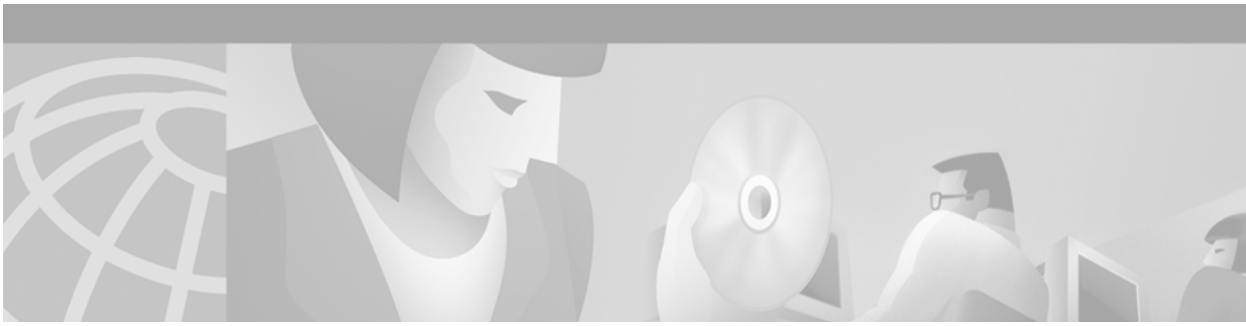


Prepaid Services

Version Number	Date	Notes
1	10/1/2001	This document was created.
2	4/12/2002	Note added to Step 5 of "OSP Clearinghouse Operation and Call Flow."

Prepaid services enable Internet telephony service providers (ITSPs) to offer calling card services that customers can pay for in advance. Prepaid services can be managed in two ways:

- Through an internally managed network infrastructure
- Through an Open Settlement Protocol (OSP) clearinghouse

In an ITSP internal network infrastructure, prepaid services are implemented through a debit card application that works in conjunction with the following:

- Interactive Voice Response (IVR)
- Authentication, authorization, and accounting (AAA) security services
- RADIUS security system
- An integrated third-party billing system

This combination of services enable you (as a carrier) to authorize voice calls and debit individual user accounts in real time at the edges of a Voice over IP (VoIP) network without requiring external service nodes. If you rely on an OSP clearinghouse to manage prepaid services, you configure your voice gateway to register with an OSP server. (OSP is designed to offer billing and accounting record consolidation for voice calls that traverse ITSP boundaries.) Third-party clearinghouses with an OSP server can offer services such as route selection, call authorization, call accounting, and intercarrier settlements, including all the complex rating and routing tables necessary for efficient and cost-effective interconnections.

In this document, we discuss how to design and implement a prepaid services solution that is managed either through your internal network infrastructure or through an OSP clearinghouse. This document contains the following sections:

- Prepaid Services Overview
- Internally Managed Prepaid Services Theory
- Internally Managed Prepaid Services Application
- OSP for Clearinghouse Services Theory
- OSP for Clearinghouse Services Application

Prepaid Services Overview

Prepaid services can be managed either internally within the infrastructure of your own network, or through the services of an OSP clearinghouse. Depending on the needs of your particular network, you can use one or both of these solutions to provide and manage prepaid services.

The key to providing internally managed prepaid services is the Cisco Systems debit card application, which coordinates the functionality of four separate applications: IVR, AAA, RADIUS, and a third-party billing system. IVR provides the customer interface; AAA and RADIUS form the infrastructure to provide authentication and billing; and IVR, AAA, and RADIUS communicate with the third-party billing system through vendor-specific attributes (VSAs).

OSP is used for inter carrier interconnect authorization and accounting, enabling carriers to admit and bill for each VoIP call accepted from another service provider. This capability is critical to toll-bypass applications, specifically international wholesale voice, because the terminating carrier must deliver the call to the Public Switched Telephone Network (PSTN), incurring a fee that must be funded out of the settlement payment from the originating carrier. Because of OSP, you can employ reliable third-party clearinghouses to handle VoIP call termination while leveraging the bandwidth efficiencies and tariff arbitrage advantages that are inherent in IP.

Internally Managed Prepaid Services Theory

The following sections describe the theory of internally managed prepaid services:

- Debit Card Application Overview
- Debit Card Application Functional Call Flow
- Internally Managed Prepaid Services Architecture

Debit Card Application Overview

The key to managing prepaid services internally within your own network is the Cisco debit card application. The debit card application integrates the functionality of IVR, AAA, RADIUS, and a third-party billing system. The IVR software infrastructure uses Tool Command Language (TCL) IVR scripts, dynamically combining prerecorded audio files to play the time, date, and dollar amount of remaining credit. AAA security services, in combination with a RADIUS server, provide the infrastructure for both authentication and accounting. The integrated third-party billing system maintains per-user credit balance information. RADIUS VSAs are used with AAA to communicate with the third-party billing system.

With Cisco IOS Release 12.1 and later releases, the debit card application offers the following functionality to support internally managed prepaid services:

- Rates a call according to the caller ID, personal identification number (PIN), and destination number.
- Plays the credit (dollar amount) remaining on a card in $$$$$$.$$ format.
- Announces the time remaining credit on the card in hours and minutes (hh:mm).
- Plays a “time-running-out” message based on the configurable timeout value.
- Plays a warning “time-has-run-out” message when the credit expires.

- Makes more than one successive call to different destinations during a single call session by using the “long pound key” feature. This feature also allows the caller to make additional calls if the called party hangs up.
- Reauthorizes each new call.
- Allows type-ahead keypad entries to be made before the prompt has been completed.
- Allows the caller to bypass announcements by pressing a touch-tone key.
- Allows retry when data (caller ID,PIN, destination number) is entered by using a special key.
- Terminates a field by size rather than by using a terminating character (#).
- Supports multiple languages.
- Sends an off-net tone to the caller.
- Provides voice-quality information to the RADIUS server on a call-by-call basis.
- Creates dynamic prompts by using prerecorded audio files.
- Supports local announcements with customized audio files.
- Determines how many languages are configured and only plays the language selection menu if needed.
- Supports extended TCL IVR scripting.

IVR and TCL IVR Scripts

The debit card application uses IVR as the mechanism at the voice gateway to present a customized interface to prepaid services customers. The IVR application provides simple voice prompting and digit collection to gather caller information for authenticating users and identifying destination telephone numbers.

IVR uses TCL scripts and audio files to provide voice prompting and digit collection. TCL scripts contain both executable files and audio files that interact with the system software. When a TCL IVR script is activated by an incoming call, the C code is activated in run-time mode and performs the work of these executables. Examples of the executables used by TCL IVR scripts include:

- Play—Plays an audio file prompt for the caller
- Collect—Collects dual tone multifrequency (DTMF) digits, such as a PIN
- Send—Sends information collected to the RADIUS server and expects some results

As of Cisco IOS Release 12.1 and later releases, TCL IVR scripts and audio files are stored on an external TFTP server for dynamic access by the voice gateways. The scripts are loaded into RAM and remain resident as long as the gateway remains active.

New TCL IVR scripts are being developed on a continuous basis. Table 1 lists the TCL IVR scripts that are included in Cisco IOS Release 12.2. Table 2 lists some of the TCL IVR scripts that can be loaded externally.

Table 1 TCL IVR Scripts Included in Cisco IOS Software

TCL IVR Script Name	Description
clid_col_dnis_3.tcl	Authenticates the caller ID three times, first with DNIS; if unsuccessful, attempts to authenticate with the caller PIN up to three times.
clid_col_npw_3.tcl	Authenticates with NULL; if unsuccessful, attempts to authenticate using the caller PIN up to three times.
clid_4digits_npw_3.tcl	Authenticates with NULL; if unsuccessful, attempts to authenticate with the caller PIN up to three times using the 14-digit account number and password entered together.
clid_4digits_npw_3_cli.tcl	Authenticates the account number and PIN, respectively, using automatic number identification (ANI) and NULL. The length of digits allowed for the account number and password are configurable through the command line interface (CLI). If the authentication fails, allows the caller to retry. The retry number is also configured through the CLI.
clid_authen_col_npw_cli.tcl	Authenticates the account number and PIN, respectively, using ANI and NULL. If the authentication fails, allows the caller to retry. The retry number is configured through the CLI. The account number and PIN are collected separately.
clid_authen_collect_cli.tcl	Authenticates the account number and PIN using ANI and DNIS. If the authentication fails, allows the caller to retry. The retry number is configured through the CLI. The account number and PIN are collected separately.
clid_col_npw_3_cli.tcl	Authenticates using ANI and NULL for account and PIN respectively. If the authentication fails, allows the caller to retry. The retry number is configured through the CLI.
clid_col_npw_npw_cli.tcl	Authenticates using ANI and NULL for account and PIN, respectively. If the authentication fails, allows the caller to retry. The retry number is configured through the CLI. The account number and PIN are collected together.

Table 2 TCL IVR Scripts Available Externally

TCL IVR Script Name	Description
clid_authen_collect_cli_rtsp.2.0.0.tcl	Authenticates using ANI or DNIS for account number and PIN. If that fails, it authenticates using account number and PIN. If authentication passes, it collects the destination number and places the call.
debitcard.tcl	Collects account ID and PIN with a single prompt.
debitcard_acct_pin.tcl	Collects the account ID and PIN with two separate prompts.

Table 2 TCL IVR Scripts Available Externally (continued)

TCL IVR Script Name	Description
debitcard_mgcp.2.0.0.tcl	Collects account ID and PIN when using Media Gateway Control Protocol (MGCP).
fax_rollover_on_busy.2.0.0.tcl	Mimics the default SESSSION script. It collects account ID and PIN, and places the call. If the call is not successfully connected it up, the call will retry the next VoIP dial peer. If the fax relay is busy, the call will be routed to the fax store and forward.
libretto_offramp5.2.0.0.tcl	Handles off ramp calls (coming from the VoIP dial peer), performs authentication if it is configured through the CLI, and the hands the call off to the Libretto off ramp application.
libretto_onramp9.2.0.0.tcl	Handles on ramp calls (coming from the POTS dial peer) and performs authentication if it is configured through the CLI. If authentication fails, the call is handed off to the Libretto application with the failure status. If authentication is successful, the user is prompted to enter the destination number. (Unless direct-inward-dial (DID) or redialer are configured, in which case the destination number has already been collected.)
remote_ip_authenticate.2.0.0.tcl	Replaces the SESSION script for a VoIP call leg. If the remote (calling side) IP address is authenticated, the script places the call. Authentication is performed on the IP address and the fixed password “cisco.” If authentication fails, the script returns with an error code, and the call is rejected.

Cisco provides a set of professionally recorded audio prompts (IVR audio files) in a number of different languages. New audio prompts are being developed on a continuous basis. Table 3 and Table 4 list some of the IVR audio files available as of Cisco IOS Release 12.1.

Table 3 Number Audio File Set

Audio Filename	Recorded Prompt	Audio Filename	Recorded Prompt
en_zero.au	Zero	en_fifteen.au	Fifteen
en_one.au	One	en_sixteen.au	Sixteen
en_two.au	Two	en_seventeen.au	Seventeen
en_three.au	Three	en_eighteen.au	Eighteen
en_four.au	Four	en_nineteen.au	Nineteen
en_five.au	Five	en_twenty.au	Twenty
en_six.au	Six	en_thirty.au	Thirty
en_seven.au	Seven	en_forty.au	Forty
en_eight.au	Eight	en_fifty.au	Fifty
en_nine.au	Nine	en_sixy.au	Sixty

Table 3 Number Audio File Set (continued)

Audio Filename	Recorded Prompt	Audio Filename	Recorded Prompt
en_ten.au	Ten	en_seventy.au	Seventy
en_eleven.au	Eleven	en_eighty.au	Eighty
en_twelve.au	Twelve	en_ninety.au	Ninety
en_thirteen.au	Thirteen	en_hundred.au	Hundred
en_fourteen.au	Fourteen	en_thousand.au	Thousand

Table 4 Additional Miscellaneous Prompts

Audio Filename	Recorded Prompt
en_second.au	Second
en_seconds.au	Seconds
en_minute.au	Minute
en_minutes.au	Minutes
en_hour.au	Hour
en_hours.au	Hours
en_cent.au	Cent
en_Cents.au	Cents
en_dollar.au	Dollar
en_dollars.au	Dollars
en_welcome.au	“Welcome to Cisco Debit Card Demo.”
en_lang_select.au	“Please press 1 for English, 2 for Mandarin.”
en_wrong_lang_sel.au	“You have made an invalid selection. Please press 1 for English or press 2 for Mandarin.”
en_no_lang_sel.au	“You did not select any language. Press 1 for English or press 2 for Mandarin.”
en_final.au	“We are having difficulties connecting your call. Please try again later.”
en_generic_final.au	“Please hang up and try again.”
en_enter_card_num.au	“Please enter card number followed by pound.”
en_invalid_digits.au	“You have entered an invalid number of digits. Please reenter your card number followed by pound.”
en_auth_fail.au	“You have entered an invalid card number. Please reenter your card number followed by pound.”
en_no_card_entered.au	“You did not enter any digits. Please enter card number followed by pound.”
en_technical_problem.au	“We are having technical difficulties. Please call back later.”
en_zero_bal.au	“You have zero balance. Please call the operator or hang up.”
en_enter_dest.au	“Please enter destination number.”
en_disconnect.au	“Your call will be disconnected.”

Table 4 Additional Miscellaneous Prompts (continued)

Audio Filename	Recorded Prompt
en_disconnected.au	"You have been disconnected."
en_dest_collect_fail.au	"Sorry, the number you have dialed is blocked. If you feel you have reached a number in error, please call the customer service number."
en_invalid_amt.au	"You have more than one million."
en_dest_busy.au	"The party you called is busy, please enter a new number or hang up and try again later."
en_enter_acct.au	"Please enter your account number followed by the pound key."
en_no_acct_entered.au	"We did not get any input, please enter your account number followed by the pound key."
en_invalid_digits_acct.au	"You have entered an invalid number of digits. Please enter your account number followed by the pound key."
en_invalid_account.au	"You have entered an invalid account number. Please enter your account number followed by the pound key."
en_no_dialpeer_match.au	"You have entered an invalid destination. Please reenter the destination number you are calling."
en_connect_cust_ser.au	"You will be connected to Customer Service."
en_dial_cust_ser.au	"Please hang up and dial the calling card customer service number."
en_no_service.au	"We're sorry, this service is not available."
en_dest_unreachable.au	"We're sorry, the destination you have called is unreachable."
en_toll_free.au	"You can only make toll-free calls."

You can find TCLWare (TCL scripts) and audio files at the following URL:

http://www.cisco.com/cgi-bin/tablebuild.pl/tclware

AAA and RADIUS

The debit card application uses AAA security services as the infrastructure with which to provide authentication and accounting services. AAA is an architectural framework for configuring a set of three independent security functions: authentication, authorization, and accounting. Authentication is the way a user is identified prior to being allowed access to services. Authorization provides a method for remote access control. Accounting provides a method for collecting and sending security server information used for billing, auditing, and reporting data such as user identities and start and stop times.

Typically, AAA works in tandem with a RADIUS server. (TACACS does not support prepaid services.) RADIUS uses Internet Engineering Task Force (IETF) standard, vendor-specific, and vendor-proprietary attributes to define specific AAA elements in a user profile that are stored on the RADIUS database. RADIUS and AAA authenticate, authorize, and perform accounting functions by associating attribute/value (AV) pairs with the appropriate user. For internally managed prepaid services, AAA works in tandem with IVR to enable voice gateways to interact with a RADIUS security server to authenticate users (typically incoming calls) and to perform accounting services.

Authentication

The gateway normally uses AAA in conjunction with IVR to check the legitimacy of a prospective gateway user based on an account number collected by IVR, or based on ANI. When the gateway uses AAA with IVR, the IVR application collects the user account and PIN information and then passes it to the AAA interface. The AAA interface makes a RADIUS authentication request with the given information, and, based on the information received from the RADIUS server, forwards either a pass or fail message to the IVR application.

Accounting

The RADIUS server collects basic start-stop connection accounting data during the accounting process for each call leg created on the gateway. The RADIUS server can be configured to collect accounting data using one of two methods:

- Start-stop. The RADIUS server collects a call-start record and a call-stop record for each call leg, producing a total of eight records for each normally completed call.
- Stop-only. The RADIUS server collects a call-stop record for each call leg, producing a total of four call records for each normally completed call.

Note There are many circumstances in which you will receive more than eight start-stop records or four stop-only records.

The various call leg start and stop records generated by the gateway can be organized by their *Connection ID*, which is the same for all call legs of a connection. The Connection ID is a 128-bit field displayed in hexadecimal format that can vary in appearance. In the examples cited in this document, the Connection ID is of the form 3C5AEAB9 95C80008 0 587F34 (one 4-octet string, a space, one 4-octet string, a space, a zero, a space, and one 3-octet string). The billing application uses the Connection ID to generate all the information needed for accurate and timely billing.

Start records by definition cannot contain the time connection details required for billing by time; these details are contained in the stop records. (All RADIUS billing information pertaining to the call is contained in the stop records.) However, some deployments choose to use start/stop records so that they will know when a call was terminated abnormally and thus has no stop record. Start records, in conjunction with update records, provide a more accurate and deterministic real-time measurement technique for identifying when a call started than do stop-only records. Start-stop records are also useful when packets get lost. Stop-only accounting records are configured if RADIUS network traffic or storage needs are an issue. Update records can be obtained from Cisco routers by using the **aaa accounting update** global configuration command.

