow Asterisk to play back these prerecorded messages. The `codec_gsm` module enables the GSM codec for use by Asterisk. Similarly, in order to read these files or save new voicemail messages as GSM files, the `format_gsm` module needs to be installed to enable the GSM audio file format interpreter.

Table 3. Modules for Minimized Voice-Call and Voice Mail Asterisk

Module Type	Module Name	Why it is necessary	Dependencies
Application	<code>app_dial.c</code>	Allows an extension to be reached when it is dialed. Enables the <code>Dial()</code> function for use within the dialplan.	
Application	<code>app_voicemail.c</code>	Gives access to voice mail functionality. Enables functions within the dialplan such as <code>VoiceMail()</code> and <code>VoiceMailMain()</code>	<code>res_smdi.c</code>
Channel Driver	<code>chan_local.c</code>	Used internally by channel driver modules.	
Channel Driver	<code>chan_sip.c</code>	Enables Asterisk to use the Session Initiation Protocol.	<code>chan_local.c</code>
Codec Translator	<code>codec_gsm.c</code>	Handles GSM audio format	
Format Interpreter	<code>format_gsm.c</code>	Allows Asterisk to store or play GSM formatted sound files (voice mail messages, and pre-recorded core sound files).	
PBX Module	<code>pbx_config.c</code>	Enables the use of extensions from the static file <code>extensions.conf</code>	
Resource Module	<code>res_smdi.c</code>	Used by voice mail	
Core Sound Files	CORE-SOUNDS-EN-GSM	Archive of prerecorded sound files used for a number of services including voice mail	

E. SUMMARY

We have defined a set of functional requirements and technical requirements for a Voice-Call Asterisk system and a Voice Mail Asterisk system. We described the set of Asterisk modules needed by the minimal Asterisk systems we found to meet these requirements. Next, we describe the experimental process used to discover these minimal systems and verify that they met their respective requirements.

THIS PAGE INTENTIONALLY LEFT BLANK

V. EXPERIMENTATION

A. INTRODUCTION

This chapter describes the experiments performed using candidate Asterisk configurations to verify that they meet our requirements. First, the experimentation platforms are described. In addition, we provide a brief overview of how each candidate system tested was built from its source code (see Appendix A for more specific information). Next, the tests are briefly outlined and described (see Appendix C for more details). Then, we describe how each Asterisk test system was configured for the experiments. After this, the tests are described and our results presented. We conclude with an analysis based on the results of our experiments.

B. EXPERIMENTATION

The minimal set of modules needed for various Asterisk functions was determined based on the requirements identified in Chapter IV. Using these requirements makes it simple to develop tests for our target functionalities in any candidate minimal Asterisk system. The network topology for our experiments is illustrated in Figure 6. Each Asterisk system was installed on a Fedora 10 virtual machine running in VMware Workstation. The directions for building Asterisk on Fedora can be found in Appendix A. Since the experimentation was focused on the functionality of the server, the clients used for the tests only needed to satisfy two requirements. The clients must be able to support a softphone that can use SIP to communicate with a SIP server and use GSM for the audio format. The client machine configuration should be able to be reproduced on any platform, assuming it meets these requirements. For these experiments, the open source Ekiga softphone (v.3.0.1) [37] was installed on Fedora 10 and used for both of the SIP clients (for detailed directions, see Appendix A).

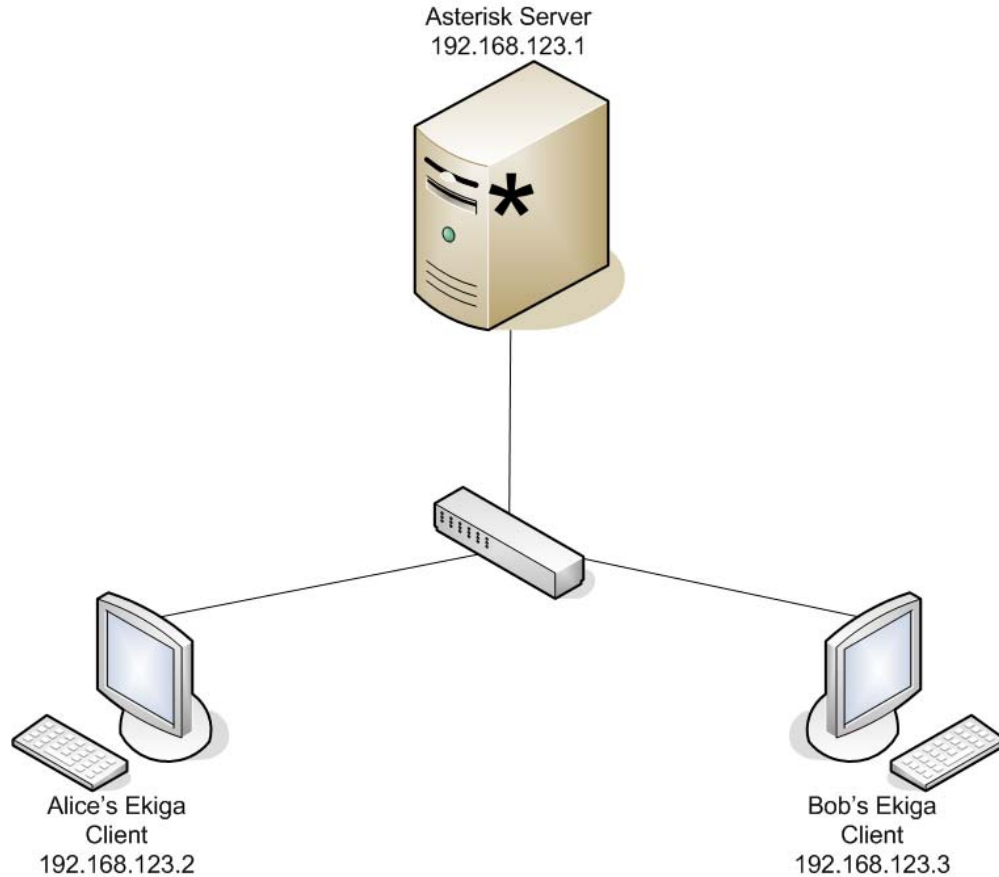


Figure 6. Experimentation network topology

1. Module Selection

Because the Asterisk code is not accompanied by detailed specifications or descriptions of its internal organization, initial candidate systems were built based on the names and available descriptions of modules selected during installation. These names and descriptions, combined with the requirements outlined in Chapter IV, were the basis for choosing modules for inclusion into the systems. Initially, this set of modules was larger than necessary, but using our testing procedures the set was quickly reduced to the minimal required set of modules. Eventually, the target functionality for each Asterisk system was met with regards to the Voice-Call Asterisk and Voice Mail Asterisk system

requirements from Chapter IV. Additionally, no system with fewer modules selected during the build process was able to pass all of our functional testing during experimentation.

2. Build Process

The build process is used to construct an executable file. This file will execute as the Asterisk server. A number of steps are included in this process, but only a brief overview is given in this section. For greater detail about installing Asterisk, see Appendix A.

In order to ensure that Asterisk worked with the selected modules, a new virtual machine for each candidate Voice-Call Asterisk and Voice Mail Asterisk system was created. Additionally, a third virtual machine was created to install a default Asterisk build to be used for comparison purposes. The process for building Asterisk made use of the `/usr/src` directory to decompress the source archive. The resulting directory was then entered. The `make clean` command was then issued to ensure that previous remnants of possible Asterisk installs were removed (since these were pristine VMs, this step was technically unnecessary). Next, `./configure` was run to determine system variables and settings, and to check for dependency requirements. At this point, the actual Asterisk instance was ready for construction. Upon running `make menuselect`, a menu was displayed in the terminal. This interface allows modules to be selected or deselected. All of the modules were deselected, with the exception of the appropriate modules for the desired build. The final modules included in each minimal system are listed in Table 2 and Table 3 in Chapter IV, and detailed installation instructions can be found in Appendix A. These settings were saved and `menuselect` was exited. Finally, the `make install` command was used to install Asterisk. Since the experimentation was confined to a local network, there was no Internet connectivity. This resulted in the need to download the sound files used by the voice mail system on a separate system (with Internet connectivity), and install them into the necessary directory.

Once all of the modules were identified, the Voice-Call Asterisk, Voice Mail Asterisk, and a default configuration of Asterisk were built and installed. This allowed for a comparison of file and process sizes, as well as individual testing. As seen in Table 4, for our final minimized Asterisk systems, there was a significant reduction in the overall size of Asterisk, compared to the default system built. As a note, these sizes and thread counts were obtained from freshly started instances of Asterisk. In addition, there were no calls being handled when the values were reported. The background threads active after startup for Voice Mail Asterisk are listed in Figure 7.

Table 4. Asterisk Build Details

Asterisk System	Resident Set Size	Virtual Size	Threads when Idle	Executable Size
default	7856 Kb	27156 Kb	32	47784 Kb
Voice-Call	2448 Kb	9128 Kb	8	18316 Kb
Voice Mail	2800 Kb	9624 Kb	9	19824 Kb

```

File Edit View Terminal Tabs Help
*CLI> core show threads
0xb7eceb90 monitor_sig_flags   started at [ 3501] asterisk.c main()
0xb7f0ab90 tps_processing_function     started at [  451] taskprocessor.c ast_taskprocessor_get()
0xb7f46b90 do_monitor                   started at [20435] chan_sip.c restart_monitor()
0xb7f82b90 do_parking_thread            started at [ 4546] features.c ast_features_init()
0xb7fbeb90 tps_processing_function     started at [  451] taskprocessor.c ast_taskprocessor_get()
0xb7ffab90 do_devstate_changes         started at [  752] devicestate.c ast_device_state_engine_init()
0xb8036b90 logger_thread               started at [ 1016] logger.c init_logger()
0xb8072b90 listener                     started at [ 1163] asterisk.c ast_makesocket()
0xb80aeb90 tps_processing_function     started at [  451] taskprocessor.c ast_taskprocessor_get()
9 threads listed.
*CLI>

```

Figure 7. Background threads in Voice Mail Asterisk

C. TESTING

A number of tests were performed for both of the minimal Asterisk instances. These tests demonstrated that each configuration met its requirements from Chapter IV. Due to the fact that slightly different features and functions were being tested, the tests for each set of requirements are not identical. The following is a list of pertinent items tested.

- SIP registration
- Users' ability to call each other
- Unanswered calls continuing to ring (Voice-Call Asterisk only)
- Undefined extensions cannot complete a call (exception test)
- With no answer after a certain amount of time, the voice mail system handles the call and allows the user to leave a message
- A message's recipient is able to call the voice mail system to retrieve and manage her mailbox

The testing is outlined in the following tables. Each test is more fully described later in this chapter.

Table 5. Voice-Call Asterisk Requirements Tests

Test Number	Exception Test	Test Description	Test Objective
A1		Register user agent with Asterisk Server.	Determine if SIP is working for registration
A2		Alice calls Bob and vice versa. The call recipient does not answer. The callee's phone continues to ring until the caller hangs up.	Determine if SIP signaling works and ensure call is not handled automatically
A3		Alice calls Bob and vice versa. The call recipient answers.	Determine that SIP signaling works and RTP works in network
A4	X	Users dial undefined extensions.	Ensure that no unintentional extensions can be reached (non-exhaustive)

Table 6. Voice Mail Asterisk Requirements Tests

Test Number	Exception Test	Test Description	Test Objective
B1		Register user agent with Asterisk Server.	Determine if SIP is working for registration
B2		Alice calls Bob, and vice versa.	Determine that SIP signaling works and RTP works in network
B3	X	Users dial undefined extensions.	Ensure that no unintentional extensions can be reached (non-exhaustive)
B4		Alice calls Bob, and the line is not answered by Bob, and vice versa.	Ensure that voice mail system handles incoming call at appropriate time. Ensure codecs and translators work to save a message and play menu instructions.
B5		User calls voice mail system to retrieve mail.	Ensure that voice mail system handles incoming call directly to the voice mail system. Ensure codecs and translators work to save a message and play menu instructions.

Before conducting each of the tests, a number of steps were taken to ensure network connectivity. The client-based firewalls were configured to allow RTP data to pass through the network from one client to the next. Additionally, the physical network infrastructure was tested using the *ping* command against each system's IP address.

1. Voice-Call Asterisk Testing

So that the test VoIP network could function in the desired way, a simple scheme was developed to configure each candidate Asterisk system. This scheme consists of a number of configuration files, which create the logical configuration that makes up the actual phone extensions. For basic voice calls, three configuration files were necessary: `modules.conf`, `extensions.conf`, and `sip.conf`. These files will be explained in the next subsection.

a. Configuration Files

In order for Asterisk to "know" how to react to various types of requests (e.g., a SIP INVITE message), it must contain properly formatted

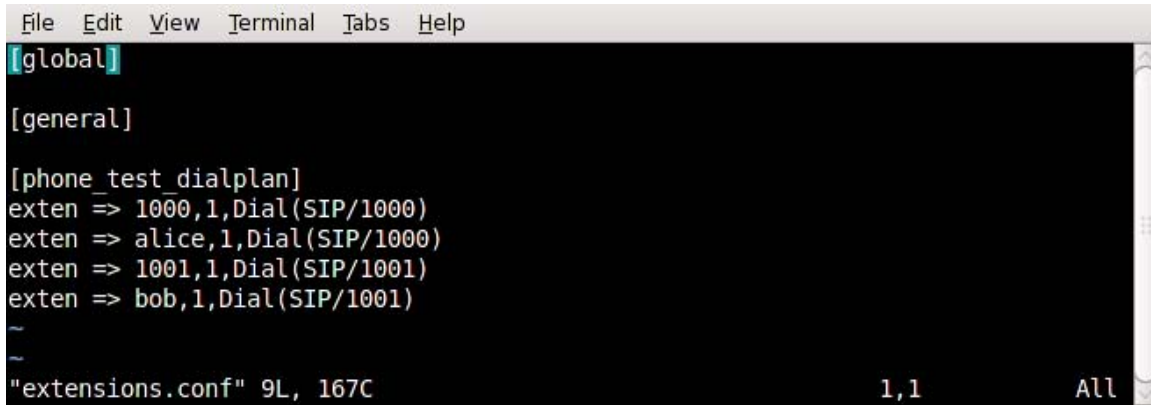
configuration files. Each candidate Asterisk system we tested required a number of these configuration files. First, `modules.conf` is needed as a technicality. While removing the Asterisk modules that are extraneous to the current study, it was determined that the functionality of `pbx_config` was not automatically enabled, even though the module was embedded into the executable during the build process. This module, `pbx_config`, must be loaded because it helps Asterisk to parse text files. To ensure availability of this functionality, `modules.conf` loads `pbx_config` using the `autoload` directive (see Figure 8) which loads the contents of the `/usr/lib/asterisk/modules` directory and loads the `pbx_config` module.

A screenshot of a text editor window. The title bar shows 'File Edit View Terminal Tabs Help'. The main text area contains the configuration for the pbx_config module: `[modules]` on the first line, `autoload=yes` on the second line, and a tilde character `~` on the third line. The status bar at the bottom indicates the file is `"modules.conf"` at line 2, column 24. The cursor is at line 1, column 1.

Figure 8. Voice-Call and Voice Mail Asterisk's `modules.conf`

Asterisk's dialplan is contained in the `extensions.conf` file. For an overview of the role of the dialplan in Asterisk, see Chapter IV. For testing purposes, a very basic dialplan (see Figure 9) was developed to test the necessary functionality and to keep unnecessary complexity to a minimum. In our dialplan, we define a "context" called `phone_test_dialplan`. Within this context there are two possible destination extensions, with two ways to reach each. The `"exten =>"` syntax points to each of the extensions that will be reached as this context is entered. The first argument after this defines how the phone will be reached, either by dialing 1000 or entering `alice`³. Finally, the `Dial()` function is invoked. Together these directives state that when extension 1000 is registered from the `phone_test_dialplan` context, extension 1000 on the SIP channel will attempt to be contacted.

³ SIP supports the ability to make a call using either a numbered extension (1000) or a name (`alice`); the ability to call using a name was provided for illustrative purposes.



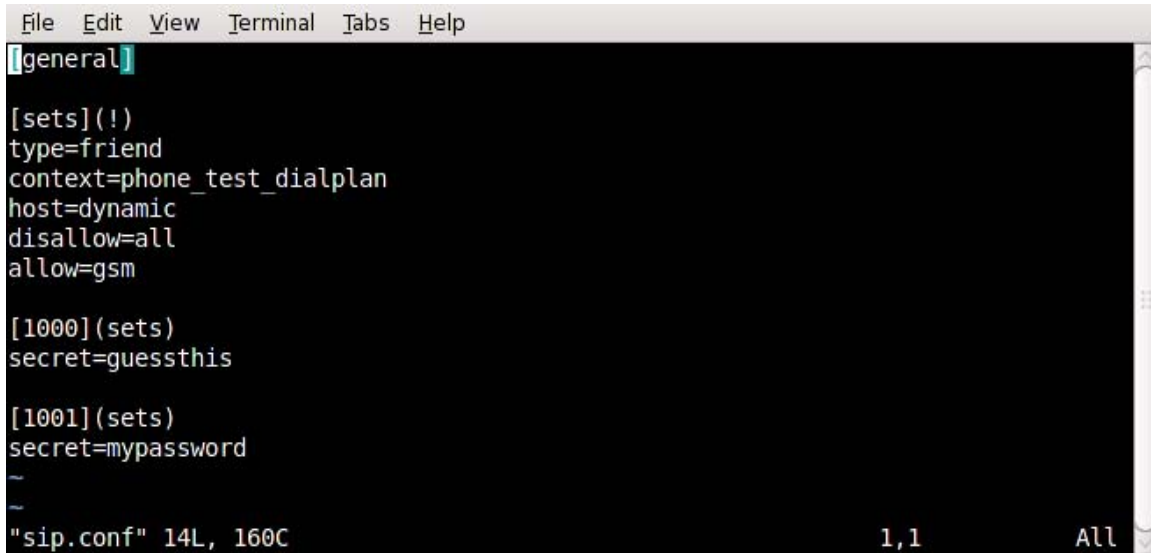
```
File Edit View Terminal Tabs Help
[global]

[general]

[phone_test_dialplan]
exten => 1000,1,Dial(SIP/1000)
exten => alice,1,Dial(SIP/1000)
exten => 1001,1,Dial(SIP/1001)
exten => bob,1,Dial(SIP/1001)
~
~
"extensions.conf" 9L, 167C 1,1 All
```