Call Detail Records

For debit-card networks, the voice gateways can send accounting data in the form of call detail records (CDRs) to the RADIUS server in one of two ways:

- Using the overloaded Acct-Session-ID RADIUS attribute
- Using vendor-specific RADIUS attributes

If the gateway sends a CDR and does not receive a response from the RADIUS server within a certain period of time, it will produce duplicate CDRs and deliver them to the RADIUS server. For example, the gateway will send duplicate CDRs if it does not receive a timely response from the RADIUS server acknowledging receipt of the original record. The only difference in these duplicate CDRs is in the attribute/value (AV) pair Acct-Delay-Time (attribute 41). The first value for Acct-Delay-Time is 0;

when duplicate records are created, the Acct-Delay-Time value is incremented in each subsequent record. All other fields in the duplicate CDRs remain the same. The particular billing application is responsible for discarding these duplicate records.

VSAs

VSAs are defined as Attribute 26 of the IETF standard group of AV pairs. For the RADIUS server to receive accounting information from the gateway using Attribute 26, the gateway must be configured to recognize RADIUS VSAs. After you configure the gateway to recognize VSAs, the RADIUS server will no longer overload the Acct-Session-ID attribute. Instead, the information elements in the overloaded Acct-Session-ID attribute will be captured in separate VSAs.

The following example shows the value string for a typical VSA:

Attribute 26 23 0000000967146833

The string elements are described as follows:

- Attribute 26 indicates a VSA.
- The value 23 represents the length in bytes.
- The value 0000000967146833 is broken into three parts:
  - For the first four octets, 00000009, the high-order octet is 0. The three low-order octets are the assigned vendor network management private enterprise code as defined in RFC 1700 (the Cisco Systems assigned vendor code is 9).
  - The next octet, 67, represents the VSA attribute number in hex (hex 67 = attribute 103 or return code).
- The last three octets, 146833, are vendor configurable.

Table 5 lists the VSAs used by Cisco voice products and describes their formats and purposes.

Table 6 lists the RADIUS codes that identify the kind of packet RADIUS is sending. Table 7 lists RADIUS return codes, which you can define by using TCL IVR scripts as long as the RADIUS server is configured to understand which response or return message is required.

Table 5 VSA Formats and Descriptions

AVpair	VSA No.	Format for value/text	Sample value/text	Purpose
Cisco-NAS-port	2	String of characters and numbers	BRI0/0:1	Incoming port identification on NAS or gateway. The Cisco-NAS-port syntax is: signalling type controller: timeslot group/control channel: bearer channel The Cisco-NAS-port VSA has the same function as RADIUS attribute 5, but it uses strings assigned by Cisco IOS to its hardware ports. See the "Cisco-NAS-Port VSA Format" section.
h323-billing-model = value	109	0 = credit/postpaid 1= debit/prepaid	1	Type of billing service for a specific call.

Table 5 VSA Formats and Descriptions (continued)

AVpair	VSA No.	Format for value/text	Sample value/text	Purpose
h323-call-origin=value	26	answer=legs 1 and 3 originate = legs 2 and 4 callback = legs 1 and 3	answer	The gateway's behavior in relation to the connection that is active for this leg. For example, answer on a leg 1; originate on a leg 2; callback on leg 1.
h323-call-type=value	27	Telephony; VOIP; VOFR	VOIP	Protocol type or family used on this leg of the call.
h323-conf-id=value	24	16-byte number in hexadecimal notation with one space between each 4-byte integer	3C5AEAB9 95C80008 0 587F34	Unique call identifier generated by the gateway. Used to identify the separate billable events (calls) within a single calling session. In Cisco IOS call control application programming interface (CCAPI), this value is called the globally unique identifier (GUID). The h323-conf-id is different from the h323-incoming-conf-id. For example, in long pound calls (calls in which you press the # key to make a new call) with a prepaid application, a new h323-conf-id value is generated for each new call. The new value is generated in the leg following authorization (either leg 2 or leg 4) and is subsequently passed to each downstream leg. Gateway-retries due to a connection request failure do not result in a new value; each retry uses the same h323-conf-id value. See the “Generating h323-incoming-conf-id and h323-conf-id” section.
h323-connect-time=value	28	hh:mm:ss:mmm ZON DDD MMM ## YYYY	18:27:30:094 PST Fri Aug 25 2000	Connect time in Network Time Protocol (NTP) format: hour, minutes, seconds, microseconds, timezone, day, month, day_of_month, and year.
h323-credit-amount=value	101	Decimal digits in the format: n.nn or n	1000.00 = one thousand; 1000 = 1000 cents or 10.00	Amount of credit (in currency) that the account contains.
h323-credit-time=value	102	Integer in decimal notation	300	Number of seconds for which the call is authorized.

Table 5 VSA Formats and Descriptions (continued)

AVpair	VSA No.	Format for value/text	Sample value/text	Purpose
h323-currency = value	110	3-character value from ISO 4217	USD	Currency for use with h323-credit-amount.
h323-disconnect-cause=value	30	2-character, ASCII-encoded hexadecimal number representing a Q.931 code. Range is 01 to A0 (which is 1-160 decimal)	4	Q.931 disconnect cause code retrieved from CCAPI. The source of the code is the disconnect location such as a Public Switched Telephone Network (PSTN), Terminating Gateway (TGW), or Session Initiation Protocol (SIP).
h323-disconnect-time=value	29	hh:mm:ss:mmm ZON DDD MMM ## YYYY	18:27:30.094 PST Fri Aug 25 2000	Disconnect time in NTP format: hour, minutes, seconds, microseconds, timezone, day, month, day_of_month, year.
h323-gw-id=value	33	Character string	bowie.cisco.com, AS5300_5	Domain name server (DNS) name or local name of the voice gateway that is sending the VSA.
h323-incoming-conf-id=value	1	16-byte number in hexadecimal notation with one space between each 4-byte integer	33C5AEAB9 95C80008 AF27092C 587F34 and 3C5AEAB9 95C80008 0 587F34	A unique number for identifying a calling session on a gateway, where a session is closed when the calling party hangs up. The h323-incoming-conf-id number is used to: • Match the outbound and inbound call legs for a session on a particular gateway • Collect and match all records for multiple calls placed (within the bounds of a session) on the gateway. The value used for legs 1 and 2 on the originating gateway (OGW) can be different than the value used for legs 3 and 4 on a TGW. The h323-incoming-conf-id is different from h323-conf-id. For example, the h323-incoming-conf-id value remains the same in the start/stop records for long pound calls. See the section, “Generating h323-incoming-conf-id and h323-conf-id” section.

Table 5 VSA Formats and Descriptions (continued)

AVpair	VSA No.	Format for value/text	Sample value/text	Purpose
h323-ivr-in=<value_1>:<value_2>	1	Customer defined	color:red	User-definable AV pairs sent from the RADIUS server to the voice gateway. You can read and use the value at the gateway via a customized TCL IVR script.
h323-ivr-out=<value_1>:<value_2>	1	Customer defined	color:blue	User-definable AV pairs sent from the voice gateway to the RADIUS server. You can set (write) the value via a customized TCL IVR script.
h323-preferred-lang=value	107	2-character code from ISO 639-1	en	Language to use when playing the audio prompt specified by h323-prompt-id.
h323-prompt-id=value	104	Integer in decimal notation	27	Index into an array that selects prompt files used at the gateway.
h323-redirect-ip-address=value	108	Numerals in dotted decimal notation	192.1.175.16	IP address for an alternate or redirected call.
h323-redirect-number=value	106	E.164 format (decimal digits with no spacing characters)	14088531111	Phone number to which the call is redirected; for example, to an 800 number or a customer service number.
h323-remote-address=value	23	Numerals in dotted decimal notation: nnn.nnn.nnn.nnn	156.221.17.128	IP address of the remote gateway
h323-remote-id=value	1	String	joshi4.mydomain	DNS name or locally defined hostname of the remote gateway
h323-return-code=value	103	Decimal numbers	0	Return codes are instructions from the server to the voice gateway. Table 2 lists return code values.
h323-setup-time=value1	25	hh:mm:ss.mmm ZON DDD MMM ## YYYY	18:27:28.032 UTC Wed Dec 9 1998	Setup time in NTP format: hour, minutes, seconds, microseconds, timezone, day, month, day_of_month, year.
h323-time-and-day=value	105	Decimal number: hh:mm:ss	10:36:57	Time of day at the dialed number or at the remote gateway in the format: hour, minutes, seconds.
h323-voice-quality=value	31	Decimal numbers from ICPIF table of G.113	5	Value representing impairment/calculated planning impairment factor (ICPIF) of the voice quality on the connection provided by lower-layer drivers (such as the digital-signal-processor). Low numbers represent better quality.

Table 5 VSA Formats and Descriptions (continued)

AVpair	VSA No.	Format for value/text	Sample value/text	Purpose
subscriber=value	1	String from T1/CAS or E1/R2 line/signal.	Coin	T1/CAS or E1/R2 signal information about subscriber
in-portgrp-id=text	1	ASCII string associated with the port on the GW used by this call.	<Service Provider ID>	The text carries a description associated with the incoming hardware telephony port that is used with this leg of the call.
out-portgrp-id=text	1	ASCII string associated with the port on the GW used by this call.	<Service Provider ID>	The text carries a description associated with the outgoing hardware telephony port that is used with this leg of the call.x

Table 6 RADIUS Codes

Code	Meaning
1	Access-Request
2	Access-Accept
3	Access-Reject
4	Accounting-Request
5	Accounting-Response
11	Access-Challenge
12	Status-Server (experimental)
13	Status-Client (experimental)
255	Reserved

Table 7 RADIUS Return Codes

Code	Meaning
0	Success, proceed
1	Failed—Invalid Account number
2	Failed—Invalid Password
3	Failed—Account in use
4	Failed—Zero balance
5	Failed—Card expired
6	Failed—Credit limit
7	Failed—User denied
8	Failed—Service not available
9	Failed—Called number blocked
10	Failed—Number of retries exceeded
11	Failed—Invalid argument

Table 7 RADIUS Return Codes (continued)

Code	Meaning
12	Failed—Insufficient funds
13	Toll-free call
14	Failed—Invalid card number
50	Redirect—Call will be hairpinned back to PSTN network
51	Redirect to called party (use redirect number)
52	Redirect to customer Service (use redirect number)
53	Connect IP leg to redirect IP address (108)

Debit Card Application Functional Call Flow

The following list describes a high-level call flow sequence for a debit card application. The actual call flow varies, depending on the parameters passed to the application and also on the features that are available on the RADIUS billing system that is being used. Figure 1 through Figure 5 provide a detailed flow diagram of this process.

1. A customer calls the access number of the ITSP or other company offering the service. The application begins with a welcome message (Figure 1).
2. If you have configured the application for multiple languages, the customer is prompted to select a preferred language (Figure 1).
3. After the preferred language is selected, the customer is prompted for an account number (Figure 2). The account number is the combination of the UID and PIN. These entries must have the same number of digits as configured in the gateway call application parameters. Provided the account number is the proper length and is a valid account number, the customer is authorized.
4. After successful completion of this first authorization phase, the prompt returns the amount of credit available on the customer account (Figure 4).
5. The next prompt asks for a destination number. A second authorization phase then occurs, authorizing a call to the number entered (Figure 4).
6. If the customer is authorized, the prompt returns the amount of time left in the customer account for a call to that destination (Figure 5).
7. The call is completed when a caller hangs up. If instead the caller presses and holds the pound (#) button on the telephone keypad for more than 2 seconds, the authorization process begins again at the second authorization phase.
8. The prompt returns a new credit amount to the caller, and the call to the new destination begins. If the customer does not disconnect, repeated calls can be made without having to repeat first-phase authentication.
9. If, at any time during a call, the credit amount left in the customer's account reaches the preconfigured warning amount (typically, 1 minute of service left), a warning prompt is played (Figure 5).
10. If a caller continues to talk until all the time is consumed, a disconnect message is played (Figure 5).

Figure 1 Debit Card Application Call Flow (Part 1)

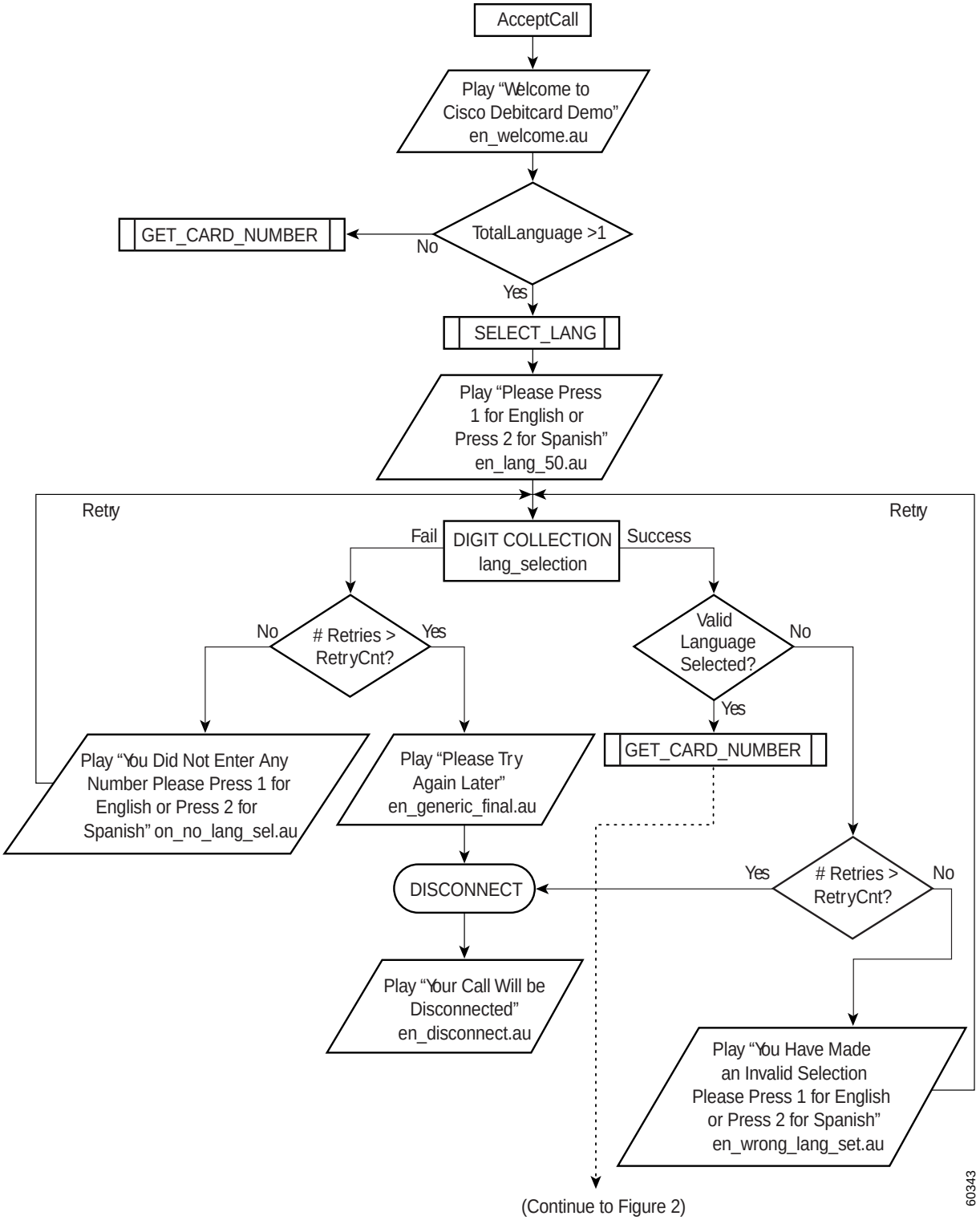


Figure 2 Debit Card Application Call Flow (Part 2)

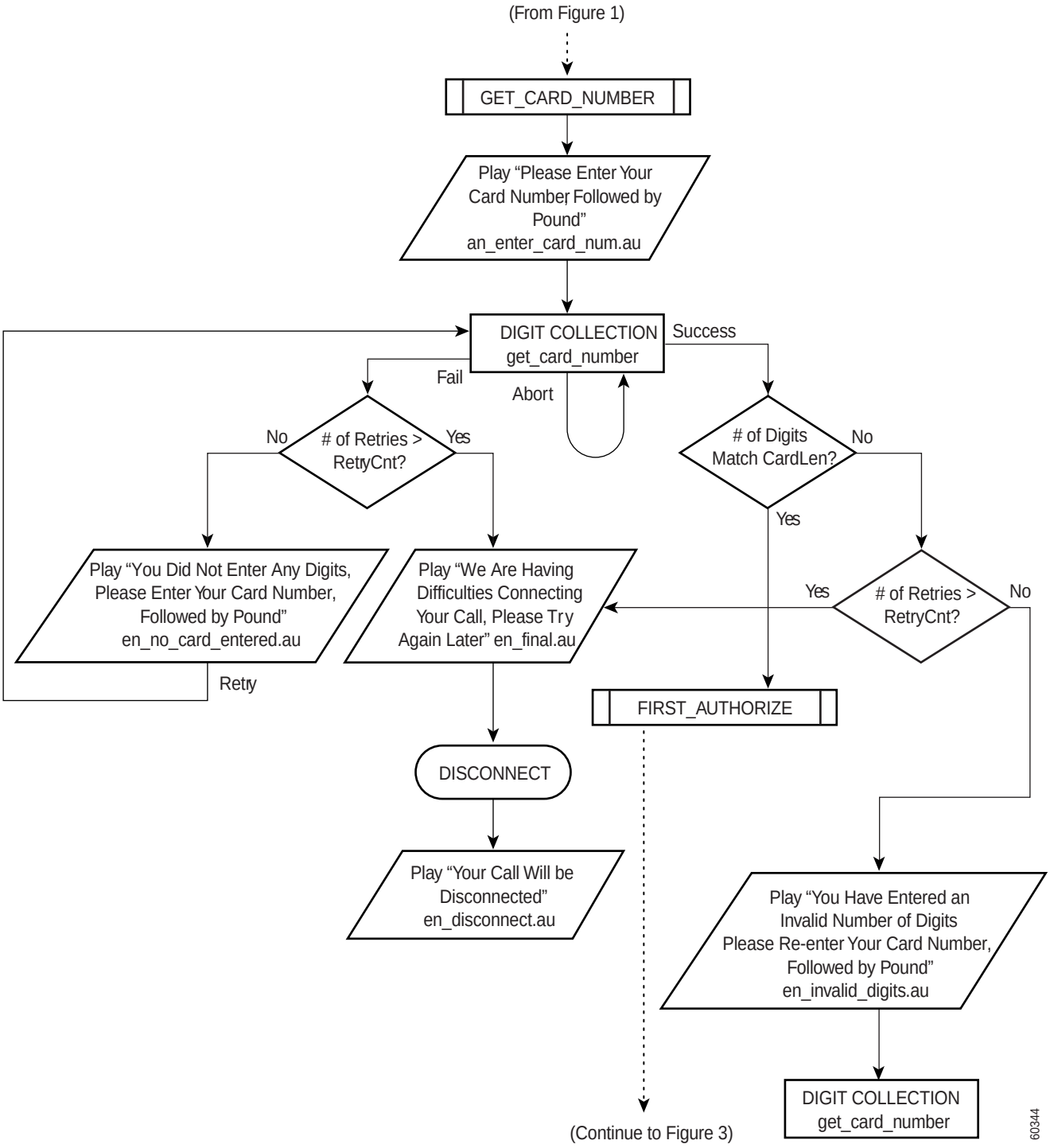


Figure 3 Debit Card Application Call Flow (Part 3)

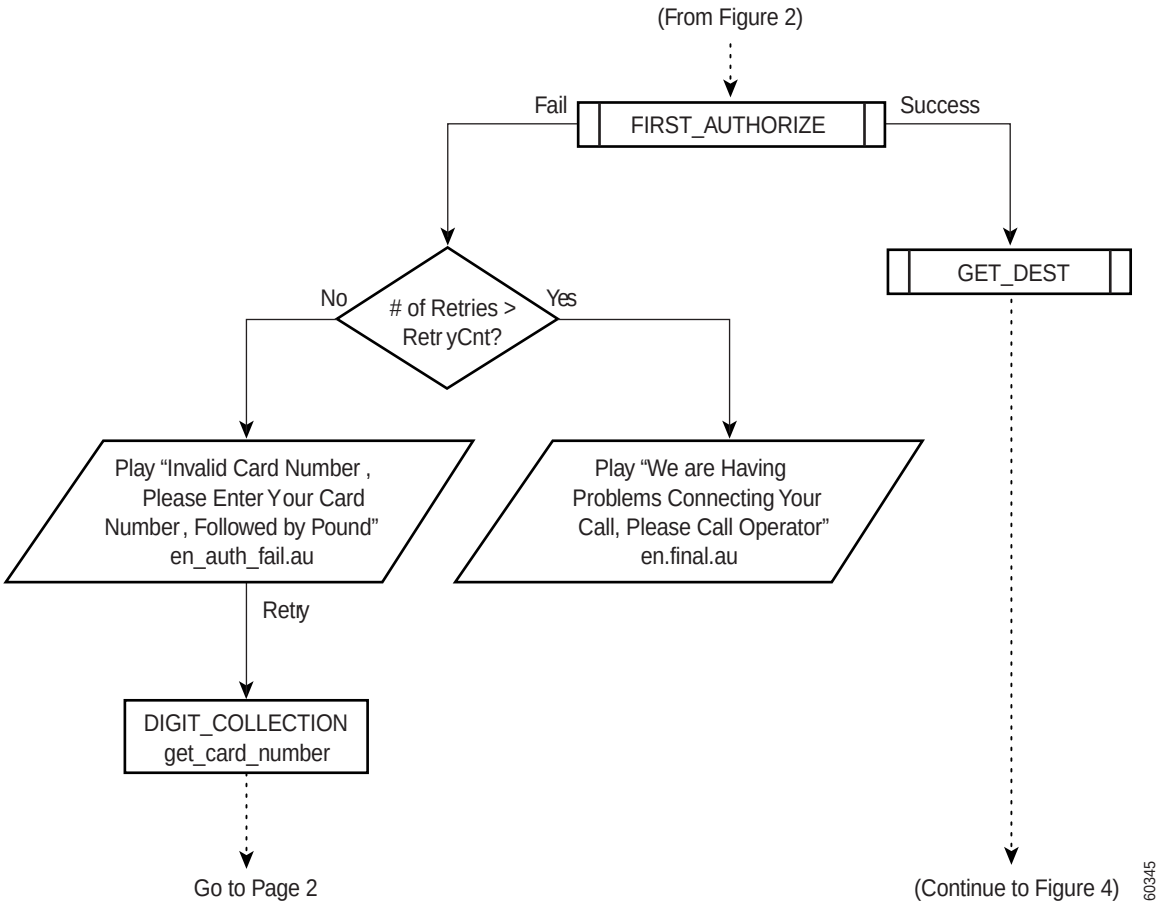


Figure 4 Debit Card Application Call Flow (Part 4)

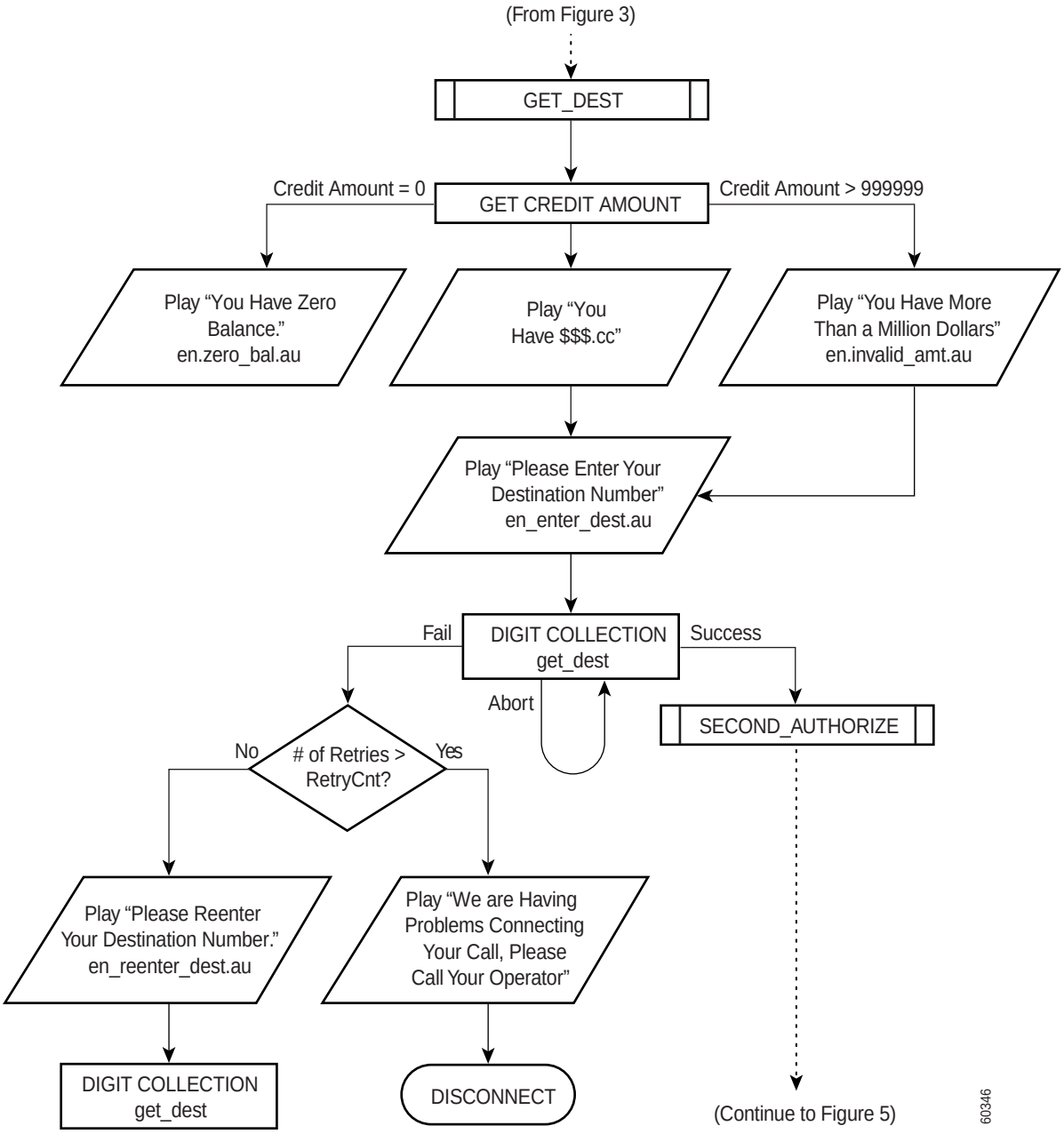
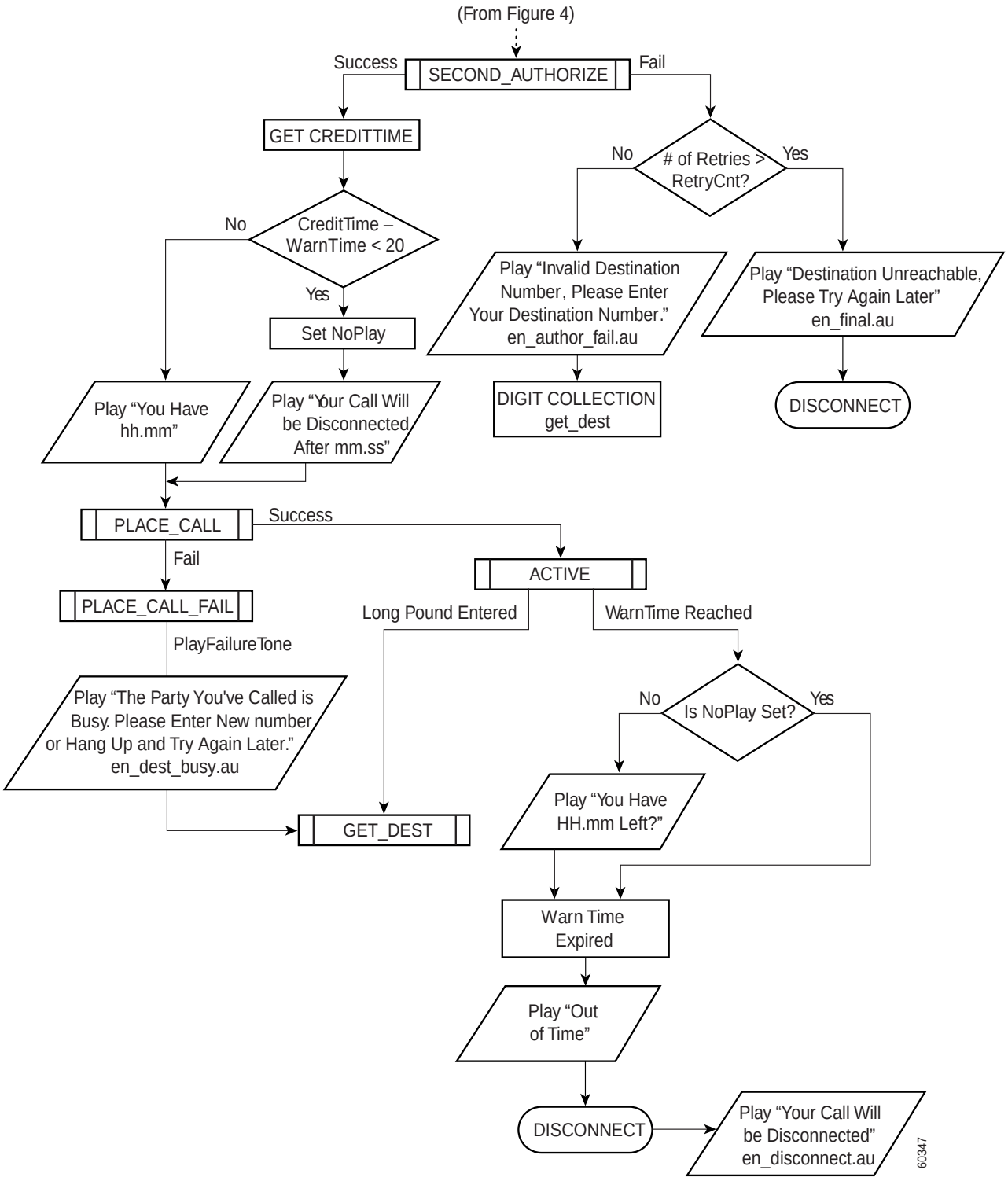


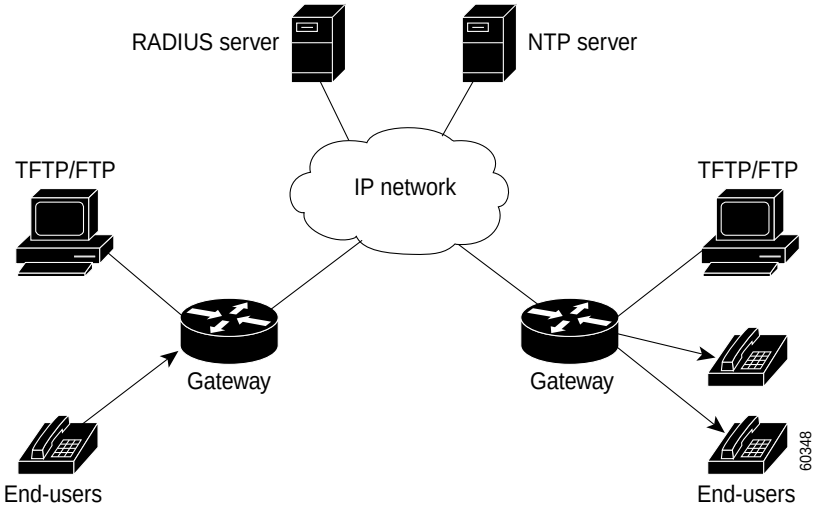
Figure 5 Debit Card Application Call Flow (Part 5)



Internally Managed Prepaid Services Architecture

The architecture for a network designed to manage prepaid services from within its own infrastructure can vary depending on the relative locations of the voice gateways. Figure 6 illustrates an architectural model in which the FTP/TFTP severs are local to the gateways and the RADIUS server is centralized. Both FTP/TFTP and RADIUS servers can be centralized or geographically dispersed, depending on the nature of the packet telephony network involved.

Figure 6 Internally Managed Prepaid Services Network Topology



The billing and accounting components of prepaid services consist of VoIP gateways, TCL IVR scripts, audio files, a TFTP server, and an integrated, third-party, RADIUS-based billing system.

Hardware and Software Requirements

The following sections describe the hardware and software requirements to support internally managed prepaid services:

- Network Devices
- System Platform Requirements
- Cisco IOS Software and VCWare Requirements
- Memory

Network Devices

Given the network topology shown in Figure 6, you need the following devices to support internally managed prepaid services:

- Gateways—The debit card application operates on Cisco VoIP H.323 gateways, including the Cisco AS5300 universal access servers and the Cisco 2600 series, 3620, and 3640 routers.
- RADIUS server—A standard RADIUS server performs the back-end billing process. This RADIUS server must be enabled to parse and understand VSAs and must be able to respond with the required VSAs, RADIUS codes, and RADIUS return codes. For smaller-scale deployments, the RADIUS and TFTP servers can be on the same device.

- TFTP/ FTP server—This server stores the audio files and TCL script files necessary to operate the debit card application. The TCL IVR scripts and audio files prompt the user for information such as account number or destination number, and return values such as time remaining or money remaining on the card. There are approximately 175 audio files for each language.
- NTP server—This server has a stratum-1 or stratum-2 clock. All of the devices in the network must synchronize their clocks to the NTP server to ensure accurate and consistent accounting records.