Figure 9. Voice-Call Asterisk's extensions.conf file

Since the dialplan indicates that SIP channels are being accessed (see next paragraph), the users that correspond to the extensions on this channel need to be set up in sip.conf (see Figure 10). In this file, we created a generic template context entitled "sets." The template context is later referenced by the individual extensions using the (sets) syntax. It is as though the whole block was entered for each extension created. The type "friend" allows each user to both send and receive calls from the Asterisk server [33]. The value for "context" refers to the dialplan context into which the SIP user should enter by default when using this channel. In our example, it is the same for both users. The host variable "dynamic" signifies that the host's IP address is dynamically configured on the network. The "disallow" and "allow" lines relate to the codecs. First, all codecs are disallowed and then GSM is the sole audio codec enabled. Finally, the variable secret defines the user's password for access to the channel. In our example, each user has a unique password. These entries relate to Alice and Bob, respectively, as can be confirmed by the configuration of their extensions in Figure 9.



```
File Edit View Terminal Tabs Help
[general]
[sets](!)
type=friend
context=phone_test_dialplan
host=dynamic
disallow=all
allow=gsm

[1000](sets)
secret=guessthis

[1001](sets)
secret=mypassword
~
~
"sip.conf" 14L, 160C 1,1 All
```

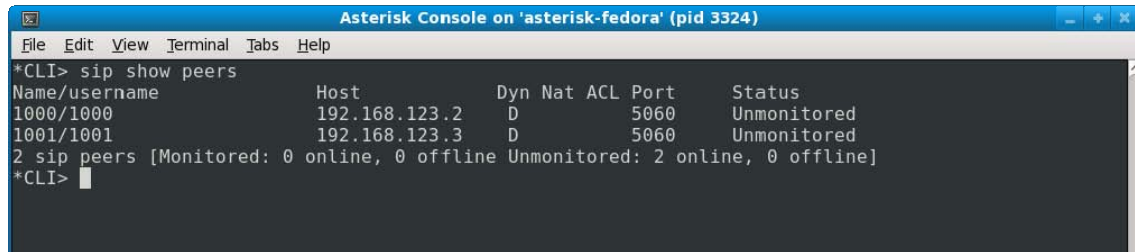
Figure 10. Voice-Call Asterisk's sip.conf file

b. Description of Tests

All of the tests proceeded using this configuration for the Voice-Call Asterisk (see Appendices A and C for details). All of the tests were performed after first invoking Asterisk from the command line using the “-vvvvc” switch⁴.

For test A1, the softphone attempts to register with Asterisk. Although the user enters the extension being registered and her password (corresponding to the password from the sip.conf file) into the SIP configuration section of the softphone, the actual registration is transparent to the user. Because of this, the command line interface (CLI) command *sip show peers* was issued to see if the given user was indeed registered (see Figure 11). Additionally, a message indicating that a registration has been made is displayed. However, the amount of information shown is limited and only identifies the registered user's extension and IP address. Test A1 completed successfully.

⁴ The Asterisk command line switch “-v” signifies the program's level of verbosity, or the amount of feedback it displays to the screen. Successive “v” characters will increase the verbosity level of the starting Asterisk process. The increased verbosity in Asterisk's feedback is helpful for providing information about various tests.



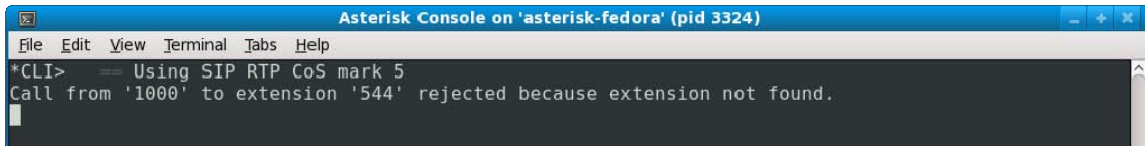
```
Asterisk Console on 'asterisk-fedora' (pid 3324)
File Edit View Terminal Tabs Help
*CLI> sip show peers
Name/username      Host          Dyn Nat ACL Port      Status
1000/1000          192.168.123.2 D          5060     Unmonitored
1001/1001          192.168.123.3 D          5060     Unmonitored
2 sip peers [Monitored: 0 online, 0 offline Unmonitored: 2 online, 0 offline]
*CLI>
```

Figure 11. CLI sip show peers output

Next, Alice called Bob and his softphone, although registered, was not answered. This test cannot be exhaustive since the phone can only ring a finite number of times. However, the line continued to ring until the caller hung up since there was nothing to handle the incoming call. This test was then repeated in the opposite direction such that Bob called Alice. For both directions, test A2 completed with expected results.

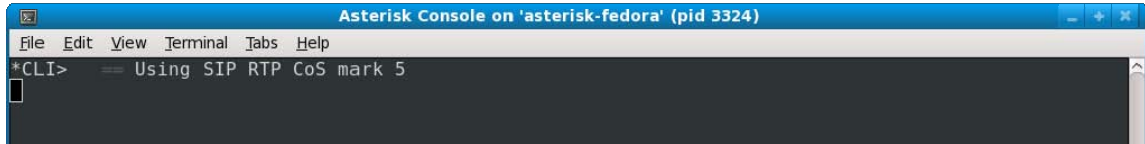
Next, Alice called Bob and the softphone on the receiving end was answered. The voice could be heard and detected from each softphone, hence, test A3 passed. Again, this test was performed successfully in the opposite direction, with Bob calling Alice.

Finally, the exception test was performed to verify that calls to undefined extensions cannot be made. This test tried a handful of various undefined extensions. The test results were confirmed both at the client and the command line interface. The behavior observed from the client-side is as follows: If the number was invalid, shortly after the call was sent, it would be hung up, without ringing. Confirmation was provided at the command line in the form of a message that stated the call could not be completed because the given extension does not exist (see Figure 12). This error is generated for all numbers except the subset of numbers associated with valid extensions. For example, although extension 100 is not a valid extension the output is slightly different (see Figure 13). Since 100 is a prefix of Alice's extension (1000) Asterisk is unable to disambiguate this as an undefined extension or a user getting ready to dial extension 1000. The same behavior is observed for extensions 1 and 10.



```
Asterisk Console on 'asterisk-fedora' (pid 3324)
File Edit View Terminal Tabs Help
*CLI> == Using SIP RTP CoS mark 5
Call from '1000' to extension '544' rejected because extension not found.
```

Figure 12. Error associated with dialing extension “544”



```
Asterisk Console on 'asterisk-fedora' (pid 3324)
File Edit View Terminal Tabs Help
*CLI> == Using SIP RTP CoS mark 5
```

Figure 13. Error associated with dialing extension “100”

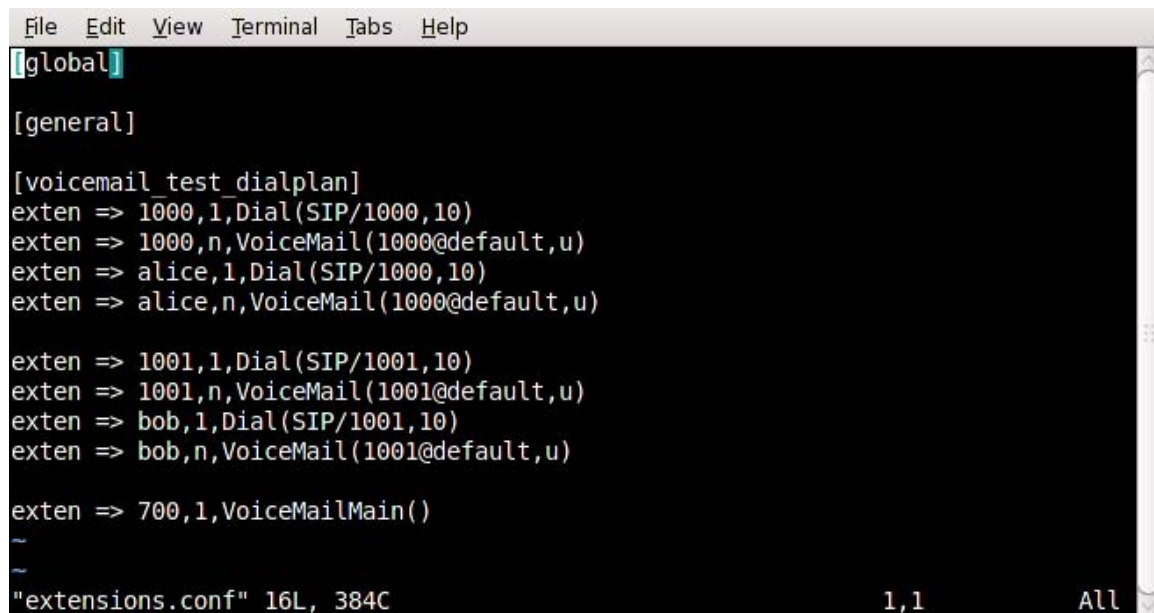
2. Voice Mail Asterisk testing

Tests of Asterisk’s voice mail system differ from those performed in voice-call testing. The overall process was the same as the Voice-Call Asterisk testing, but some of the configuration files were different and additional tests were performed. In addition to the configuration files used for the voice-call tests, a voicemail.conf file had to be configured. Some minor changes to the configurations of the extensions.conf and sip.conf files were also needed. The next subsection describes the configuration files and the changes from those used for Voice-Call Asterisk.