System Platform Requirements

The IVR and debit card applications are supported on the following Cisco platforms:

- Cisco AS5000 series universal access servers (5300, 5400, 5800, 5850)
- Cisco 3620, 3640, 3660 routers
- Cisco 2600 series routers

Cisco IOS Software and VCWare Requirements

To support prepaid services, the Cisco devices in your network must have the following versions of Cisco IOS software and VCWare installed:

- Cisco IOS Release 12.0(7)T or a later release
- VCWare Version 4.10

Note Various versions of VCWare and DSPWare are required to complement different versions of Cisco IOS software. The Cisco AS5350 access server does not require separate VCWare, because it is already built in to its Cisco IOS software. For complete information, please consult the compatibility matrix at the following URL: http://www.cisco.com/univercd/cc/td/doc/product/access/acs_serv/5300/iosrn/vcwrn/vcwrmtrx.htm

Memory

To support prepaid services, the Cisco devices in your network must have the following minimum memory installed:

- 32 MB Flash memory
- 128 MB DRAM memory

Internally Managed Prepaid Services Application

The following sections describe the application of prepaid services:

- Internally Managed Prepaid Services Configuration Guidelines
- Internally Managed Prepaid Services Call Example

Internally Managed Prepaid Services Configuration Guidelines

This section gives general configuration guidelines. To configure a typical voice gateway to support internally managed prepaid services, perform the following tasks:

- Loading TCL IVR Scripts and Storing Sound Files
- Configuring Call Application Parameters
- Configuring Dial Peers
- Configuring AAA
- Configuring RADIUS and VSA Parameters
- Configuring NTP

Loading TCL IVR Scripts and Storing Sound Files

You can use a Cisco.com password, to download the most recent TCL IVR scripts and audio files from the following URL:

http://www.cisco.com/cgi-bin/tablebuild.pl/tclware

After you have downloaded the scripts and audio files, you need to load them onto the TFTP server so that they will be available for the voice gateway. When you unzip the TCLWare bundle into the /tftpboot directory of the TFTP server, the various TCL and audio files will be positioned in the recommended subdirectory hierarchy. Table 8 gives the recommended directory structure.

Table 8 TCL Script and Audio File Directory Structure

Directory	Contents
/tftpboot/tcl	TCL IIVR scripts
/tftpboot/au/en	English language prompts and audio files
/tftpboot/au/ch	Chinese language prompts and audio files
/tftpboot/au/sp	Spanish language prompts and audio files

After you have downloaded the TCL IVR scripts and audio files into the appropriate subdirectories on the TFTP server, you are ready to configure your voice gateway to support prepaid services.

The following example shows how the TCL script is loaded into DRAM on the voice gateway.

```
pmeas1(config)# call application voice debit tftp://mojo/tcl/debitcard.tcl

Loading tcl/debitcard.tcl from 10.19.21.101 (via Ethernet0): !!!!
[OK—16098/31744 bytes]
```

This command instructs the gateway to load the TCL call application script **debitcard.tcl** from the */tftpboot/tcl* subdirectory on the machine *mojo* and refer to it with the tag **debit**. The debit card application will then be available for association with an incoming dial peer and its corresponding destination number. When a user calls the number configured on that dial peer in the originating gateway, the debit card application is activated. After the application is loaded, it is available in the gateway and remains available until you enter the **no** form of the command or reboot the gateway. If this gateway is rebooted, the debitcard.tcl script will be loaded automatically.

You must receive successful feedback when loading the TCL script. If the application fails to load properly, it will not function and the entire debit card procedure will fail. Failure to load the debit card script is indicated by output such as the following:

```
pmeas1(config)# call application voice debit tftp://mojo/tcl/debitcard.tcl

%Error opening tftp://mojo/tcl/debitcard.tcl (Timed out)
Sep 30 09:41:00.565 UTC: %IVR-3-NOSCRIPT: Could not open IVR script
tftp://mojo/tcl/debitcard.tcl
errno=66568=
```

If you receive an error message similar to the one shown while trying to load the application, you must remedy that situation before you continue. The following example shows how to fix a failed call application load.

```
pmeas1(config)# show running-config
.
.
.
clock timezone PST -7
ip subnet-zero
no ip domain-lookup
ip domain-name cisco.com
ip name-server 192.168.8.69
ip name-server 192.168.0.21
!
cns event-service server
call application voice debit tftp://mojo/tcl/derbitcard.tcl
.
.
.
pmeas01(config)#
pmeas01(config)# call app voice debit tftp://mojo/tcl/debitcard.tcl
                                  ^
% Invalid input detected at '^' marker.
pmeas01(config)#
pmeas01(config)# no call application voice debit
Deleting TCL IVR App: debit with url
tftp://mojo/tcl/debitcard.tcl
pmeas01(config)#
pmeas01(config)# call app voice debit tftp://mojo/tcl/debitcard.tcl
Loading tcl/debitcard.tcl from 10.19.21.101 (via FastEthernet0): !!!!
[OK—16098/31744 bytes]
```

To fix a failed call application load, you must first remove the failed script. In this example, the script that failed to load properly is still displayed in the **show running-config** output. If you try to reload the application script with the same keyword (debit) before removing the failed script, you will be unsuccessful. In this example, the first attempt to reload the script is unsuccessful because the failed script was not removed. The output indicates that the keyword **debit** is still in use.

Scripts can fail to load for various reasons, but the most common reason is that there is a problem reaching the application file. Make sure you can manually reach the directory path (by using Telnet from the voice gateway) in which the application is located.

Configuring Call Application Parameters

At this point, if your prepaid services network was fully configured, a call to 555-1200 would return the built-in IVR response, “No prompts available.” The TCL call application script begins the IVR process, but because no configuration parameters detail the location of the audio files, it defaults to the embedded audio file noPromptPrompt, as illustrated in the following output from the **debug voip ivr all** command:

```
4w1d: $ $ta_PromptCmd() url=Tcl_GetVar2() [flash:enter_account.au]
4w1d: $ $ta_PromptCmd() Get prompt url=[flash:enter_account.au] name=[enter_account.au]
4w1d: $ $ta_PromptCmd() >>mc_createFromFileUrl
4w1d: $ $mc_createFromFileUrl (url:[flash:enter_account.au], name:[enter_account.au])::
4w1d: mc_load can not open flash:enter_account.au. errno=2=No such file or directory
4w1d: mc_load can not open flash:enter_account.au. errno=2=No such file or directory
4w1d: $ $mc_createFromFileUrl(name[enter_account.au]) load failed.
4w1d: $ $mc_createFromFileUrl(url[flash:enter_account.au]) load failed.
4w1d: $ $mc_delete()::
4w1d: $ $ta_PromptCmd() pArgs->content = 61D15750 noPromptPrompt;
4w1d: $ $ta_PromptCmd() >> ccGetApp(pcapp)
```

The call application parameters are contained in the configuration commands that govern the authentication parameters and location of the audio files. The following configuration example configures the UID and PIN lengths of the account number, and the location of the sets of language sound files used in the IVR scripts. This configuration is only necessary on originating gateways.

```
pmeas01(config)# call application voice debit uid-len 10
pmeas01(config)# call application voice debit pin-len 4
pmeas01(config)# call application voice debit language 1 en
Please make sure to use the corresponding set-location command
pmeas01(config)# call application voice debit set-location en 0 tftp://mojo/au/en/
pmeas01(config)# call application voice debit language 2 ch
Please make sure to use the corresponding set-location command
pmeas01(config)# call application voice debit set-location ch 0 tftp://mojo/au/ch/
```

The first two commands configure the length of the UID and the PIN. The lengths for the PIN and UID appear in the configuration output only if they are different from the default values.

The last four commands label and store the audio files. The **language 1 en** and **set-location en** lines go together in a pair as do the **language 2 ch** and **set-location ch** lines, as mentioned in the feedback from the session output.

In this example, **language 1 en** references the choice of language (in this case, English) and **set-location en** defines the machine or path where the files can be found. The number tag corresponds to the number that the IVR prompt will request (“Please press 1 for English”).

When a call is placed to the gateway, IVR looks in the audio file directory that was configured for the beginning IVR messages (en_welcome.au, en_lang_sel1.au and ch_lang_sel2.au). These three messages play the welcome message in English and the “select language” message in English and then Mandarin. Because these files must play to initiate the IVR process, they must be located in both language directories. In this case, because the **set-location** command for the Mandarin audio files was entered last (the **call application voice debit set-location ch 0 tftp://mojo/au/ch/** command), the TCL application looks in the Chinese audio file subdirectory for the welcome message and the select language option messages. Placing a call to the gateway with the **debug voip ivr all** command enabled produces the following output and confirms the expected behavior. Important items are bolded for emphasis.

```
4d11h: App debit: Handling callID 58
4d11h: callingNumber=408, calledNumber=5710961, redirectNumber=
4d11h: accountNumber=, finalDestFlag=0,
guid=86db.7ca8.8c6c.0096.0000.0000.1729.90ac
4d11h: peer_tag=1
4d11h: settlement_validate_token: cid(58), target=, tokenp=0x0
4d11h: :/acceptCall/
4d11h: Accepting CallID=58
4d11h: :/getVariable/
4d11h: :/setVariable/
!language type set to 1
4d11h: ta_SetVariableCmd.
4d11h: :/getVariable/
4d11h: :/setVariable/
!language type set to 2
4d11h: ta_SetVariableCmd.
4d11h: :/getVariable/
4d11h: :/setVariable/
!long pound enabled
4d11h: ta_SetVariableCmd.
4d11h: :[callID]
4d11h: :/puts/
4d11h: cid( 58) app running state select_language
4d11h: ta_PlayPromptCmd() 4d11h
4d11h: ta_PlayPromptCmd. CallID=58
4d11h: $ $pc_mc_makeDynamicS() calloc mcDynamicS_t
4d11h: $ $mc_createFromFileUrl (url:[tftp://mojo/au/ch/en_welcome.au], name:
[en_welcome.au])::
4d11h: $ $mc_getFromUrlName() en_welcome.au on ram mc_waitq_delete: mc=619219A4
mc_waitq_unlink: elm=61D88ECC
mc_waitq_unlink: prompt_free=2D2F4 prompt_active=0
mc_waitq_delete: prompt_free=2D2F4 prompt_active=6584
4d11h: $ $du_get_vpPromptName() OK###
4d11h: $ $du_mcDynamicS_silence() ms_int 1000 postSilence 1000
4d11h: $ $mc_createFromFileUrl (url:[tftp://mojo/au/ch/en_lang_sel1.au], nam
e:[en_lang_sel1.au])::
4d11h: $ $mc_getFromUrlName() en_lang_sel1.au on ram mc_waitq_delete: mc=61D8EFA 8
mc_waitq_unlink: elm=61DA3090
mc_waitq_unlink: prompt_free=291A8 prompt_active=6584
mc_waitq_delete: prompt_free=291A8 prompt_active=A6D0
4d11h: $ $du_get_vpPromptName() OK###
4d11h: $ $du_mcDynamicS_silence() ms_int 1000 postSilence 1000
4d11h: $ $mc_createFromFileUrl (url:[tftp://mojo/au/ch/ch_lang_sel2.au],
name: [ch_lang_sel2.au])::
4d11h: $ $mc_getFromUrlName() ch_lang_sel2.au on ram mc_waitq_delete: mc=61D8F070
mc_waitq_unlink: elm=61CD1EA0
mc_waitq_unlink: prompt_free=2537C prompt_active=A6D0
mc_waitq_delete: prompt_free=2537C prompt_active=E4FC
```

The debug output indicates that the IVR application is looking in the Chinese language subdirectory of the TFTP server for the English language audio files. This scenario is the proper behavior and the output is derived from a successful call.

After the IVR variables are correctly configured, the next step is to associate the debit card application with an incoming POTS dial peer.

Configuring Dial Peers

The debit card TCL application is initiated dynamically by reference to a POTS dial peer in the originating voice gateway. By matching a certain dialed number to a POTS dial peer, the debit card application begins a TCL application referenced in the dial peer. The following example shows a POTS dial peer configured for the originating gateway:

```
dial-peer voice 555 pots
 destination-pattern 5551200
 application debit
 port 0:D
```

In this example, if a user called the number 555-1200, it would match the configured POTS dial peer and activate the TCL application with the keyword **debit**, provided that the application exists and is reachable according to the application access commands configured in the gateway. The keyword **debit** is a tag given to the actual TCL script (debitcard.tcl) stored on the TFTP server.

You can display all the TCL scripts available in a gateway with the following **show** command:

```
pmeas11# show call application voice summary

name                description
session             Basic app to do DID, or supply dialtone.
fax_hop_on          Script to talk to a fax redialer
clid_authen         Authenticate with (ani, dnis)
clid_authen_collect Authenticate with (ani, dnis), collect if that fails
clid_authen_npw     Authenticate with (ani, NULL)
clid_authen_col_npw Authenticate with (ani, NULL), collect if that fails
clid_col_npw_3      Authenticate with (ani, NULL), and 3 tries collecting
clid_col_npw_npw    Authenticate with (ani, NULL) and 3 tries without pw
SESSION             Default system session application
debit               tftp://mojo/tcl/debitcard.tcl
```

To display an application in its entirety, use the appropriate name of the application instead of the **summary** keyword. For example, the **show call application voice debit** privileged EXEC command would display the debitcard.tcl script.

The following example shows a VoIP dial peer configured for the terminating gateway:

```
dial-peer voice 514 voip
 destination-pattern 1514…….
 session target ipv4:10.10.12.23

dial-peer voice 213 voip
 destination-pattern 1213…….
 session target ipv4:10.20.120.14
```

VoIP dial peers associate a number dialed to a network destination—in this case an IP address. The terminating gateway VoIP dial peer need not be associated with an application.

Configuring AAA

In order for the debit card application to work with the RADIUS server to collect the appropriate connection accounting information, you must at least configure the **aaa new-model** global configuration command on your voice gateways. The following example configures AAA to use RADIUS on the gateways. The same configuration is used on both the originating and terminating gateways.

```
aaa new-model
aaa authentication login h323 group radius
aaa authentication login NONE none
aaa authorization exec h323 group radius
aaa accounting connection h323 start-stop group radius

line con 0
 login authentication NONE

gw-accounting h323 vsa
```

In this example, AAA is enabled and a named list called h323 defines that RADIUS should be used to provide authentication, authorization, and accounting. The named list NONE enables network administrators to log in to the console port and bypass authentication. CDRs will be delivered to the RADIUS server using the VSA method.

Configuring RADIUS and VSA Parameters

In the following example, both the originating and terminating gateways are configured to support RADIUS. The same configuration is used on both gateways. (Note that you must enable AAA before you can configure any RADIUS parameters.)

```
radius-server host 172.22.42.49 auth-port 1645 acct-port 1646
radius-server key testing123
radius-server vsa send accounting
radius-server vsa send authentication
```

In this example, the first command configures the IP address of the RADIUS server and the ports on which the gateway expects the RADIUS server to be listening. The authentication and accounting ports are the default ports for Cisco gateways. The second command configures the password used for authenticating the RADIUS server. The last two commands enable the use of VSAs for H.323 authentication and accounting.

Configuring NTP

You must configure NTP in order to pass time stamps that are synchronized between the gateways and servers. To instruct the gateway to synchronize its time-of-day clock with an NTP server, use the **ntp server** *ip-address* global configuration command. For test purposes, to enable one of the gateways to act as the authoritative NTP server, issue the **ntp master** global configuration command. Then other gateways can synchronize with the master by pointing to the master in their configurations with the **ntp server** *ip-address* global configuration command.

The clocks on Cisco routers, however, are typically stratum-8 clocks as opposed to stratum-1 and stratum-2 NTP servers. Stratum 8 clocks are not considered accurate enough to keep time for a production network.

Internally Managed Prepaid Services Call Example

This section contains sample debug output from a successful internally managed prepaid services call. In this example, a pair of gateways are used to make a debit card call with a MindCTI RADIUS back-end billing system performing the AAA. This example shows the behavior and characteristics of the debit card application. Basic voice connectivity has been established, so this example does not verify the call control system. However, we have set some billing debug commands so that you will be able to see how IVR and AAA operate. The two debug sessions described in the following sections are the output received for a successful call, then long pound, then another successful call:

- Originating Gateway Debug Output
- Terminating Gateway Debug Output

Originating Gateway Debug Output

The following debug commands were activated on the originating gateway:

```
debug voip ivr all
debug radius
debug aaa authentication
debug aaa authorization
debug aaa accounting
```

In the following output, a call was placed through a PBX. The number that the PBX sends out is configurable—the PBX in this example was configured to send the area code for San Jose (408).