a. Configuration files

In the extensions.conf file, the change needed for voice mail testing is an additional line for each possible user and a subtle change to the arguments of the Dial() function. The argument added to the Dial() function for each of the extensions represents the number of seconds that the call will be attempted. As illustrated in Figure 13, in our configuration Asterisk will ring the extensions for ten seconds. If the call is not answered within this time, the dialplan proceeds to the next line. The “exten => 1000”, is the same as the previous line. The n represents the next step of the dialplan to perform; this notation helps avoid renumbering when adding steps to the dialplan (in our example, n represents 2). Next, the VoiceMail() function takes two arguments. The first argument,

1000@default, indicates the target mailbox (mailbox 1000 in “default” context of the voicemail.conf file). With this argument, the VoiceMail() function knows where to store the collected message. The second argument can be either “u” or “b” and signifies to the VoiceMail() function whether to play the unavailable or busy message for the user [33]. When this line is reached, Asterisk answers the call and allows the caller to save a message for the unanswered extension. Another addition to this configuration file is a new extension (indicated as extension 700 in Figure 14). Although it has the same format as other extensions, it does not result in dialing a channel extension, but instead results in the invocation of the VoiceMailMain() function. Through the invocation of this function, a user is prompted to login and is able to manage personal voice mail.



```
File Edit View Terminal Tabs Help
[global]
[general]
[voicemail_test_dialplan]
exten => 1000,1,Dial(SIP/1000,10)
exten => 1000,n,VoiceMail(1000@default,u)
exten => alice,1,Dial(SIP/1000,10)
exten => alice,n,VoiceMail(1000@default,u)

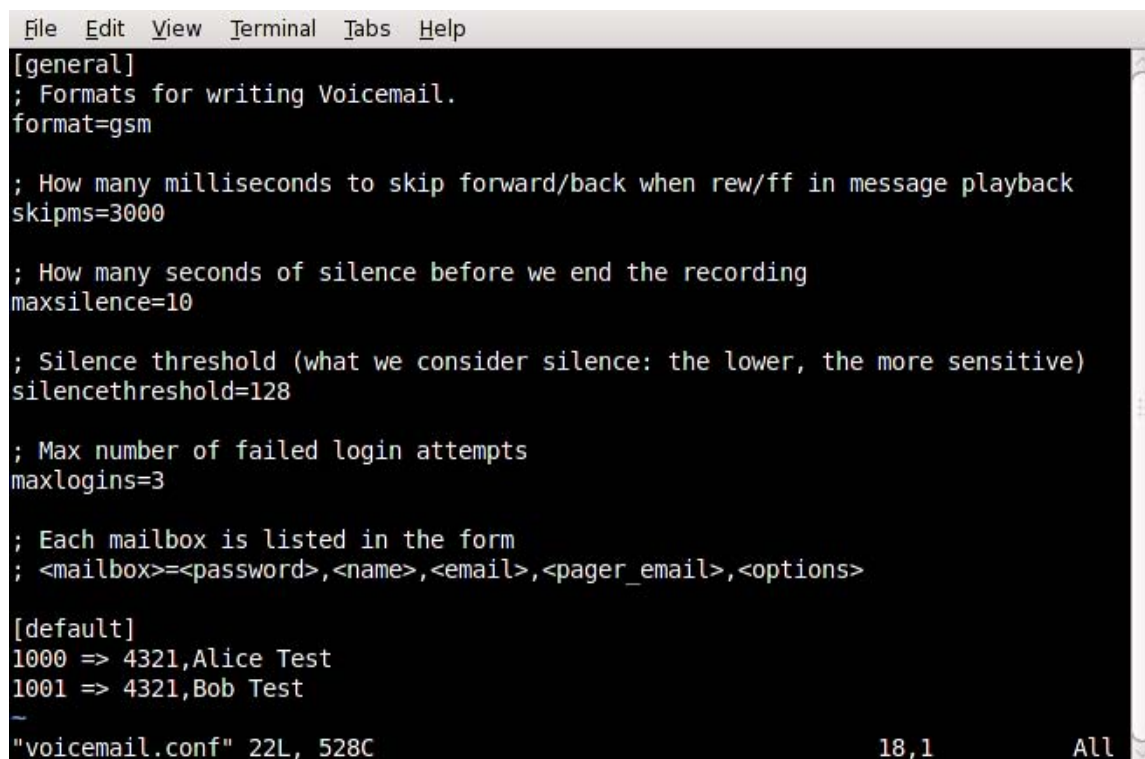
exten => 1001,1,Dial(SIP/1001,10)
exten => 1001,n,VoiceMail(1001@default,u)
exten => bob,1,Dial(SIP/1001,10)
exten => bob,n,VoiceMail(1001@default,u)

exten => 700,1,VoiceMailMain()
~
~
"extensions.conf" 16L, 384C 1,1 All
```

Figure 14. Voice Mail Asterisk’s extensions.conf file

Settings for the voice mail system include a number of items in the general context that define a number of variables and settings. First, “format=gsm” indicates that incoming voice mail messages should be saved in the GSM format. This is necessary, based on the requirement that this audio format is to be used for sound files. The next four settings come from the default voice mail settings shipped with Asterisk’s sample configuration files. As described in the comments of the file, “skipms=3000,” configures the system to

skip ahead or back 3000 milliseconds when a message is being fast-forwarded or rewind. The variable “maxsilence” is used to determine how many seconds of silence are detected by Asterisk (i.e., the caller is no longer speaking) before a recording is ceased. It is set to 10 seconds. Next, the default value for the threshold of silence is set at 128, a value based on the algorithm used by Asterisk to determine silence. The threshold of silence becomes more sensitive with lower values. The last variable is the number of incorrect login attempts, which is set at “maxlogins=3.” Inside the “default” context, we create the mailbox 1000 using the notation “1000 =>” with the mailbox password (4321) as the first argument (see Figure 15). The second argument is simply the user’s name. In the sip.conf file, we allow a user to access mailbox 1000 by adding the line “mailbox=1000” under that user’s configuration data (see Figure 16).



```
File Edit View Terminal Tabs Help
[general]
; Formats for writing Voicemail.
format=gsm

; How many milliseconds to skip forward/back when rew/ff in message playback
skipms=3000

; How many seconds of silence before we end the recording
maxsilence=10

; Silence threshold (what we consider silence: the lower, the more sensitive)
silencethreshold=128

; Max number of failed login attempts
maxlogins=3

; Each mailbox is listed in the form
; <mailbox>=<password>,<name>,<email>,<pager_email>,<options>

[default]
1000 => 4321,Alice Test
1001 => 4321,Bob Test
~
"voicemail.conf" 22L, 528C 18,1 All
```

Figure 15. Voice Mail Asterisk’s voicemail.conf file

```
File Edit View Terminal Tabs Help
[general]
[sets](!)
type=friended
context=voicemail_test_dialplan
host=dynamic
disallow=all
allow=gsm

[1000](sets)
secret=guessthis
mailbox=1000

[1001](sets)
secret=mypassword
mailbox=1001
~
~
"sip.conf" 16L, 190C 1,1 All
```

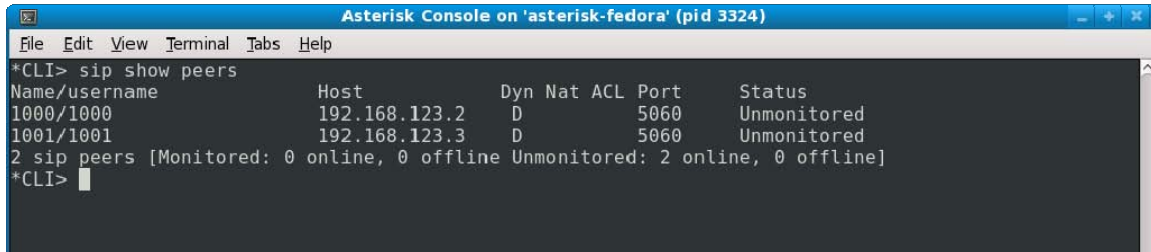
Figure 16. Voice Mail Asterisk's sip.conf file

With the aforementioned configuration, two extensions were created, each with a mailbox. One was created for Alice and one for Bob. Another extension was created for the voice mail system. This allowed users to dial into the system and retrieve and manage their messages. Tests proceeded in an environment based on this configuration.

b. Description of tests

Test B1 works in precisely the same manner as test A1. The Asterisk server was again started in the same manner as the first set of experiments, by using the “-vvvvvc” switch to have sufficient feedback. After a user configures her SIP user agent, assuming the network is functioning correctly, the softphone can register with the Asterisk server. The SIP command *sip show peers* issued (see Figure 17) to the Asterisk command line interface was used to list registered clients. This test completed successfully, verifying that both Alice's and Bob's softphones were successfully registered at their respective extensions. Likewise, test B2 completed in a manner similar to A3 with calls being placed to each user. Test B3 completed with undefined

extensions being called, in the same way test A4 was executed. Based on these results, it was determined that, with the configuration changes made to enable voice mail, SIP was working properly, RTP data was successfully traversing the network, and unexpected extensions could not be reached.



```
Asterisk Console on 'asterisk-fedora' (pid 3324)
File Edit View Terminal Tabs Help
*CLI> sip show peers
Name/username      Host           Dyn Nat ACL Port      Status
1000/1000          192.168.123.2 D           5060     Unmonitored
1001/1001          192.168.123.3 D           5060     Unmonitored
2 sip peers [Monitored: 0 online, 0 offline Unmonitored: 2 online, 0 offline]
*CLI>
```

Figure 17. CLI command sip show peers

The remaining tests check the functionality of the voice mail system. Test B4 is similar to test A2, in that the user does not answer the ringing line. However, after the predetermined time (i.e., 10 seconds) the dialplan directs the voice mail system to answer the call and allows the caller to record a message. After a message is left, the corresponding message file can be found in a standard directory. The `/var/spool/asterisk/voicemail/[cntxt]/[ext]/INBOX/` directory is the default for saved voice mail. This location can be found when “[cntxt]” is replaced by the corresponding context from the `voicemail.conf` file and “[ext]” is replaced by the recipient’s extension. As an example, Alice’s mailbox, based on the presented configuration files, is the directory `/var/spool/asterisk/voicemail/default/1000/INBOX/`, which contains Alice’s newly received voice messages.

Test B5 tests a user’s ability to call into the voice mail system and retrieve and manage voice mail messages. After being prompted for a mailbox and password, the user was able to manage personal voice mail. The only functions examined were the ability to playback and delete messages (see Appendix C). With all of the tests complete, it was concluded that our final candidate Voice Mail Asterisk system met the requirements from Chapter IV.

D. ANALYSIS AND RECOMMENDATIONS

1. Analysis of Minimal Asterisk

Based on the information gathered, the desired functionality for the simple SIP-based VoIP implementation was attained. In our final Voice-Call Asterisk system, users are able to make calls to one another using the SIP protocol. In our Voice Mail Asterisk system, users have the additional ability to send voice mail messages to, or receive messages from, other users. These messages are stored and managed in mailboxes, accessible to the user when she dials a special extension in the system and provides the correct credentials.

While the functionality meeting our requirements was verified to be available in our systems, it is not immediately clear what additional functionality may also be available within the systems. During the testing, when the Asterisk server was started and examined, it appeared that some features were available and threads were active that did not correspond with those modules explicitly selected during the installation. Although some Asterisk functionality appears to be present and running in systems with no modules loaded, we speculate that this functionality is internal to Asterisk and cannot be removed without source code modifications.

2. Recommendations

Having completed this analysis and identification of modules meeting the required core capabilities, Asterisk should be more fully dissected. This could help identify potential functionality that is not necessary and could be removed by modifying Asterisk's source code. The number of extensions that can be configured should be tested. Network traffic that can be supported with a single Asterisk server on the VoIP network should also be examined. Such an analysis could provide insight into an upper bound of concurrent conversations over the VoIP network. Additionally, it may be beneficial to conduct more thorough

exception testing. This could provide more assurance that there are not arbitrary extensions that exhibit undesired effects or enable hidden functionality.

E. SUMMARY

After successful completion of all tests, the possibility of providing a minimized set of Asterisk VoIP functions for porting to a MLS environment is realized. Although many useful features have not been examined, the core functionality of a VoIP implementation using SIP and RTP, as identified in the requirements, has been identified. The modules necessary for simple voice and voice mail communication provide a starting point for any future investigation whose ultimate goal is to support an Asterisk-based VoIP implementation in MYSEA.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. FUTURE WORK AND CONCLUSION

A. INTRODUCTION

With the determination that it is possible to minimize the functionality of Asterisk and build a minimal working PBX, the field lies open for future work. This chapter discusses some ideas for continuing to work with Asterisk as a candidate for achieving VoIP as a service within MYSEA. We then provide some brief concluding remarks, summarizing our research effort.

B. FUTURE WORK

1. Real-time Voice Communication within MLS Context

Some work needs to be done to port the Voice-Call and Voice Mail Asterisk systems to run in MYSEA. Through this research, the modules necessary for Asterisk to meet core requirements for our concept of operation were identified. Having identified these key modules, the technical work to port Asterisk components to operate in MYSEA may begin, with a well-designed, initial focus.

The recreation of voice call and voice mail capability within MYSEA would also require implementation of a VoIP infrastructure that takes mitigations of known security issues into account. One portion of these security issues is the potential of misconfigured or erroneous Asterisk configuration files. Creating a secure set of Asterisk configuration files that scales with the number of MYSEA users would be nontrivial.

2. Voice Communications Originating Outside the MLS Network

Another area of work that can be explored is the possibility of receiving calls from standard telephone lines originating outside of the MYSEA architecture. This would require enabling extra Asterisk functionality compared

to the systems we consider in this work, since the drivers and hardware that interact with standard phone lines would need to be accessible to Asterisk. This is a considerably more difficult task, but the potential benefits of connecting Asterisk to the PSTN make it a natural object of future study.

3. Voice Communications between MLS Enclaves

Once Asterisk is ported to run within MYSEA, research should be done to determine how two MYSEA networks could share their voice communication. Perhaps each enclave would have an Asterisk server inside its network perimeter that connects each classification level to the MYSEA network. Each server would be able to receive calls at its classification level from remote MLS networks that have VoIP capabilities. With further research and development, it could even become a nearly “plug and play” feature to help simplify voice intercommunication with other MYSEA enclaves.

C. CONCLUSION

The goal of this research was to determine a set of requirements associated with VoIP services that includes voice mail. An instance of Asterisk was produced that exhibits functionality meeting these requirements. Based on a natural division of our requirements (voice calls and voice calls with voice mail support), it was determined that two minimal Asterisk systems should be created: one meeting our voice call requirement and another that expands these, satisfying our voice mail requirement. Candidate systems were then tested and found to provide the required functionality. The Asterisk modules required to create these system were identified. Those modules required by our minimal systems are described in Chapter IV. This goal was established with the future vision to port Asterisk to run within MYSEA. Overall, our goal was accomplished ,and the groundwork was laid to aid with the future use of Asterisk within the MYSEA architecture.

APPENDIX A: INSTALLATION PROCEDURES

This appendix focuses on installing a minimal version of Asterisk. This installation is based on Asterisk 1.6.1.0 running on Fedora 10. These procedures assume a basic knowledge of Linux command-line administration including networking, traversing the file system, and using text editors.

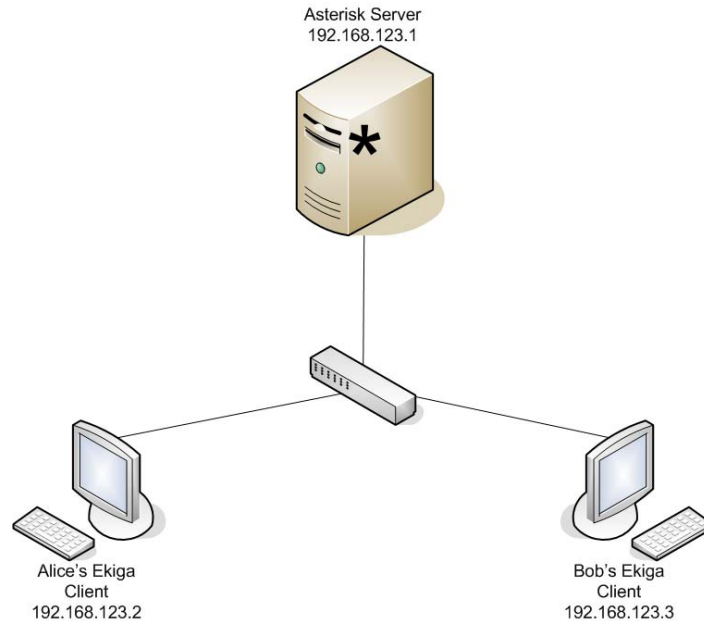


Figure 18. Test Network Topology

A. PREPARING FEDORA 10 FOR THE ASTERISK SERVER

This is to prepare Fedora 10 for an installation of Minimal Asterisk. After the file system has been formatted, the user will be presented with the option to select packages or to use default packages. Select the radio button “Customize Now” to be able to select the desired packages. These directions will work with at least these broad categories in addition to several specific packages:

- Applications—Editors (vim)—Select the editor most comfortable for the tester
- Base System—Base

Once installed, the following packages are needed to be able to compile Asterisk:

- gcc (yum install gcc—installs the following: glibc-headers, kernel-headers, glibc-devel, glibc, gcc, glibc-common),
- c++ support (yum install gcc-c++—installs the following: gcc-c++, libstdc++-devel)
- ncurses-devel (yum install ncurses-devel)—fixes "configure: error: *** termcap support not found"

B. INSTALLING MINIMAL ASTERISK WITH VOICE CALL CAPABILITY

This section assumes the steps from the section “Preparing Fedora 10” have been followed”.

- Download Asterisk 1.6.1.0 from <http://downloads.asterisk.org/pub/telephony/asterisk/releases/asterisk-1.6.1.0.tar.gz> (This can be done from another system and imported using an external storage medium such as CD, DVD, or USB drive) use wget or if you have installed the server with a graphical user interface you can browse to it using a web browser.