Note In this debug output, pertinent information is bolded. Added comments are in regular type, and unnecessary parts are deleted. Look for successful voice prompt loads, RADIUS authentication and authorization transactions, VSA output, and RADIUS return codes.

```
pmeas11#
1w6d: AAA: parse name=<no string> idb type=-1 tty=-1
1w6d: AAA/MEMORY: create_user (0x61BEFB38) user='408' ruser='5550961' port='' re
m_addr='408/5710961' authen_type=NONE service=H323_VSA priv=0
1w6d: AAA/ACCT/CONN: Found list "h323"
```

These lines refer to the AAA accounting list in the gateway:

```
1w6d: AAA/ACCT/CONN/START User 408, Port , Location "unknown"
1w6d: AAA/ACCT/CONN/START User 408, Port ,
task_id=81 start_time=939938654 timezone=PST service=connection protocol=h323
1w6d: AAA/ACCT: user 408, acct type 1 (397058159): Method=radius (radius)
1w6d: RADIUS: ustruct sharecount=2
1w6d: RADIUS: added cisco VSA 33 len 23 "h323-gw-id=sj7_pmeas11."
1w6d: RADIUS: added cisco VSA 24 len 41 "h323-conf-id=86DB7CA8 8C6C016E 0 466555A0"
1w6d: RADIUS: added cisco VSA 26 len 23 "h323-call-origin=answer"
1w6d: RADIUS: added cisco VSA 27 len 24 "h323-call-type=Telephony"
1w6d: RADIUS: added cisco VSA 25 len 48 "h323-setup-time=14:03:46.180 PST Thu Oc
t 14 1999"
1w6d: App debit: Handling callID 142
1w6d: callingNumber=408,
```

In the following output, all the call setup variables are listed, both standard IETF RADIUS attributes and VSAs. The start record is for the answer telephony call leg (call leg 1). IVR is shown beginning next.

```
calledNumber=5710961, redirectNumber=
1w6d: accountNumber=, finalDestFlag=0,
guid=86db.7ca8.8c6c.016e.0000.0000.4665.55a0
1w6d: peer_tag=1
1w6d: RADIUS: Initial Transmit id 19 172.22.42.52:1646, Accounting-Request, len278
1w6d: Attribute 4 6 AC162758
1w6d: Attribute 61 6 00000000
1w6d: Attribute 1 5 3430381E
1w6d: Attribute 30 9 35373130
1w6d: Attribute 31 5 34303828
1w6d: Attribute 40 6 00000001 //start record
1w6d: Attribute 6 6 00000001
1w6d: Attribute 26 31 0000000921196833 //h323-gw-id
1w6d: Attribute 26 49 00000009182B6833 //h323-conf-id
1w6d: Attribute 26 31 000000091A196833 //answer
1w6d: Attribute 26 32 000000091B1A6833 //telephony
1w6d: Attribute 26 56 0000000919326833 //setup time
1w6d: Attribute 44 10 30303030
1w6d: Attribute 41 6 00000000
```

In this output, English is set as language number 1 and Chinese is set as language number 2:

```
1w6d: settlement_validate_token: cid(142), target=, tokenp=0x0
1w6d: :/acceptCall/
1w6d: Accepting CallID=142
1w6d: :/getVariable/
1w6d: :/setVariable/
1w6d: ta_SetVariableCmd. language type set to 1
1w6d: :/getVariable/
1w6d: :/setVariable/
1w6d: ta_SetVariableCmd. language type set to 2
```

The long-pound feature is enabled, which means the user can press the pound symbol to make another call without needing to be reauthenticated:

```
1w6d: :/getVariable/
1w6d: :/setVariable/
1w6d: ta_SetVariableCmd. long pound enabled
```

The welcome message is loaded from the /tftpboot/au/ch subdirectory of the TFTP server and copied into RAM and played:

```
1w6d: :[callID]
1w6d: :/puts/
1w6d: cid( 142) app running state select_language
1w6d: ta_PlayPromptCmd() 1w6d
1w6d: ta_PlayPromptCmd. CallID=142
1w6d: $ $pc_mc_makeDynamicS() calloc mcDynamicS_t
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/ch/en_welcome.au], name:[
en_welcome.au])::
```

The prompt, “Select 1 for English,” is loaded and played:

```
1w6d: $ $mc_getFromUrlName() en_welcome.au on ram mc_waitq_delete: mc=619219A4
mc_waitq_unlink: elm=61CD27B0
mc_waitq_unlink: prompt_free=A433B prompt_active=0
mc_waitq_delete: prompt_free=A433B prompt_active=6584
1w6d: $ $du_get_vpPromptName() OK###
1w6d: $ $du_mcDynamicS_silence() ms_int 1000 postSilence 1000
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/ch/en_lang_sel1.au], name
:[en_lang_sel1.au])::
```

The Chinese prompt, “Select 2 for Chinese,” is loaded and played:

```
1w6d: $ $mc_getFromUrlName() en_lang_sel1.au on ram mc_waitq_delete: mc=61D8EFA8
mc_waitq_unlink: elm=61D82198
mc_waitq_unlink: prompt_free=A01EF prompt_active=6584
mc_waitq_delete: prompt_free=A01EF prompt_active=A6D0
1w6d: $ $du_get_vpPromptName() OK###
1w6d: $ $du_mcDynamicS_silence() ms_int 1000 postSilence 1000
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/ch/ch_lang_sel2.au], name
:[ch_lang_sel2.au])::
```

After the language is chosen, IVR begins the authentication process by loading and playing the “Enter card number” prompt. Now that the IVR application knows that the chosen language is English, it goes to the /au/en subdirectory for the files:

```
1w6d: $ $mc_getFromUrlName() ch_lang_sel2.au on ram mc_waitq_delete: mc=61D8F070
.
.
.
1w6d: $ $pc_mc_makeDynamicS() calloc mcDynamicS_t
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/en/en_enter_card_num.au],
name:[en_enter_card_num.au])::
1w6d: $ $mc_getFromUrlName() en_enter_card_num.au on ram mc_waitq_delete: mc=61921618
```

The script returns a success upon matching the card number against the database:

```
1w6d: pcapp CallID 142 returning PCAPP_MATCHED. string=0000701234
1w6d: $ $pcapp_finished() >>pcapp_return()
1w6d: $ $pcapp_finished() >>ms_delete()
```

The following RADIUS records concern the authentication phase of the call:

```
1w6d: cid(142) ta_get_event returning collect success
1w6d: :[called]
1w6d: :/puts/
1w6d: cid( 142) app running state second_authorize
1w6d: :/ani/
1w6d: :[authorize]
1w6d: authorization
1w6d: account=000070 the account number is broken out into uid
1w6d: password=1234 and pin numbers
1w6d: destination= destination is blank because it has not yet been entered
1w6d: password=1234
1w6d: AAA: parse name=<no string> idb type=-1 tty=-1
1w6d: AAA/MEMORY: create_user (0x61C27260) user='000070' ruser='' port='' rem_ad
dr='408/5255233' authen_type=ASCII service=LOGIN priv=0
1w6d: unknown AAA/AUTHOR/EXEC (1758884971): Port='' list='h323' service=EXEC
1w6d: AAA/AUTHOR/EXEC: unknown (1758884971) user='000070'
1w6d: unknown AAA/AUTHOR/EXEC (1758884971): found list "h323"
1w6d: unknown AAA/AUTHOR/EXEC (1758884971): Method=radius (radius)
```

The call is assigned a conference ID. This number will be consistent through all call legs of this call. It is used to identify records concerning the same call.

```
1w6d: RADIUS: authenticating to get author data
1w6d: RADIUS: ustruct sharecount=2
1w6d: RADIUS: added cisco VSA 24 len 41 "h323-conf-id=86DB7CA8 8C6C016E 0 466555A0"
1w6d: RADIUS: Initial Transmit id 21 172.22.42.52:1645, Access-Request, len 121
1w6d: Attribute 4 6 AC162758
1w6d: Attribute 61 6 00000000
1w6d: Attribute 1 8 30303030
1w6d: Attribute 26 49 00000009182B6833
1w6d: Attribute 30 9 35323535
1w6d: Attribute 31 5 34303802
1w6d: Attribute 2 18 7B8D2364
1w6d: RADIUS: Received from id 21 172.22.42.52:1646, Access-Accept, len 76
```

The following RADIUS VSAs are received from the RADIUS server in response to the authentication request. The user credit amount is kept in the RADIUS database and is accessed in real time at the time of the call. The h323-credit-amount will be played to the caller via the TCL IVR scripts. The script will build the amount $32.91 from (thirty) and (two) and (dollars) and (ninety) and (one) and (cents). The credit amount shows up as a minus so that credit spent on a call will be added to the total until the credit amount reaches 0. Note that not all attributes returned are supported.

```
1w6d: Attribute 26 26 0000000967146833 h323-return-code
1w6d: Attribute 26 30 000000096B186833 h323-preferred-lang
1w6d: Attribute 26 33 00000009651B6833 h323-credit-amount
1w6d: Attribute 26 23 000000096D116269 unsupported VSA
1w6d: Attribute 26 25 000000096E136375 unsupported VSA
1w6d: RADIUS: saved authorization data for user 61C4C2D0 at 61C4C3E4
1w6d: RADIUS: cisco AVPair ":h323-return-code=0"
1w6d: RADIUS: cisco AVPair ":h323-preferred-lang=en"
1w6d: RADIUS: cisco AVPair ":h323-credit-amount=-32.91"
1w6d: RADIUS: unrecognized cisco VS option 109
1w6d: RADIUS: Bad attribute (unsupported attribute): type 26 len 23 data 0x9
1w6d: RADIUS: unrecognized cisco VS option 110
1w6d: RADIUS: Bad attribute (unsupported attribute): type 26 len 25 data 0x9
```

After authentication is passed, the system prompts for the destination number:

```
1w6d: AAA/AUTHOR (3803615862): Post authorization status = PASS_ADD
.
.
.
1w6d: cid( 142) app running state get_dest
1w6d: :/getVariable/
1w6d: ta_PlayPromptCmd() 1w6d
1w6d: ta_PlayPromptCmd. CallID=142
1w6d: $ $du_get_vpPromptName() OK###
1w6d: $ $du_mcDynamicS_silence() ms_int 1000 postSilence 1000
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/en/en_enter_dest.au], nam
e:[en_enter_dest.au])::
```

The “Enter destination” prompt is loaded and played and the user enters the destination number. The destination string entered is 555-5233.

```
1w6d: $ $mc_getFromUrlName() en_enter_dest.au on ram mc_waitq_delete: mc=61D9F56
0
.
.
.
1w6d: pcapp CallID 142 returning PCAPP_MATCHED. string=5555233
.
.
.
1w6d: cid(142) ta_get_event returning collect success
1w6d: :[callID]
1w6d: :/puts/
1w6d: cid( 142) app running state second_authorize
1w6d: :/ani/
1w6d: :[authorize]
1w6d: authorization
```

Authorization is performed for the particular destination:

```
1w6d: account=000070
same UID and PIN as deserved in CDR for the first call.
1w6d: password=1234
1w6d: destination=5555233
```

Now the destination is added:

```
1w6d: AAA: parse name=<no string> idb type=-1 tty=-1
1w6d: AAA/MEMORY: create_user (0x61C27260) user='000070' ruser='' port='' rem_ad
dr='408/5555233' authen_type=ASCII service=LOGIN priv=0
1w6d: unknown AAA/AUTHOR/EXEC (1758884971): Port='' list='h323' service=EXEC
1w6d: AAA/AUTHOR/EXEC: unknown (1758884971) user='000070'
1w6d: unknown AAA/AUTHOR/EXEC (1758884971): found list "h323"
```

The AAA authorization list configured in the router is found. The user is authorized for a call to 555-5233 and the time left for this destination is returned as 20900 seconds (5 hours, 48 minutes, and 20 seconds).

```
1w6d: unknown AAA/AUTHOR/EXEC (1758884971): Method=radius (radius)
1w6d: RADIUS: authenticating to get author data
1w6d: RADIUS: ustruct sharecount=2
1w6d: RADIUS: added cisco VSA 24 len 41 "h323-conf-id=86DB7CA8 8C6C016E 0 466555A0"
1w6d: RADIUS: Initial Transmit id 21 172.22.42.52:1645, Access-Request, len 121
1w6d: Attribute 4 6 AC162758
1w6d: Attribute 61 6 00000000
1w6d: Attribute 1 8 30303030
1w6d: Attribute 26 49 00000009182B6833
1w6d: Attribute 30 9 35323535
1w6d: Attribute 31 5 34303802
1w6d: Attribute 2 18 7B8D2364
1w6d: RADIUS: Received from id 21 172.22.42.52:1646, Access-Accept, len 76
1w6d: Attribute 26 26 0000000967146833
1w6d: Attribute 26 30 0000000966186833
1w6d: RADIUS: saved authorization data for user 61C27260 at 61C4C4E4
1w6d: RADIUS: cisco AVPair ":h323-return-code=0"
1w6d: RADIUS: cisco AVPair ":h323-credit-time=20900"
1w6d: AAA/AUTHOR (1758884971): Post authorization status = PASS_ADD
```

In the following output, the authorized user is prompted with the remaining time available (5 hours and 48 minutes) for a call to the specified destination:

```
1w6d: ta_PlayPromptCmd. CallID=142
1w6d: $ $pc_mc_makeDynamicS() calloc mcDynamicS_t
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/en/en_you_have.au], name:
[en_you_have.au])::
.
.
.
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/en/en_five.au], name:[en_
five.au])::
.
.
.
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/en/en_hours.au], name:[en
_hours.au])::
.
.
.

1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/en/en_and.au], name:[en_a
nd.au])::
.
.
.
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/en/en_forty.au], name:[en
_forty.au])::
.
.
.
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/en/en_eight.au], name:[en
_eight.au])::
.
.
.
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/en/en_minutes.au], name:
[en_minutes.au])::
```

The following RADIUS debug output is the start packet, call leg 2, for the active call. The VSAs, spelled out at the top of this section, deliver call details to the RADIUS server.

```
1w6d: Placing call for callID 142 to destination=5555233
1w6d: placecall CallID 142 got event CC_EV_CALL_HANDOFF
1w6d: Matched peers(1)
1w6d: placecall pc_setupPeer cid(142), destPat(5555233), matched(1), prefix(), p
eer(61C2BD34)
1w6d: placecall cid(142) state change PC_CS_INIT to PC_CS_CALL_SETTING
.
.
.
1w6d: RADIUS: ustruct sharecount=2
1w6d: RADIUS: added cisco VSA 33 len 23 "h323-gw-id=sj7_pmeas11."
1w6d: RADIUS: added cisco VSA 24 len 41 "h323-conf-id=86DB7CA8 8C6C016E 0 466555
A0"
1w6d: RADIUS: added cisco VSA 26 len 26 "h323-call-origin=originate"
1w6d: RADIUS: added cisco VSA 27 len 19 "h323-call-type=VoIP"
1w6d: RADIUS: added cisco VSA 25 len 48 "h323-setup-time=14:04:54.450 PST Thu Oc
t 14 1999"
1w6d: RADIUS: Initial Transmit id 22 172.22.42.52:1646, Accounting-Request, len279
1w6d: Attribute 4 6 AC162758
1w6d: Attribute 61 6 00000000
1w6d: Attribute 1 8 30303030
1w6d: Attribute 30 9 35323535
1w6d: Attribute 31 5 34303828
1w6d: Attribute 40 6 00000001
1w6d: Attribute 6 6 00000001
1w6d: Attribute 26 31 0000000921196833
1w6d: Attribute 26 49 00000009182B6833
1w6d: Attribute 26 34 000000091A1C6833
1w6d: Attribute 26 27 000000091B156833
1w6d: Attribute 26 56 0000000919326833
1w6d: Attribute 44 10 30303030
1w6d: Attribute 41 6 00000000
1w6d: RADIUS: Received from id 22 172.22.42.52:1646, Accounting-response, len 46
1w6d: Attribute 26 26 0000000967146833
```

In the following output, the user presses the pound symbol 543 seconds (9 minutes and 3 seconds) into the call:

```
1w6d: cid(142) ta_get_event returning active
1w6d: :[callID]
1w6d: :/puts/
1w6d: cid( 142) app running state active
1w6d: :/startTimer/
1w6d: Wait for 21380 seconds Now the timer starts for the call
1w6d: cid(142) ta_get_event returning digit
1w6d: ta_StartTimerCmd(): ta_get_event [digit]
1w6d: :/startTimer/
1w6d: Wait for 20837 seconds the long pound is pressed 543 seconds into the call
1w6d: cid(142) ta_get_event returning longpound
1w6d: ta_StartTimerCmd(): ta_get_event [longpound]
```

The following output is the stop record for the first call. Notice the conf-id for comparison. We know it is a stop record because it has connect-time, disconnect-time, and disconnect-cause (only present in stop records) and it is type “originate VoIP” (call leg 2).

```
1w6d: RADIUS: ustruct sharecount=1
1w6d: RADIUS: added cisco VSA 33 len 23 "h323-gw-id=sj7_pmeas11."
1w6d: RADIUS: added cisco VSA 24 len 41 "h323-conf-id=86DB7CA8 8C6C016E 0 466555A0"
1w6d: RADIUS: added cisco VSA 26 len 26 "h323-call-origin=originate"
1w6d: RADIUS: added cisco VSA 27 len 19 "h323-call-type=VoIP"
1w6d: RADIUS: added cisco VSA 25 len 48 "h323-setup-time=14:04:54.450 PST Thu Oc
t 14 1999"
1w6d: RADIUS: added cisco VSA 28 len 50 "h323-connect-time=14:05:02.260 PST Thu
Oct 14 1999"
1w6d: RADIUS: added cisco VSA 29 len 53 "h323-disconnect-time=14:05:34.740 PST T
hu Oct 14 1999"
1w6d: RADIUS: added cisco VSA 30 len 24 "h323-disconnect-cause=10"
1w6d: RADIUS: added cisco VSA 31 len 20 "h323-voice-quality=0"
1w6d: RADIUS: added cisco VSA 23 len 30 "h323-remote-address=10.10.1.15"
1w6d: :[callID]
1w6d: :/puts/
1w6d: cid( 142) app running state first_authorize
1w6d: :/ani/
1w6d: :[authorize]
1w6d: authorization
1w6d: account=000070
1w6d: password=1234
1w6d: destination=
```