From this point on run as root

- Save or copy the file to /usr/src (command: cp asterisk-1.6.1.0.tar.gz /usr/src)
- Extract the archive data within /usr/src (command: tar xfvz asterisk-1.6.1.0.tar.gz). This command will create a directory called asterisk-1.6.1.0
- Move into the created directory (command: cd asterisk-1.6.1.0)

From here on, run the commands in this directory `/usr/src/asterisk-1.6.1.0`

- `./configure`
- `make`
- `make menuselect`
- `menuselect` options will be presented, go through and deselect ALL items from all modules.
- Select only the following items within `menuselect`:
 - Applications: `app_dial.c`
 - Channel Drivers: `chan_local.c`
 - Channel Drivers: `chan_sip.c`
 - PBX: `pbx_config.c`
 - Module embedding: APPS
 - Module embedding: CHANNELS
 - Module embedding: PBX
- Now select “Save & Exit”
- Compile and create the asterisk executable (command: `make install`)
- Confirm that Asterisk is installed by running the following commands
 - `which asterisk`—Result should be `“/usr/sbin/asterisk”`
 - `asterisk -V`—Result should be `“Asterisk 1.6.1.0”`
- In order to run, three configuration files need to be saved. These files create two extensions, one for Alice and one for Bob. It will allow them to call one another.

- Work from within the `/etc/asterisk` directory. (command: `cd /etc/asterisk`)
- Using Appendix B, create the three configuration files needed for the direct call sample. Use the following commands to create the files:
 - `touch extensions.conf`
 - `touch modules.conf`
 - `touch sip.conf`
- Now, using a preferred text editor, add the lines from Appendix B to the corresponding file. At a minimum, only the uncommented lines are necessary to get Asterisk working. The following is an example of editing these files using the vi text editor.
 - `vi extensions.conf`
 - `i` (enters insert mode)
 - Type or paste the text of the file.
 - Hit the ESC key. This causes vi to run in the command mode, no text can be inserted. The following keystrokes will save changes and exit: `“:wq”`.
- Follow the previous four steps for each configuration file, replacing the name of the configuration file when starting the vi editor.
- Ensure that the software based firewall is disabled (command: `service iptables stop`) (To disable the firewall from startup issue this command: `chkconfig - -level 345 iptables off`)

- Set the server's IP address (command: `ifconfig eth0 192.168.123.1 netmask 255.255.255.0`). For the `ifconfig` command, the network interface may be different from `eth0`, depending on various linux configurations, ensure that the correct interface's IP address is set (Configuration dependent).
- To start Asterisk, from the command line, invoke the command `asterisk -vvvvvc`. This will connect with the command line interface to the Asterisk process that starts. Each `v` represents a level of verbosity which, in this case, is set to level 5.

C. INSTALLING MINIMAL ASTERISK WITH VOICE MAIL CAPABILITY

- This section assumes the steps from the section "Preparing Fedora 10" have been followed.
- Download Asterisk 1.6.1.0 from <http://downloads.asterisk.org/pub/telephony/asterisk/releases/asterisk-1.6.1.0.tar.gz> - (This can be done from another system and imported using an external storage medium such as CD, DVD, or USB drive) use `wget` or if you have a gui you can browse to it using a web browser. (From this point on run as root) Save it to `/usr/src`
- For voice mail, prerecorded media files are needed. These can be downloaded from: <http://downloads.asterisk.org/pub/telephony/sounds/asterisk-core-sounds-en-gsm-current.tar.gz> - These will be moved later in the directions
- Extract the archive data within `/usr/src` using: `tar xfvz asterisk-1.6.1.0.tar.gz`, this will create a directory called `asterisk-1.6.1.0`
- `cd asterisk-1.6.1.0`
- From here on, run the commands in this directory `/usr/src/asterisk-1.6.1.0`

- ./configure
- make
- make menuselect
- menuselect options will be presented, go through and deselect ALL items from all modules.
- Select only the following items within menuselect:
 - Applications: app_dial.c
 - Applications: app_voicemail.c
 - Channel Drivers: chan_local.c
 - Channel Drivers: chan_sip.c
 - Codec Translators: codec_gsm.c
 - Format Interpreters: format_gsm.c
 - PBX: pbx_config.c
 - Resource Modules: res_smdi.c
 - Module embedding: APPS
 - Module embedding: CHANNELS
 - Module embedding: CODECS
 - Module embedding: FORMATS
 - Module embedding: PBX
 - Module embedding: RES
- Select "Save & Exit"
- Compile and create the asterisk executable (command: make install).

- Confirm that Asterisk is installed by running the following commands
 - which asterisk—Result should be “/usr/sbin/asterisk”
 - asterisk-V—Result should be “Asterisk 1.6.1.0”
- In order to run, four configuration files need to be created.
- Work from within the /etc/asterisk directory. (command: `cd /etc/asterisk`)
- Using Appendix B, create the four configuration files needed for Voice Mail Asterisk. Use the following commands to create the files:
 - `touch extensions.conf`
 - `touch modules.conf`
 - `touch sip.conf`
 - `touch voicemail.conf`
- Now, using a preferred text editor, add the lines from Appendix B to the corresponding file. At a minimum, only the uncommented lines are necessary to get Asterisk working. The following is an example of editing these files using the vi text editor.
 - `vi extensions.conf`
 - `i` (enters insert mode)
 - Type or paste the text of the file.
 - Hit the ESC key. This causes vi to run in the command mode, no text can be inserted. The following keystrokes will save changes and exit: `“:wq”`.
- Follow the previous four steps for each configuration file, replacing the name of the configuration file when starting the vi editor.

- Change directories into `/var/lib/asterisk/sounds/en` (command `cd /var/lib/asterisk/sounds/en`). Create it if it does not exist (command: `mkdir -p /var/lib/asterisk/sounds/en`).
- Extract the archive of sounds into the current working directory (command: `pwd` should return the following directory `/var/lib/asterisk/sounds/en`) (command: `tar xfvz asterisk-core-sounds-en-gsm-current.tar.gz`). The voice mail application looks for the prerecorded menu selections within this folder.
- Ensure that the software based firewall is disabled (command: `service iptables stop`) (To disable the firewall from startup issue this command: `chkconfig --level 345 iptables off`)
- Set the server's IP address (command: `ifconfig eth0 192.168.123.1 netmask 255.255.255.0`). For the `ifconfig` command, the network interface may be different from `eth0`, depending on various linux configurations, ensure that the correct interface's IP address is set (Configuration dependent).
- To start Asterisk, from the command line, invoke the command `asterisk -vvvvvc`. This will connect with the command line interface to the Asterisk process that starts. Each `v` represents a level of verbosity, this is set to level 5.

D. INSTALLATION OF CLIENT COMPUTERS

The tests undertaken herein used two Fedora 10 clients. Unlike the Asterisk server, these were tested as normal workstations. The following describes the procedures to setup these computers for use as clients with Ekiga, the softphone application used for testing. Both clients were used separately to connect to the Asterisk servers. For example, when the Voice-Call Asterisk was not used, the server was shut down and the voice mail server was started. They both have the same IP address so needed to be used one at a time.

- Complete a default install of Fedora 10 using the install media (CD or DVD). No additional packages or libraries are needed as the default install includes Ekiga 3.0.1.
- After reboot, create a user with normal privileges (e.g., Alice or Bob).
- Login as the newly created user

1. **Disable iptables and configure the network**

- Open a terminal and su to the root account (command: `su -`)
- Run the command `service iptables stop`, to turn off the software based firewall
- Set the client's IP address (command to be issued on Alice's station: `ifconfig eth0 192.168.123.2 netmask 255.255.255.0`) (command to be issued on Bob's station: `ifconfig eth0 192.168.123.3 netmask 255.255.255.0`). For the `ifconfig` command, the network interface may be different from `eth0`, depending on various Linux configurations, ensure that the correct interface's IP address is set (Configuration dependent).
- Test that Alice and Bob can ping each other and the server. Test that the server can ping Alice and Bob's computers. (command: `ping` corresponding IP address).

2. **Run Ekiga and configure the client**

- Before starting the clients, they need to have a server to which to connect. For each test, start the corresponding server (i.e. direct call or voice mail) by issuing the following command
 - `asterisk -vvvvvc`

- Click on Applications > Internet > IP Telephony, VoIP and Video Conferencing (Ekiga's icon, as of this writing, is yellowish circle with a phone receiver and a sound wave on it).
- When Ekiga starts up for the first time, it presents the user with a Configuration Assistant. The Configuration Assistant can either be canceled or to prevent it from starting up each time Ekiga starts, quickly go through the guide with the following actions.
 - On screen 1 of 8, select the Forward button.
 - Screen 2, enter the full name (ex. Alice User).
 - Screen 3, select the checkbox at the bottom of the screen stating, "I do not want to sign up for the ekiga.net free service."
 - Screen 4, select the checkbox at the bottom of the screen stating, "I do not want to sign up for the Ekiga Call Out service."
 - Screen 5, select LAN for the connection type.
 - Screen 6, be sure all audio settings are set to Default.
 - Screen 7, select forward.
 - Screen 8, select Apply.
- Configure Ekiga to use the same codec as the Asterisk installation (GSM). Select the Edit menu and Preferences. On the left-hand side, find Audio. If the audio section is not expanded click the triangle next to it to expand the section. Select codecs from under audio. Uncheck all of the codec options except gsm. Note: gsm and ms-gsm are not the same. Be sure that gsm is selected and not ms-gsm.

- Once Ekiga starts, register by clicking on the Edit menu and selecting Accounts.
- In the accounts window, select the Accounts menu and Add a SIP account. Use the following information, as specified in the configuration files, to register the users.
 - Alice's settings are as follows
 - Name: Alice
 - Registrar: 192.168.123.1
 - User: 1000
 - Authentication user: 1000
 - Password: guessthis
 - Timeout: Leave at default setting
 - Bob's settings are as follows
 - Name: Bob
 - Registrar: 192.168.123.1
 - User: 1001
 - Authentication user: 1001
 - Password: mypassword
 - Timeout: Leave at default setting
- Click OK
- Click close on the Accounts window
- If the client is properly set up, a message similar to following should appear at the Asterisk command line interface:

Registered SIP '1000' at 192.168.123.2 port 5060

Saved useragent "Ekiga/3.0.2" for peer 1000

(This example is based on registering extension 1000 with an Ekiga softphone)


```
[general]

[phone_test_dialplan]
exten => 1000,1,Dial(SIP/1000)
exten => alice,1,Dial(SIP/1000)
exten => 1001,1,Dial(SIP/1001)
exten => bob,1,Dial(SIP/1001)
```

modules.conf

```
;/;/;/
;
; Filename: /etc/asterisk/modules.conf
; Author: Jeff Wiley
; Date: 29 June 2009
;
; Purpose: Load available modules. Although the documentation
; says this loads modules from the
; /usr/lib/asterisk/modules directory, it has been
; noted that even when there are no modules in this
; directory, asterisk will not work unless the
; modules used in asterisk are loaded, like this one,
; individually, or automatically, as the ones selected
; during the make menuselect process.
;
;/;/;/

[modules]
    autoload=yes
```

sip.conf

```
;/;/;/
;
; Filename: /etc/asterisk/sip.conf
; Related files: /etc/asterisk/extensions.conf
; Author: Jeff Wiley
; Date: 11 July 2009
;
; Purpose: This file defines the necessary elements for
; Asterisk to act like a SIP server. This files
; creates two possible extensions that can be
; registered namely, 1000 and 1001. Both of these
; use the [sets] section and the variable
; choices placed therein.
;
;/;/;/
;
; Description of selections:
;
; type=friends
; means that the registered users can both send
; and receive calls
; context=phone_test_dialplan
; corresponds to the context from the extensions
; file
; host=dynamic
; means the user can register from any IP address
; disallow=all
; disables all of the codecs from being used on
; this channel
; allow=gsm
; as has been configured in the installation,
; this further enforces the use of the GSM codec
```


sip.conf

```
;/;/
;
; Filename: /etc/asterisk/sip.conf
; Related files: /etc/asterisk/extensions.conf
; /etc/asterisk/voicemail.conf
; Author: Jeff Wiley
; Date: 11 July 2009
;
; Purpose: This file defines the necessary elements for
; Asterisk to act like a SIP server. This files
; creates two possible extensions that can be
; registered namely, 1000 and 1001. Both of these
; use the [sets] section and the variable choices
; placed therein.
;
;/;/
;
; Description of selections:
;
; type=friends
; means that the registered users can both send
; and receive calls
; context=voicemail_test_dialplan
; corresponds to the context from the extensions
; file
; host=dynamic
; means the user can register from any IP address
; disallow=all
; disables all of the codecs from being used on
; this channel
; allow=gsm
; as has been configured in the installation,
; this further enforces the use of the GSM codec
;
; The syntax used for the specific users:
; [1000](sets)
; Means to use the items defined in [sets] within
; the given user.
;
; Within the two users 1000 and 1001, the selections
; are as follows:
;
; secret=guessthis
; this is the cleartext password, there are ways
; to make this more secure, this is just for
; ease of use
; mailbox=1000
; this is the node that receives a message
; waiting indicator (MWI) when voice
; mail is received
;
;/;/

[general]

[sets](!)
type=friend
context=voicemail_test_dialplan
host=dynamic
disallow=all
allow=gsm
```



```

; they appear to have been stored using your new format list.
; If you don't do this, very unpleasant things may happen to
; your users while they are retrieving and manipulating
; their voice mail.
;
; In other words: don't change the format list on a production
; system unless you are _VERY_ sure that you know what you
; are doing and are prepared for the consequences.
;
;;;;;;End original test content

; How many milliseconds to skip forward/back when rew/ff
; in message playback
skipms=3000

; How many seconds of silence before we end the recording
maxsilence=10

; Silence threshold (what we consider silence: the lower,
; the more sensitive)
silencethreshold=128

; Max number of failed login attempts
maxlogins=3

;;;This message is part of the original test voicemail.conf file
;
; Each mailbox is listed in the form <mailbox>=
; <password>,<name>,<email>,<pager_email>,<options>
; if the e-mail is specified, a message will be sent when a
; message is
; received, to the given mailbox. If pager is specified, a
; message will be sent there as well. If the password is
; prefixed by '-', then it is considered to be unchangeable.
;
;;;;;;End original test content

[default]
1000 => 4321,Alice Test
1001 => 4321,Bob Test

```

APPENDIX C: TEST PROCEDURES

A. VOICE CALL TESTING PROCEDURES

Registration

- From the Asterisk command line interface, unregister both extensions. (command: sip unregister 1000 and sip unregister 1001)
- Be sure both extensions are no longer registered (command: sip show peers). This will show the extensions but both should say “(Unspecified)” under the Host heading.
- Reregister Alice and Bob. In Ekiga, select the Edit menu and Accounts. Uncheck and recheck the checkbox next to the account. A message will also appear on the Asterisk CLI indicating that a SIP registration was completed verifying the extension.
- Once both clients have been reregistered, on the Asterisk CLI, see that both extensions are again registered (command: sip show peers). To verify that the registration process was successful, the “Host” heading will have an IP address for both extensions.

User to user call with no answer

- Make a call from Alice to Bob. In the location bar, ensure Bob’s extension is entered correctly. The location bar should have the following: sip:1001@192.168.123.1. To place the call, press the green phone icon next to the location bar.
- Do not answer the call. Verify that Bob’s phone indicates that an incoming call is available. Although the phone cannot ring indefinitely, allow the phone to ring in order to verify that SIP is working to initiate a connection.

- Repeat with a call from Bob to Alice. In the location bar, ensure Alice's extension is entered correctly. The location bar should have the following: sip:1000@192.168.123.1.
- The test is complete when a call has gone from each user and has been indicated on the receiving end.

User to user call

- Make a call from Alice to Bob. In the location bar, ensure Bob's extension is entered correctly. The location bar should have the following: sip:1001@192.168.123.1.
- Answer the call to verify that RTP is sending the call data. Talk into the microphone and listen to the other client's output.
- Repeat with a call from Bob to Alice. In the location bar, ensure Alice's extension is entered correctly. The location bar should have the following: sip:1000@192.168.123.1.
- The test is complete when a call has gone from each user and has been answered on the receiving end.

Dial undefined extension

- Place a call using an invalid extension. The location bar should have something similar the following: sip:411@192.168.123.1. To place the call, press the green phone next to the location bar.
- These calls will not complete. On the Asterisk CLI, a message will indicate the following. Note: Due to ambiguity, if any number leading up to an existing extension is dialed, the error will not appear. This is because if 1, 10, or 100 is dialed, Asterisk is unable to know if the user is going to complete the rest of the extension. Using number such as these only gives the first line of the following message. Otherwise Asterisk will time out without attempting to call an extension.

== Using SIP RTP CoS mark 5

Call from '1000' to extension '411' rejected because extension not found.

- This is not an exhaustive test. The test is successful when calls to unregistered extensions are not able to complete.

B. VOICE MAIL BUILD TESTING PROCEDURES

Registration

- From the Asterisk command line interface, unregister both extensions. (command: sip unregister 1000 and sip unregister 1001)
- Be sure both extensions are no longer registered (command: sip show peers) This shows the extensions but both should say “(Unspecified)” under the Host heading.
- Reregister Alice and Bob. In Ekiga, select the Edit menu and Accounts. Uncheck and recheck the checkbox next to the account. A message will also appear on the Asterisk CLI indicating that a SIP registration was completed verifying the extension.
- Once both clients have been reregistered, on the Asterisk CLI, see that both extensions are again registered (command: sip show peers). To verify that the registration process was successful, the “Host” heading will have an IP address for both extensions.

User to user call

- Make a call from Alice to Bob. In the location bar, ensure Bob’s extension is entered correctly. The location bar should have the following: sip:1001@192.168.123.1. To place the call, press the green phone icon next to the location bar.
- Answer the call to verify that RTP is sending the call data.

- Repeat with a call from Bob to Alice. In the location bar, ensure Alice's extension is entered correctly. The location bar should have the following: sip:1000@192.168.123.1.
- The test is complete when a call has gone from each user and has been answered on the receiving end.

Dial undefined extension

- Place a call using an invalid extension. The location bar should have something similar the following: sip:411@192.168.123.1.
- These calls will not complete. On the Asterisk CLI, a message will indicate the following:

== Using SIP RTP CoS mark 5

Call from '1000' to extension '411' rejected because extension not found.

- This is not an exhaustive test. The test is successful when calls to unregistered extensions are not able to complete.