In the following output, the next call is authorized. Notice that the conf-id is the same as the first call. It is, in effect, the same session. A new credit amount is given and a prompt is made for a new destination. This call proceeds in the same manner as the previous one.

```
1w6d: AAA: parse name=<no string> idb type=-1 tty=-1
1w6d: AAA/MEMORY: create_user (0x61C67F20) user='000070' ruser='' port='' rem_ad
dr='408' authen_type=ASCII service=LOGIN priv=0
1w6d: unknown AAA/AUTHOR/EXEC (3696672751): Port='' list='h323' service=EXEC
1w6d: AAA/AUTHOR/EXEC: unknown (3696672751) user='000070'
1w6d: unknown AAA/AUTHOR/EXEC (3696672751): found list "h323"
1w6d: unknown AAA/AUTHOR/EXEC (3696672751): Method=radius (radius)
1w6d: RADIUS: authenticating to get author data
1w6d: RADIUS: ustruct sharecount=2
1w6d: RADIUS: added cisco VSA 24 len 41
"h323-conf-id=86DB7CA8 8C6C016E 0 466555A0"
.
.
.
1w6d: RADIUS: saved authorization data for user 61C67F20 at 620157E0
1w6d: RADIUS: cisco AVPair ":h323-return-code=0"
1w6d: RADIUS: cisco AVPair ":h323-preferred-lang=en"
1w6d: RADIUS: cisco AVPair ":h323-credit-amount=-33.75"
.
.
.
1w6d: $ $mc_createFromFileUrl (url:[tftp://mojo/au/en/en_enter_dest.au], nam
e:[en_enter_dest.au])::
1w6d: $ $mc_getFromUrlName() en_enter_dest.au on ram mc_waitq_delete: mc=61D9F560
.
.
.
1w6d: prompt and collect app got callID 142
.
.
.
1w6d: $ $mc_make_packets_DQ()::
1w6d: $ $mc_make_packets_DQ() post pak silence = 1000
1w6d: $ $mc_make_packets_DQ() mc:61D9F560 name:en_enter_dest.au
1w6d: $ $mc_make_packets_DQ() count: 36 ##
.
.
.
1w6d: pcapp CallID 142 returning PCAPP_MATCHED. string=5554094
.
.
.
1w6d: cid(142) ta_get_event returning collect success
.
.
.
1w6d: RADIUS: added cisco VSA 33 len 23 "h323-gw-id=sj7_pmeas11."
1w6d: RADIUS: added cisco VSA 24 len 41 "h323-conf-id=86DB7CA8 8C6C016E 0 466555A0"
1w6d: RADIUS: added cisco VSA 26 len 26 "h323-call-origin=originate"
1w6d: RADIUS: added cisco VSA 27 len 19 "h323-call-type=VoIP"
1w6d: RADIUS: added cisco VSA 25 len 48 "h323-setup-time=14:05:58.760 PST Thu Oc
t 14 1999"
.
.
.
1w6d: ta_StartTimerCmd(): ta_get_event [digit]
1w6d: :/startTimer/
1w6d: Wait for 20794 seconds
```

The following output is another stop record created at the termination of the second call. Connect and disconnect times are sent to the RADIUS server.

```
1w6d: AAA/ACCT: user 408, acct type 1 (953168740): Method=radius (radius)
1w6d: RADIUS: ustruct sharecount=1
1w6d: RADIUS: added cisco VSA 33 len 23 "h323-gw-id=sj7_pmeas11."
1w6d: RADIUS: added cisco VSA 24 len 41 "h323-conf-id=86DB7CA8 8C6C016E 0 466555A0"
1w6d: RADIUS: added cisco VSA 26 len 23 "h323-call-origin=answer"
1w6d: RADIUS: added cisco VSA 27 len 24 "h323-call-type=Telephony"
1w6d: RADIUS: added cisco VSA 25 len 48 "h323-setup-time=14:03:46.180 PST Thu Oc
t 14 1999"
1w6d: RADIUS: added cisco VSA 28 len 50 "h323-connect-time=14:03:46.200 PST ThuOct 14
1999"
1w6d: RADIUS: added cisco VSA 29 len 53 "h323-disconnect-time=14:07:36.320 PST T
hu Oct 14 1999"
1w6d: RADIUS: added cisco VSA 30 len 24 "h323-disconnect-cause=10"
1w6d: RADIUS: added cisco VSA 31 len 20 "h323-voice-quality=0"
1w6d: cid(142) incoming disconnected
1w6d: cid(0) ta_get_event returning incoming disconnected
1w6d: TCL script eval for callID 142 completed. code=OK
1w6d: incoming disconnected
1w6d: RADIUS: Initial Transmit id 29 172.22.42.52:1646, Accounting-Request, len
.
.
.
1w6d: AAA/MEMORY: free_user (0x61BEFB38) user='408' ruser='5550961' port='' rem_
addr='408/5554094' authen_type=NONE service=H323_VSA priv=0
sj7_pmeas11#
```

Terminating Gateway Debug Output

The following debug commands were activated on the terminating gateway:

```
debug radius
deb aaa authentication
debug aaa authorization
debug aaa accounting
```

The following debug output shows the activity on the terminating gateway for the activities and calls described in the previous debug output. Because IVR is not running on the terminating gateway, we have restricted debug data to AAA and RADIUS.

The important items in this output are those that compare the times and billing amounts to the originating gateway records. From this output you can see that the records collected from either gateway are sufficient to generate accurate billing records.

Notice in the following output that the conf-ID values match. There is a 2-second difference, however, in the setup times. This delay reflects the time it took to make the call setup.

```
sj7_pmeas01#
Oct 14 14:05:22.600 PST: AAA: parse name=<no string> idb type=-1 tty=-1
Oct 14 14:05:22.600 PST: AAA/MEMORY: create_user (0x61C024B0) user='408' ruser='
5255233' port='' rem_addr='408/5555233' authen_type=NONE service=H323_VSA priv=0
Oct 14 14:05:22.600 PST: AAA/ACCT/CONN: Found list "h323"
Oct 14 14:05:22.600 PST: AAA/ACCT/CONN/START User 408, Port , Location "unknown"
Oct 14 14:05:22.600 PST: AAA/ACCT/CONN/START User 408, Port ,
task_id=56 start_time=939938722 timezone=PST service=connection protocol=h323
Oct 14 14:05:22.600 PST: AAA/ACCT: user 408, acct type 1 (2416182195): Method=ra
dius (radius)
Oct 14 14:05:22.600 PST: RADIUS: ustruct sharecount=2
Oct 14 14:05:22.600 PST: RADIUS: added cisco VSA 33 len 23 "h323-gw-id=sj7_pmeas01."
Oct 14 14:05:22.604 PST: RADIUS: added cisco VSA 24 len 41 "h323-conf-id=86DB7CA8
8C6C016E 0 466555A0"
Oct 14 14:05:22.604 PST: RADIUS: added cisco VSA 26 len 23 "h323-call-origin=answer"
Oct 14 14:05:22.604 PST: RADIUS: added cisco VSA 27 len 19 "h323-call-type=VoIP"
Oct 14 14:05:22.604 PST: RADIUS: added cisco VSA 25 len 48 "h323-setup-time=14:04:56.620
PST Thu Oct 14 1999"
```

The following information displayed is the start record for call leg 4 (originate Telephony):

```
Oct 14 14:05:22.756 PST: RADIUS: added cisco VSA 33 len 23 "h323-gw-id=sj7_pmeas 01."
Oct 14 14:05:22.756 PST: RADIUS: added cisco VSA 24 len 41 "h323-conf-id=86DB7CA
8 8C6C016E 0 466555A0"
Oct 14 14:05:22.756 PST: RADIUS: added cisco VSA 26 len 26 "h323-call-origin=originate"
Oct 14 14:05:22.756 PST: RADIUS: added cisco VSA 27 len 24 "h323-call-type=Telephony"
Oct 14 14:05:22.756 PST: RADIUS: added cisco VSA 25 len 48 "h323-setup-time=14:0
4:56.770 PST Thu Oct 14 1999"
Oct 14 14:05:22.756 PST: RADIUS: Initial Transmit id 155 172.22.42.52:1646,
Accounting-Request, len 281
Oct 14 14:05:22.756 PST: Attribute 4 6 AC16275A
Oct 14 14:05:22.756 PST: Attribute 61 6 00000000
Oct 14 14:05:22.756 PST: Attribute 1 5 3430381E
Oct 14 14:05:22.756 PST: Attribute 30 9 35323535
Oct 14 14:05:22.756 PST: Attribute 31 5 34303828
Oct 14 14:05:22.756 PST: Attribute 40 6 00000001
Oct 14 14:05:22.756 PST: Attribute 6 6 00000001
Oct 14 14:05:22.756 PST: Attribute 26 31 0000000921196833
Oct 14 14:05:22.756 PST: Attribute 26 49 00000009182B6833
Oct 14 14:05:22.756 PST: Attribute 26 34 000000091A1C6833
Oct 14 14:05:22.756 PST: Attribute 26 32 000000091B1A6833
Oct 14 14:05:22.756 PST: Attribute 26 56 0000000919326833
Oct 14 14:05:22.760 PST: Attribute 44 10 30303030
Oct 14 14:05:22.760 PST: Attribute 41 6 00000000
Oct 14 14:05:22.772 PST: RADIUS: Received from id 155 172.22.42.52:1646,
Accounting-response, len 46
Oct 14 14:05:22.772 PST: Attribute 26 26 0000000967146833
```

The connect time is approximately 1 second after call leg 2.

```
Oct 14 14:06:02.885 PST: RADIUS: added cisco VSA 33 len 23 "h323-gw-id=sj7_pmeas01."
Oct 14 14:06:02.885 PST: RADIUS: added cisco VSA 24 len 41 "h323-conf-id=86DB7CA
8 8C6C016E 0 466555A0"
Oct 14 14:06:02.885 PST: RADIUS: added cisco VSA 26 len 23 "h323-call-origin=answer"
Oct 14 14:06:02.885 PST: RADIUS: added cisco VSA 27 len 19 "h323-call-type=VoIP"
Oct 14 14:06:02.885 PST: RADIUS: added cisco VSA 25 len 48 "h323-setup-time=14:0
4:56.620 PST Thu Oct 14 1999"
Oct 14 14:06:02.885 PST: RADIUS: added cisco VSA 28 len 50 "h323-connect-time=14
:05:03.780 PST Thu Oct 14 1999"
```

The disconnect time is about 2 seconds after call leg 2. The time of the call (disconnect-time minus connect-time) is 32.48 seconds for call leg 2 and 33.12 seconds for call leg 3 (a 0.640-second difference between call leg 2 and call leg 3).

```
Oct 14 14:06:02.885 PST: RADIUS: added cisco VSA 29 len 53 "h323-disconnect-time
=14:05:36.900 PST Thu Oct 14 1999"
```

OSP for Clearinghouse Services Theory

Packet telephony service providers interested in expanding their geographic coverage have been faced with limited options. To help alleviate this problem, Cisco has implemented the Open Settlement Protocol (OSP), a client/server protocol defined by the ETSI TIPHON standards organization. OSP is designed to offer billing and accounting record consolidation for voice calls that traverse ITSP boundaries; it also allows service providers to exchange traffic with other service providers without establishing multiple bilateral peering agreements.

Because of OSP, you can employ a reliable third-party clearinghouse to handle VoIP call termination while leveraging the bandwidth efficiencies and tariff arbitrage advantages that are inherent in IP. You can use a clearinghouse as both a technical and business bridge; by signing on with such an organization and using OSP, you can extend service beyond the boundaries of your own network and immediately access the entire clearinghouse network of affiliated service providers.

OSP Background

In the time-division multiplexing (TDM) circuit-switched world, interconnecting carriers calculated settlements based on minutes used in circuits exchanged between their switches, often exchanging Signaling System 7 (SS7) information and voice paths. Call authorization was based on the physical demarcation point; if a call arrived, it was deemed “authorized.” This scenario required a stable business relationship except in the case of international traffic, where third-party wholesale carriers often provided such services.

VoIP service providers have adapted to such arrangements by terminating calls on the PSTN and reoriginating the call on a circuit switch. However, such an approach limits the cost-effectiveness of currently available packet telephony. Even interconnection between VoIP networks was problematic—solutions were usually tightly integrated with the proprietary and nonstandard implementations of H.323 protocols by individual vendors.

OSP avoids this problem. By allowing gateways to transfer accounting and routing information securely, this protocol provides common ground between VoIP service providers.

Third-party clearinghouses with an OSP server can offer route selection, call authorization, call accounting, and intercarrier settlements, including all the complex rating and routing tables necessary for efficient and cost-effective interconnections. Cisco has worked with a variety of leading OSP clearinghouses to ensure interoperability with their OSP server applications. OSP-based clearinghouses provide the least cost and the best route selection algorithms based on a variety of parameters their subscriber carriers provide, including cost, quality, and specific carrier preferences. Prepaid calling services, click-to-dial, and clearinghouse settlements can be offered over the same packet infrastructure.

OSP Clearinghouses Benefits

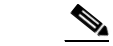
The OSP clearinghouse solution gives virtually all VoIP providers the worldwide calling reach they require. This service can be used separately or in conjunction with internally managed prepaid calling services. The benefits of using an OSP clearinghouse include the following:

- End-to-end VoIP support
- Cost-effective worldwide calling coverage
- Guaranteed settlement of authorized calls
- Incremental revenue increase by terminating calls from other service providers
- Simplified business and credit relationships
- Outsourced complex rating and routing tables
- Flexibility in selecting appropriate termination points
- Secure transmission using widely accepted encryption for sensitive data
- Single authentication for the actual gateway or platform at initialization time
- Secure interface between the settlement client and the server settlement

OSP Clearinghouse Operation and Call Flow

The following list describes a high-level call flow sequence for an OSP clearinghouse application:

1. A customer places a call via the PSTN network to a VoIP gateway, which authenticates the customer by communicating with a RADIUS server.
2. The originating VoIP gateway attempts to locate a termination point within its own network by communicating with a gatekeeper using H.323 RAS. If there is no appropriate route, the gatekeeper informs the gateway to search for a termination point elsewhere.
3. The gateway contacts an OSP server at the third-party clearinghouse. The gateway then establishes a Secure Socket Layer (SSL) connection to the OSP server and sends an authorization request to the clearinghouse. The authorization request contains pertinent information about the call, including the destination number, the device ID, and customer ID of the gateway.
4. The OSP server processes the information and, assuming the gateway is authorized, returns routing details for the possible terminating gateways that can satisfy the request of the originating gateway.



Note Depending on how you implement OSP, most OSP servers can supply the least-cost and best-route selection algorithms according to your requirements for cost, quality, and other parameters, selecting up to three routes.

5. The clearinghouse creates an authorization token and then replies to the originating gateway with a token and up to three selected routes. If any or all of the three routes have identical cost and quality parameters, the settlement server randomizes the qualifying routes. The originating gateway uses the IP addresses supplied by the clearinghouse to set up the call.

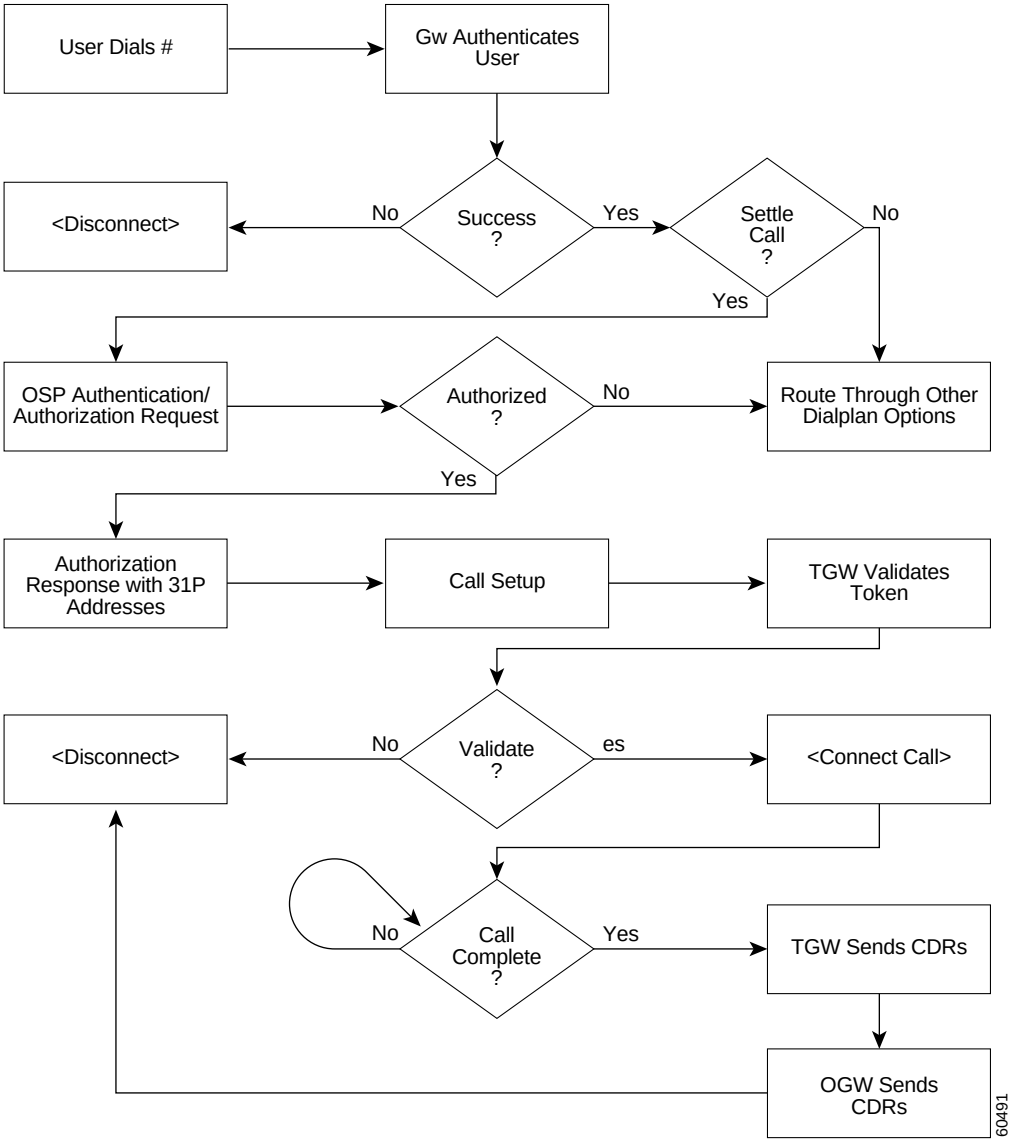
Note To interoperate with Cisco gateways, the token should be signed but not encrypted, conforming to PKCS#7 for Signed Data. Section 7 in DTS03004 describes in detail the neccessary steps to create a signature.