Make a call to leave a voice mail message

- Make a call from Alice to Bob. In the location bar, ensure Bob's extension is entered correctly. The location bar should have the following: sip:1001@192.168.123.1.
- Do not answer the call. After the specified amount of time (10 seconds) the Voice mail system will answer.
- Leave a message.
- After the message has been left, a message waiting indicator is sent by Asterisk. Ekiga will also play a sound but to further verify that a message has been left, click on the Edit menu and select Accounts. Next to the account name there should be a display indicating that a message is waiting (e.g. 1/0).

- Repeat with a call from Bob to Alice. In the location bar, ensure Alice's extension is entered correctly. The location bar should have the following: sip:1000@192.168.123.1.
- The test is complete when a call for each user has been handled by the voice mail system and a message has been saved.

Retrieve voice mail

- Dial into the voice mail system. The location bar should have the following: sip:700@192.168.123.1.
- In order to send the numbers click the Dialpad tab in Ekiga. The extension and password, as well as menu selections, are entered in this manner.
- Follow the audio instructions once the voice mail system has answered to listen to the new message. Enter the following for extensions and passwords to access the corresponding mailbox.
 - Alice's settings are as follows
Extension: 1000
Password: 4321
 - Bob's settings are as follows
Extension: 1001
Password: 4321
- This test is successful when the message for each user has been retrieved.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] L. Tse, "Feasibility study of VoIP integration into the MYSEA environment," M.S. thesis, Naval Postgraduate School, Monterey, California, September 2005.
- [2] J. Rosenberg et al., "SIP: session initiation protocol," RFC 3261, Internet Engineering Task Force, June 2002. Available: <http://www.rfc-editor.org/rfc/rfc3261.txt> (accessed September 15, 2009).
- [3] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, "RTP: a transport protocol for real-time application," RFC 3550, Internet Engineering Task Force, July 2003. Available: <http://www.rfc-editor.org/rfc/rfc3550.txt> (accessed September 2, 2009).
- [4] Asterisk: The Open Source PBX & Telephony Platform, 2009 Available: <http://www.asterisk.org/> (accessed September 15, 2009).
- [5] Digium®, The Asterisk Company, 2009 Available: <http://www.digium.com> (accessed August 15, 2009).
- [6] J. Malone, "Open source PBX is 18% of North America market," No Jitter, Jan 28, 2009. Available: <http://www.nojitter.com/showArticle.jhtml?articleID=212903167> (accessed August 15, 2009).
- [7] "Asterisk features," Voip-Info.org: A reference guide to all things VOIP, 2008. [Online]. Available: <http://www.voip-info.org/wiki/view/Asterisk+Features> (accessed August 16, 2009).
- [8] "Features," Asterisk The Open Source PBX & Telephony Platform, 2009. [Online]. Available: <http://www.asterisk.org/features> (accessed August 15, 2009).
- [9] F. E. Gonçalves, *Configuration guide for Asterisk PBX*, 2nd ed., V. Office Networks Ltd., 2006.
- [10] M. Spencer, "Distributed universal number discovery (DUNDi)," Internet Draft, October 2004. Available: <http://www.dundi.com/dundi.txt> (accessed September 15, 2009).

- [11] “Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON) Release 4; Open Settlement Protocol (OSP) for inter-domain pricing, authorization and usage exchange,” The European Telecommunications Standards Institute (ETSI), Sophia Antipolis, France, Tech. Specification, ETSI TS 101 321 (v4.1.1), November 2003. Available: http://portal.etsi.org/docbox/EC_Files/EC_Files/ts_10202403v040101p.pdf (accessed September 15, 2009).
- [12] T. Nguyen, T. Levin and C. Irvine, “MYSEA testbed,” In *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop (IAW'05)*, 2005, pp. 438–439.
- [13] D. E. Bell, L. LaPadula, “Secure computer system: unified exposition and Multics interpretation” Tech. Rep. ED-TR-75-306, MITRE Corp., Hanscom AFB, MA 1975.
- [14] K. J. Biba, “Integrity considerations for secure computer systems,” Tech. Rep. ED-TR-76-372, MITRE Corp., 1977.
- [15] J. F. Horn, “IPSec-based dynamic security services for the MYSEA environment,” M.S. thesis, Naval Postgraduate School, Monterey, California, June 2005.
- [16] C. E. Irvine, D. J. Shifflett, P. C. Clark, T. E. Levin and G. W. Dinolt, “MYSEA security architecture,” Naval Postgraduate School, Monterey, California, Tech. Report, NPS-CS-02-006, May 2002.
- [17] C. E. Irvine et al., “Overview of a high assurance architecture for distributed multilevel security,” in *Proceedings of the 2004 IEEE Systems, Man and Cybernetics Information Assurance Workshop*, West Point, NY, June 2004.
- [18] Australian Associated Press, “Small business gets \$120,000 phone bill after hackers attack VoIP phone,” [news.com.au](http://www.news.com.au/technology/story/0,28348,24939188-5014239,00.html), January 20, 2009. Available: <http://www.news.com.au/technology/story/0,28348,24939188-5014239,00.html> (accessed August 3, 2009).
- [19] H. Dwivedi, *Hacking VOIP*, San Francisco: No Starch Press, 2009.
- [20] D. R. Kuhn, T. J. Walsh, and S. Fries, “Security considerations for voice over IP systems,” NIST SP 800-58, US National Institute of Standards and Technology, December 2003.
- [21] P. Oechslin, “Making a faster cryptanalytic time-memory trade-off,” in *Advances in Cryptology—CRYPTO 2003*, pp. 617–630, 2003.

- [22] S. Whalen, "An introduction to ARP spoofing," White Paper, version 1.82, April 2001. Available: http://packetstorm.linuxsecurity.org/papers/protocols/intro_to_arp_spoofing.pdf (accessed July 31, 2009).
- [23] S. Hanley, "DNS overview with a discussion of DNS spoofing," November 2000. [Online]. Last accessed 07/2009, Available: <http://www.piclist.com/images/org/sans/www/http/infosecFAQ/DNS/DNS.htm> (accessed July 31, 2009).
- [24] M. Spencer, "The Asterisk(R) open source PBX," README for Asterisk 1.6.1.0, 2008. Available: <http://svn.digium.com/svn/asterisk/tags/1.6.1.0/README> (accessed September 15, 2009).
- [25] Asterisk Development Team, "Security," Asterisk Reference Information Version 1.6.1.0, pp. 10–11, 2009. Available: <http://svn.digium.com/svn/asterisk/tags/1.6.1.0/doc/tex/security.tex> (accessed August 4, 2009).
- [26] "Asterisk security advisory," 2009. [Online]. Available: <http://www.asterisk.org/security> (accessed July 8, 2009).
- [27] J. Todd, "Seven steps to better SIP security with asterisk," Inside the Asterisk Blog, March 2009. [Online]. Available: <http://blogs.digium.com/2009/03/28/sip-security/> (accessed September 15, 2009).
- [28] W. Mundy, "Avoiding the \$100,000 phone bill: a primer on Asterisk security," Nerd Vittles, 2009. [Online]. Available: <http://nerdvittles.com/?p=580> (accessed September 15, 2009).
- [29] "SIP TCP and TLS support," Asterisk developer's documentation, 2009. [Online]. Available: http://www.asterisk.org/doxygen/trunk/sip_tcp_tls.html (accessed July 29, 2009).
- [30] R. Bryant, "Testing of SIP TCP/TLS support," asterisk-dev mailing list, July 2007. [Online]. Available: <http://lists.digium.com/pipermail/asterisk-dev/2007-July/028454.html> (accessed July 29, 2009).
- [31] "0004903: [patch] SIP over TCP project," Asterisk.org Issue Tracker, 2005. [Online]. Available: <https://issues.asterisk.org/view.php?id=4903> (accessed September 15, 2009.)
- [32] "0005413: [branch] Secure RTP (SRTP)," Asterisk.org Issue Tracker, 2005. [Online]. Available: <https://issues.asterisk.org/view.php?id=5413> (accessed August 7, 2009).

- [33] J. Van Meggelen, J. Smith and L. Madsen, *Asterisk: the future of telephony*, 2nd ed. Sebastopol, California: O'Reilly, 2007.
- [34] Asterisk® Architecture Image (n.d.), Digium, Inc. [Online]. Available: <http://www.digium.com/images/graphics/asteriskarch.gif> (accessed August 27, 2009).
- [35] "Software PBX performance on Intel multi-core platforms - a study of Asterisk*," White Paper, 318862-001US, Intel Corporation, January 2008. Available: <http://download.intel.com/design/intarch/papers/318862.pdf> (accessed August 26, 2009).
- [36] M. Spencer, 2009, Asterisk's Developer Documentation. [Online]. Available: <http://www.asterisk.org/doxygen/trunk/> (accessed September 21, 2009).
- [37] Ekiga, "Downloads," 2009, [Online]. Available: <http://www.ekiga.org> (accessed September 16, 2009).

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Dr. Cynthia E. Irvine
Naval Postgraduate School
Monterey, California
4. Dr. Mark Gondree
Naval Postgraduate School
Monterey, California
5. Thuy D. Nguyen
Naval Postgraduate School
Monterey, California
6. Jeffrey A. Wiley, Jr.
SFS students: Civilian, Naval Postgraduate School
Monterey, California