6. The originating gateway sends the token it received from the settlement server in the setup message to the terminating gateway.
7. The terminating gateway accepts the call after validating the token, and completes the call setup.

At the end of the call, both the originating and terminating gateways send usage indicator reports (CDRs) to the OSP server. The usage indicator reports contain the call detail information that the OSP server uses to provide settlement service between the originating and terminating service providers.

Figure 7 illustrates a call flow for a typical call settled by a clearinghouse server.

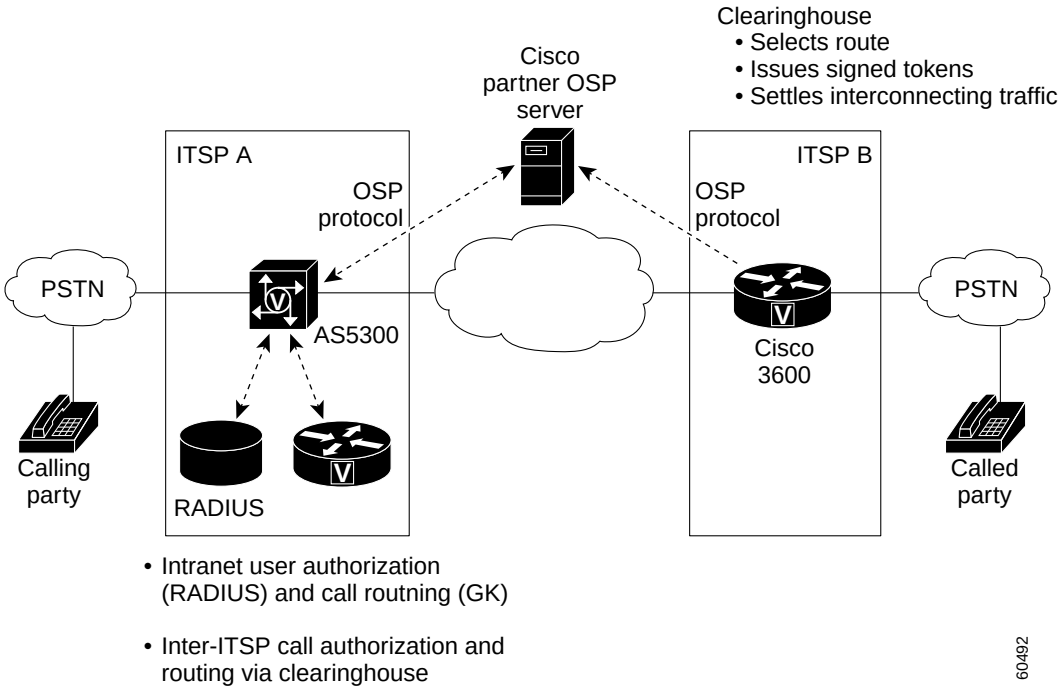
Figure 7 Typical Call Flow for OSP Clearinghouse Application



OSP Architecture

Figure 8 shows an OSP architecture model where all inter-ISP calls are routed and settled by an OSP server. Intra-ISP calls are routed and billed in the typical postpaid or prepaid model.

Figure 8 OSP Architecture Model



OSP Hardware and Software Requirements

The following sections describe the hardware and software requirements to support OSP:

- System Platform Requirements
- Memory and Software Requirements
- Hardware Components

System Platform Requirements

OSP is supported on the following Cisco platforms:

- Cisco AS5300 and AS5800 universal access servers; Cisco AS5400 universal gateway
- Cisco 2600 series routers
- Cisco 3620, 3640, and 3660 routers

Memory and Software Requirements

An OSP server must meet the following memory and software requirements:

- 16 MB Flash
- 128 MB DRAM
- TCL IVR script for settlements (session application embedded in Cisco IOS software)
- Cisco IOS Release 12.2(1) or later releases with -ik8s- or -jk8s- cryptographic images
- VCWare, DSPWare versions as listed in the compatibility matrix located at the following URL:

http://www.cisco.com/univercd/cc/td/doc/product/access/acs_serv/5300/iosrn/vcwrn/vcwrmtrx.htm

Hardware Components

To implement OSP clearinghouse services, you must deploy the following devices:

- VoIP gateway that supports 56-bit encryption, SSL, and OSP
- Certificate authority (CA)
- OSP server

The following are OSP server vendors:

- Concert: http://www.concert.com/
- GRIC: http://www.gric.com/
- Netrue: http://www.netrue.com
- OpenOSP: http://www.openosp.org/
- TransNexus: http://www.transnexus.com

Note The settlement feature cannot be enabled on dial peers that use RAS as the session target. The settlement software is offered only in cryptographic images; therefore, the images are under export controls.

OSP for Clearinghouse Services Application

The following sections describe the application of OSP clearinghouse services:

- OSP Configuration Guidelines
- OSP Troubleshooting Tips
- OSP Configuration Examples

OSP Configuration Guidelines

Configuring a Cisco router or access server to register with an online TransNexus Phase 1 OSP server consists of the tasks described in the following sections:

- Defining Gateway Identity Parameters
- Using Network Time Protocol
- Configuring the PKI
- Enrolling with the OSP Server

- Configuring Settlement Parameters
- Configuring Incoming and Outgoing Dial Peers

The specific configuration parameters may be different from those shown, depending on the OSP server with which you are registering.

Defining Gateway Identity Parameters

OSP servers typically locate gateways and clients using DNS. For this reason, you need to make sure that the voice gateway can locate the OSP server and other IP resources using DNS. The following example shows how to configure the domain name in which the gateway resides and identify a DNS server in that domain:

```
ip domain-name cisco.com
ip name-server 172.22.30.32
```

In this example, both the domain name and DNS are identified. Typically, the **ip domain-lookup** global configuration command is enabled by default in Cisco routers and gateways.

When you have configured the **ip domain-name** and the **ip name-server** commands and confirmed that the **ip domain-lookup** command is enabled, the gateway can refer to the OSP server by its domain name.

Note You should ensure that the **ip domain-lookup** global configuration command has not been overridden in the gateway. If the command is *not* listed in the **show running-config** privileged EXEC command output, the command is active.

Using Network Time Protocol

The OSP server requires accurate time stamps from gateways in order to rate calls adequately. Unless the gateway time-of-day clock (not to be confused with the system controller clock) is within a certain tolerance range (minutes) of the OSP server clock, token validation will fail. Gateways, servers, and other IP devices can synchronize their time-of-day clocks with each other through the NTP. As with DNS, a hierarchy of NTP servers is available on the Internet. Any gateway with access to the Internet can point to an available NTP server for accurate time-of-day synchronization. A list of stratum-2 NTP servers is at the following URL:

http://www.eecis.udel.edu/~mills/ntp/clock2.htm

To configure a Cisco gateway to synchronize its time to an authoritative NTP source, use the **ntp server** *ip-address* global configuration command. Although not required for simple time synchronization, the **ntp** global configuration command can be used to configure a variety of other NTP parameters, as shown in Table 9.

Table 9 NTP Global Configuration Command Keyword Options

Command	Purpose
ntp access-group	Controls NTP access.
ntp authenticate	Authenticates time sources.
ntp authentication-key	Configures the authentication key for trusted time sources.
ntp broadcastdelay	Displays the estimated round-trip delay.
ntp clock-period	Length of hardware clock tick.

Table 9 NTP Global Configuration Command Keyword Options (continued)

Command	Purpose
ntp master	Acts as NTP master clock.
ntp max-associations	Sets the maximum number of associations.
ntp peer	Configures an NTP peer.
ntp server	Configures an NTP server.
ntp source	Configures an interface for a source address.
ntp trusted-key	Configures key numbers for trusted time sources.
ntp update-calendar	Periodically updates the router calendar with NTP time.

NTP time formats are displayed in the following format, which is described in Table 10:

```
%H:%M:%S.%k %Z %tw %tn %td %Y
```

Table 10 NTP Record Field Descriptions

Value	Description (Range)
%H	Hour (00 to 23)
%M	Minutes (00 to 59)
%S	Seconds (00 to 59)
%k	Milliseconds (000 to 999)
%Z	Time zone string
%tw	Day of the week (Saturday to Sunday)
%tn	Month (January to December)
%td	Day of the month (01 to 31)
%Y	Year including century (for example, 1998)

Enabling the **debug ntp adjustments** command is a quick way to learn if your gateway is communicating with the configured NTP server. An example follows:

```
Router# debug ntp adjustments

NTP clock adjustments debugging is on
Router#
00:27:12: NTP: adj(-0.000000317), rem. offset = 0.000000000, adj = -0.000000317
00:27:13: NTP: adj(-0.000000838), rem. offset = 0.000000000, adj = -0.000000838
00:27:14: NTP: adj(-0.000000842), rem. offset = 0.000000000, adj = -0.000000842
00:27:15: NTP: adj(-0.000000933), rem. offset = 0.000000000, adj = -0.000000933
00:27:16: NTP: adj(-0.000001029), rem. offset = 0.000000000, adj = -0.000001029
00:27:17: NTP: adj(-0.000000290), rem. offset = 0.000000000, adj = -0.000000290
```

You can use the **show clock** privileged EXEC command to confirm the correct time.

Configuring the PKI

To configure the Public Key Infrastructure (PKI) for secured communication between the gateway and the OSP server, perform the tasks described in the following sections:

- Generating an RSA Key Pair
- Configuring the Enrollment Parameters
- Obtaining the CA Certificate

Generating an RSA Key Pair

The following example shows you how to generate an RSA key pair. When you enter the following command, you will receive the following feedback from the gateway:

```
Router(config)# crypto key generate rsa

The name for the keys will be: Group10_B.cisco.com
Choose the size of the key modulus in the range of 360 to 2048 for your General Purpose
Keys. Choosing a key modulus greater than 512 may take a few minutes.

How many bits in the modulus [512]:
```

Pressing **Enter** at this prompt will generate a 512-bit key. The system confirms that the RSA keys have been generated by displaying the following output:

```
Generating RSA keys ...

[OK]
```

Configuring the Enrollment Parameters

The following example shows how to configure the enrollment parameters:

```
Router(config)# crypto ca identity transnexus
Router(ca-identity)# enrollment url http://enroll.transnexus.com:2378
Router(ca-identity)# enrollment retry count 3
Router(ca-identity)# enrollment retry period 1
Router(ca-identity)# no enrollment mode ra
```

In this example, the **crypto ca identity** command places the router in CA identity configuration mode and matches the OSP parameters to the identity tag **transnexus**. The next three commands configure the enrollment URL of the OSP server and retry parameters. The fifth command makes sure you are not in resource authority configuration mode.

Obtaining the CA Certificate

The following example shows you how to obtain a CA certificate:

```
Router(config)# crypto ca authenticate ospserver
```

After you enter this command, you will receive the following feedback from the gateway:

```
Certificate has the following attributes:
Fingerprint: 96D254B4 0AEF4F23 7A545BF9 70DC4D17
% Do you accept this certificate? [yes/no]: Y <To accept this certificate, you type "Y"
here.>
```

The keyword must be the same as the one used when declaring the CA with the **crypto ca identity** command (in this example, **ospserver**).

Enrolling with the OSP Server

The following example shows you how to enroll your voice gateway with the OSP server:

```
Router(config)# crypto ca enroll ospserver
```

After you enter this command, the system will prompt you for a series of responses. All of the responses that you should enter are in bold type in the following output.

```
%
% Start certificate enrollment ..
% Create a challenge password. You will need to verbally provide this
password to the CA Administrator in order to revoke your certificate.
For security reasons your password will not be saved in the configuration.
Please make a note of it.
Password: xxxx <You enter your password here.>
Re-enter password: xxxx <You re-enter your password here.>
% The subject name in the certificate will be: Group10_B.cisco.com
% Include the router serial number in the subject name? [yes/no]: Y
% The serial number in the certificate will be: 006CE956
% Include an IP address in the subject name? [yes/no]: Y
Interface: Ethernet 0
Request certificate from CA? [yes/no]: Y
% Certificate request sent to Certificate Authority
% The certificate request fingerprint will be displayed.
% The 'show crypto ca certificate' command will also show the fingerprint
Wait here for the feedback below:
(config)# Fingerprint: 24D05F87 1DE1D0C9 4DF974D1 7AE064C6
11:15:12: %CRYPTO-6-CERTRET: from Certificate Authority
```

If you do not get the CERTRET feedback, your enrollment most likely has failed. If you continue without a proper certificate, you will not be able to register. After you receive a certificate, verify the presence of the certificate by displaying your running configuration using the **show running-config** privileged EXEC command.

```
Router# show running-config
Current configuration:
!
version 12.0
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname ogw
!
enable secret level 14 5 $1$gB9t$cNIAO.XGbV/2ebLeiYXPc.
enable secret 5 $1$W1Wx$KtFgICn0Q7X8BbxwFnR991
!
clock timezone GMT 0
ip subnet-zero
ip domain-name cisco.com
!
crypto ca identity transnexus
enrollment url http://10.100.1.3
crypto ca certificate chain transnexus
certificate AFADA65D4DF416847B6B284AB197146E
30820231 3082019A A0030201 02021100 AFADA65D 4DF41684 7B6B284A B197146E
300D0609 2A864886 F70D0101 04050030 6E310B30 09060355 04061302 55533110
300E0603 55040813 0747656F 72676961 31183016 06035504 0A130F54 72616E73
4E657875 732C204C 4C433114 30120603 55040B13 0B446576 656C6F70 6D656E74
311D301B 06035504 03131454 52414E53 4E455855 53204245 54412043 41203130
1E170D39 39303332 39323235 3235395A 170D3030 30333239 32323532 35395A30
72317030 0D060355 04051306 36434232 31413017 06092A86 4886F70D 01090813
```

```
0A31302E 3130302E 312E3130 1A06092A 864886F7 0D010902 160D6F67 772E6369
73636F2E 636F6D30 2A060355 04031423 5B747261 6E736E65 7875732E 636F6D20
47574944 3D313030 20435349 443D3430 30305D30 5C300D06 092A8648 86F70D01.
01010500 034B0030 48024100 C871D5F7 8529C9AE 9E7BC554 C5510B75 A66C9E78
405FECDB 60896552 80106C8F 7F7F9B3B 89A50D55 0578881D 3672CCFE 9BB5E515
47D03E95 CE4CC0F1 3DC20593 02030100 01A30F30 0D300B06 03551D0F 04040302
05A0300D 06092A86 4886F70D 01010405 00038181 00256D3C 087E8005 74D05759
0B9924B2 842675D5 C37A913C A2E16AC1 B146161C DFF7F96A 0053DCFC F5E1E22D
E51D4C82 9A97D2E8 B38E5CE0 902CEFE1 13181486 5929DF21 B882775E 830563A2
D15C61DE 0EFDC39D 334ECD0D E826E953 1C37ED56 2DA5D765 5B9949E6 1D33E3CE
FB3E2818 78355CDF 4A9A6118 52B6FF48 D07A6DEB 33
quit
certificate ca 0171
3082024C 308201B5 02020171 300D0609 2A864886 F70D0101 04050030 6E310B30
09060355 04061302 55533110 300E0603 55040813 0747656F 72676961 31183016
06035504 0A130F54 72616E73 4E657875 732C204C 4C433114 30120603 55040B13
0B446576 656C6F70 6D656E74 311D301B 06035504 03131454 52414E53 4E455855
53204245 54412043 41203130 1E170D39 39303332 32313334 3630395A 170D3030
30333231 31333436 30395A30 6E310B30 09060355 04061302 55533110 300E0603
55040813 0747656F 72676961 31183016 06035504 0A130F54 72616E73 4E657875
732C204C 4C433114 30120603 55040B13 0B446576 656C6F70 6D656E74 311D301B
06035504 03131454 52414E53 4E455855 53204245 54412043 41203130 819F300D
06092A86 4886F70D 01010105 0003818D 00308189 02818100 B1B8ACFC D78F0C95
0258D164 5B6BD8A4 6F5668BD 50E7524B 2339B670 DC306537 3E1E9381 DE2619B4
4698CD82 739CB251 91AF90A5 52736137 658DF200 FAFEFE6B 7FC7161D 89617E5E
4584D67F F018EDAB 2858DDF9 5272F108 AB791A70 580F994B 4CA54F08 38C32DF5
B44077E8 79830F95 96F1DA69 4CAE16F2 2879E07B 164F5F6D 02030100 01300D06
092A8648 86F70D01 01040500 03818100 2FDCB580 C29E557C 52201151 A8DB5F47
C06962D5 8FDA524E A69DE3EE C3FE166A D05C8B93 2844CD66 824A8859 974F22E0
46F69F7E 8027064F C19D28BC CA750E4E FF2DD68E 1AA9CA41 8BB89C68 7A61E9BF
49CBE41E E3A42B16 AAEDAEC7 D3B4F676 4F1A817B A5B89ED8 F03A15B0 39A6EBB9
0AFA6968 17A9D381 FD62BBB7 A7D379E5
quit
.
.
.
ntp clock-period 17182503
ntp server 10.100.1.3
end
```

You should find two large blocks of hex strings similar to the ones shown. One block is a representation of the OSP server certificate and the other represents the key from the certificate authority. The presence of both keys is an indication (although not a guarantee) that the registration has occurred correctly.

Configuring Settlement Parameters

The following example shows you how to configure settlement parameters on your gateway:

```
Router(config)# settlement 0
Router(config-settlement)# type osp
Router(config-settlement)# url https://192.168.152.17:8444/
Router(config-settlement)# response-timeout 20
Router(config-settlement)# device-id 1039928734
Router(config-settlement)# customer-id 805311438
Router(config-settlement)# no shutdown
```

The first command places the router in settlement configuration mode. The next five commands configure the settlement parameters, including the settlement URL. Notice that settlement is using the SSL protocol (denoted by https://) as the transport mechanism. The device ID and customer ID are both Transnexus-specific.

Configuring Incoming and Outgoing Dial Peers

The incoming POTS dial peer on the originating gateway is associated with the session application that initiates the OSP activities. The following example shows how to configure the incoming POTS dial peer on the originating gateway:

```
Router(config)# dial-peer voice 1 pots
Router(config-dial-peer)# application session
Router(config-dial-peer)# incoming called-number 1415.......
Router(config-dial-peer)# port 1/0/0
```

The first command opens the dial-peer configuration mode and defines the tag number of the dial peer you are configuring. The **application session** command associates the session application with the call. The last two commands configure general POTS dial-peer parameters.

The outbound VoIP dial peer on the originating gateway has a session target of **settlement**, which directs the call to the OSP server. The following example shows how to configure the outbound VoIP peer on the originating gateway:

```
Router(config)# dial-peer voice 10 voip
Router(config-dial-peer)# destination-pattern 1219.......
Router(config-dial-peer)# session target settlement
```

The first command opens dial-peer configuration mode and defines the tag number of the dial peer you are configuring. The **session target settlement** command sends the call to the OSP server. The other commands configure general VoIP dial peer parameters.

The VoIP dial peer on the terminating gateway matches the outgoing VoIP dial peer on the originating gateway and must also point to the **settlement** session target. The following example shows how to configure the inbound VoIP and outbound POTS dial peers on the terminating gateway:

```
Router(config)# dial-peer voice 10 voip
Router(config-dial-peer)# application session
Router(config-dial-peer)# incoming called-number 1415.......
Router(config-dial-peer)# session target settlement:0
Router(config-dial-peer)# exit
Router(config)# dial-peer voice 1 pots
Router(config-dial-peer)# incoming called-number 1219.......
Router(config-dial-peer)# port 1/0/0
```

The first command opens the dial-peer configuration mode and defines the tag number of the inbound VoIP dial peer you are configuring. The **application session** command associates the session application with the call. The **session target settlement** command identifies the settlement server.

The **dial-peer voice 1 pots** command defines the outbound POTS dial peer and defines its tag number. The last two commands configure general POTS dial peer parameters.

OSP Troubleshooting Tips

In general, the following commands are useful in debugging an OSP installation:

- **debug voip ivr settlement**—Displays IVR settlement information.
- **debug voip settlement network**—Shows the messages exchanged between a router and a settlement provider.
- **debug voip settlement errors—**Displays all settlement errors.
- **debug voip settlement transaction**—Displays the attributes of the transactions on the settlement gateway.

- **debug voip settlement misc—**Shows the details on the code flow of each settlement transaction.

Common Problems with Settlement Configuration

The following sections describe common problems with OSP configurations.

Settlement Database Not Set Up Properly

Problem: Calls are routed through a settlement server, but the originating gateway gets no response, or negative response.

Solution: Check with the settlement provider to make sure the router is properly registered with that provider. Router registration with the settlement provider is normally done outside of OSP.

TCL IVR Script Not Called

Problem: TCL IVR script is not used on the originating gateway or terminating gateway.

Solution: You can configure a TCL IVR script for the dial peer using the **application name** dial peer configuration command or you can use the **show call application voice summary** command to list all the available scripts on the router:

```
router# show call application voice summary

name                 description
session              Basic app to do DID, or supply dialtone.
fax_hop_on           Script to talk to a fax redialer
clid_authen          Authenticate with (ani, dnis)
clid_authen_collect  Authenticate with (ani, dnis), collect if that fails
clid_authen_npw      Authenticate with (ani, NULL)
clid_authen_col_npw  Authenticate with (ani, NULL), collect if that fails
clid_col_npw_3       Authenticate with (ani, NULL), and 3 tries collecting
clid_col_npw_npw     Authenticate with (ani, NULL) and 3 tries without pw
SESSION              Default system session application
```

No Destination Pattern Set

Problem: The inbound POTS dial peer on the originating gateway has no destination pattern set.

Solution: Because some PBX devices do not pass along the calling number in the setup message, the router uses the destination pattern number or answer-address as an alternative, and calling number is a required field for settlement.

No Session Target Settlement Set on Originating Gateway

Problem: The outbound VoIP dial peer of the originating gateway does not have the session target configured for **settlement**.

Note The router can make successful calls, but not through a settlement server. The session target attribute dictates how the router resolves the terminating gateway address for a particular called number.

Solution: Configure the **session target settlement** [*provider-number*] command.

No VoIP Inbound Dial Peer on Terminating Gateway

Problem: The terminating gateway has no VoIP inbound dial peer. Because the settlement token in the incoming setup message from the originating gateway cannot be validated, the terminating gateway rejects the call.

Solution: Create an inbound dial peer with the **session target settlement** [*provider-number*] command.

No Application Attribute on Terminating Gateway

Problem: The terminating gateway has an inbound dial peer configured, but with no application attribute. The default session application (SESSION) processes the call, but it does not support settlement.

Solution: Configure the application *application name* attribute in the inbound dial peer.

Terminating Gateway Not Synchronized with Settlement Server

Problem: The terminating gateway clock is not synchronized with the settlement server. The terminating gateway rejects the call because it is too soon or too late to use the settlement token in the incoming setup message.

Solution: Use the **ntp** or **clock set** command to synchronize the clocks between the terminating gateway and the settlement server.

Settlement Provider Not Running

Problem: The settlement provider on the originating gateway or terminating gateway is not running. No settlement transaction processing is allowed unless the provider is running.

Solution: Enable settlement using the **no shutdown** command in settlement configuration mode. Use the **show settlement** command to verify the provider status.

Router and Server Not Using SSL to Communicate

Problem: The router cannot use SSL to communicate with the server because the server URL should be “https” not “http.”

Solution: Configure a secured URL using “https.”

Problem: The router cannot use SSL to communicate with the server because the certificates of the server or router were not properly obtained.

Solution: Check the certificate enrollment process for both the server and the router.

Multiple Dial Peers Have Random Order

Problem: The originating gateway has multiple dial peers for the same called number, and settlement is never used. The order for rotary dial peers is random unless a dial peer preference is specified. The dial peer with lower preference is chosen first.

Solution: Define dial peer preference using the **preference** *number* command.

H.323 Setup Connection Timeout

Problem: The originating gateway cannot successfully set up a call with the first terminating gateway that is returned from the OSP server. The problem occurs when a gateway attempts to set up the call with the terminating gateways in the order they are received. If for some reason the H.323 call setup is not successful, there is a 15-second default timeout before the next terminating gateway on the list is contacted.

Solution: Tune the H.323 call setup timeout by using the **h225 timeout** command:

```
voice class h323 1
 h225 timeout tcp establish <value 0 to 30 seconds>

dial-peer voice 919 voip
 application session
 destination-pattern 919555....
 voice-class codec 1
 voice-class h323 1
 session target settlement
```

OSP Problem Isolation

If you are having trouble isolating the problems that are occurring with settlement, perform the following tasks:

- Check the originating and terminating gateway configurations for dial peers, settlement providers, and certificates.
- Check the network between the originating gateway, terminating gateway, and the server. Ping each device to make sure that the machines are running.
- Verify that IP calls can be made successfully. If so, the problem is specific to settlement.
- Use the **debug voip ivr settlement** command on the originating gateway to learn if the TCL IVR script initiates a settlement request to the server.
- Use the **debug voip settlement network** command on the originating gateway to capture the HTTP requests sent to the server and the response from the server. If the originating gateway gets no response from the server, contact the settlement provider.
- Use the **debug voip settlement misc** command to display the list of TOWs returned from the server. If this list is incorrect, contact the settlement provider.
- If the terminating gateway rejects the settlement token because it is too soon or too late to use it, synchronize the terminating gateway clock with the server.

OSP Configuration Examples

The following sections show configuration examples for OSP gateways:

- Configuring OSP on the Originating Gateway
- Configuring OSP on the Terminating Gateway

Configuring OSP on the Originating Gateway

The following example shows an originating gateway configured to register with an OSP server:

Note The first tuplet in each IP address in this example has been replaced with a unique variable.

```
version 12.1
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Group2_A
!
boot system flash c3640-js56i-mz_120-4_XH.bin
enable password pme123
!
clock timezone PST -7
ip subnet-zero
ip domain-name cisco.com
ip name-server xxx.156.128.1
ip name-server xxx.156.128.10
!
cns event-service server
!
crypto ca identity transnexus
! Certificate authority identity parameters
 enrollment retry count 3
 enrollment url http://enroll.transnexus.com:2378
! Clearinghouse server address
crypto ca certificate chain transnexus
! The following 2 blocks of characters are a hexadecemal representation of the
! certificates present on the gateway.
certificate 73A39A2746B2BFFC373AF35B70F427CC
 30820246 308201AF A0030201 02021073 A39A2746 B2BFFC37 3AF35B70 F427CC30
 0D06092A 864886F7 0D010104 0500306E 310B3009 06035504 06130255 53311030
 0E060355 04081307 47656F72 67696131 18301606 0355040A 130F5472 616E734E
 65787573 2C204C4C 43311430 12060355 040B130B 44657665 6C6F706D 656E7431
 1D301B06 03550403 13145452 414E534E 45585553 20424554 41204341 2031301E
 170D3939 31303132 31353430 33345A17 0D303031 30313231 35343033 345A3081
 87318184 300D0603 55040513 06413137 30443030 1A06092A 864886F7 0D010908
 130D3230 392E3234 2E313431 2E333430 1F06092A 864886F7 0D010902 16124772
 6F757032 5F412E63 6973636F 2E636F6D 30360603 55040314 2F5B7472 616E736E
 65787573 2E636F6D 20475749 443D3130 37333734 36393333 20435349 443D3830
 35333131 3433385D 305C300D 06092A86 4886F70D 01010105 00034B00 30480241
 00E288FF 7C275A55 5C375387 99FB9682 7BFC554C F2DFA453 BFFD88AB 657C0FD5
 7FC510BA 13DDEB99 DF7E5FAA 5BE5952E B974F8DB 1B333F2C D4C5689D 61812121
 DB020301 0001A30F 300D300B 0603551D 0F040403 0205A030 0D06092A 864886F7
 0D010104 05000381 81007D83 08924EFD F2139D01 504FAC21 35108FCF 083D9DA7
 495649F6 6D1E28A6 1A687F1C CAF5BDBD 37E8E8A1 54401F4A 73BBFB05 786E01BC
 AF966529 AC92648B 2A4B9FEC 3BFFEBF8 81A116B5 4D3DAA93 7E4C24FB E3624EB3
 D630C232 D016149D 427557A1 F58F313E F92F9E9D ADBA3873 92EBF7F0 861E0413
 F81CD5C0 E4E18A03 2FA2
quit
certificate ca 0171
 3082024C 308201B5 02020171 300D0609 2A864886 F70D0101 04050030 6E310B30
 09060355 04061302 55533110 300E0603 55040813 0747656F 72676961 31183016
 06035504 0A130F54 72616E73 4E657875 732C204C 4C433114 30120603 55040B13
 0B446576 656C6F70 6D656E74 311D301B 06035504 03131454 52414E53 4E455855
 53204245 54412043 41203130 1E170D39 39303332 32313334 3630395A 170D3030
 30333231 31333436 30395A30 6E310B30 09060355 04061302 55533110 300E0603
 55040813 0747656F 72676961 31183016 06035504 0A130F54 72616E73 4E657875
```

```
 732C204C 4C433114 30120603 55040B13 0B446576 656C6F70 6D656E74 311D301B
 06035504 03131454 52414E53 4E455855 53204245 54412043 41203130 819F300D
 06092A86 4886F70D 01010105 0003818D 00308189 02818100 B1B8ACFC D78F0C95
 0258D164 5B6BD8A4 6F5668BD 50E7524B 2339B670 DC306537 3E1E9381 DE2619B4
 4698CD82 739CB251 91AF90A5 52736137 658DF200 FAFEFE6B 7FC7161D 89617E5E
 4584D67F F018EDAB 2858DDF9 5272F108 AB791A70 580F994B 4CA54F08 38C32DF5
 B44077E8 79830F95 96F1DA69 4CAE16F2 2879E07B 164F5F6D 02030100 01300D06
 092A8648 86F70D01 01040500 03818100 2FDCB580 C29E557C 52201151 A8DB5F47
 C06962D5 8FDA524E A69DE3EE C3FE166A D05C8B93 2844CD66 824A8859 974F22E0
 46F69F7E 8027064F C19D28BC CA750E4E FF2DD68E 1AA9CA41 8BB89C68 7A61E9BF
 49CBE41E E3A42B16 AAEDAEC7 D3B4F676 4F1A817B A5B89ED8 F03A15B0 39A6EBB9
 0AFA6968 17A9D381 FD62BBB7 A7D379E5
quit
!
voice-port 1/0/0
!
voice-port 1/0/1
!
voice-port 1/1/0
!
voice-port 1/1/1
!
dial-peer voice 1 pots
! The incoming pots dial peer on the originating gateway is associated
! with the session application that initiates the OSP activities.
 application session
 destination-pattern 9549204
 port 1/0/0
!
dial-peer voice 10 voip
 destination-pattern 7671234
 session target ipv4:10.24.141.35
!
dial-peer voice 101 voip
! The outgoing VoIP dial peer has a session target of settlement, which directs the call
! to the OSP server.
 application session
 destination-pattern 1T
 session target settlement
!
process-max-time 200
settlement 0
 type osp
! The settlement parameters include the URL to the settlement server; in this case,
! using SSL
 url https://10.144.152.17:8444/
 device-id 1073746933
 customer-id 805311438
 no shutdown
!
interface Ethernet0/0
 no ip address
 no ip directed-broadcast
 shutdown
!
interface Ethernet0/1
 description flat management network
 ip address 10.24.141.34 255.255.255.240
 no ip directed-broadcast
!
ip classless
ip route 0.0.0.0 0.0.0.0 10.24.141.33
no ip http server
!
```

```
line con 0
 exec-timeout 0 0
 transport input none
line aux 0
 speed 115200
line vty 0 4
 exec-timeout 0 0
 password pme123
 no login
!
ntp clock-period 17180168
ntp source Ethernet0/1
! NTP parameters are pointing to a Stratum 2 NTP server
ntp server 209.24.141.33
end
```

Configuring OSP on the Terminating Gateway

The following example shows a terminating gateway configured to support OSP.

Note The first tuplet in each IP address in this example has been replaced with a unique variable.

```
version 12.1
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Group2_B
!
enable password pme123
!
clock timezone PST -7
ip subnet-zero
ip domain-name cisco.com
ip name-server xxx.156.128.1
ip name-server xxx.156.128.10
!
cns event-service server
!
crypto ca identity transnexus
! Certificate authority identity parameters
 enrollment retry count 3
 enrollment url http://enroll.transnexus.com:2378
crypto ca certificate chain transnexus
certificate 0172
 30820264 308201CD 02020172 300D0609 2A864886 F70D0101 04050030 6E310B30
 09060355 04061302 55533110 300E0603 55040813 0747656F 72676961 31183016
 06035504 0A130F54 72616E73 4E657875 732C204C 4C433114 30120603 55040B13
 0B446576 656C6F70 6D656E74 311D301B 06035504 03131454 52414E53 4E455855
 53204245 54412043 41203130 1E170D39 39303332 32313334 3631345A 170D3030
 30333231 31333436 31345A30 8185310B 30090603 55040613 02555331 10300E06
 03550408 13074765 6F726769 61311830 16060355 040A130F 5472616E 734E6578
 75732C20 4C4C4331 31302F06 0355040B 13285472 616E736E 65787573 20536574
 746C656D 656E7420 53657276 65722044 6576656C 6F706D65 6E743117 30150603
 55040313 0E747261 6E736E65 7875732E 636F6D30 819F300D 06092A86 4886F70D
 01010105 0003818D 00308189 02818100 AF4E4E7A 7AE56E12 8526027B 4FAA7E16
 07710217 72EF63B9 8C0CAD75 C40724FE 71779746 937C8499 0EE9B19E FE7E76D0
 12A9FD09 DA7FE092 979FA5C6 066F6FAB 3614229A A352708E 87BE67A0 B7D1B8F1
 2238DCD7 E1D5D538 E632974E 2B15A124 E72BEBCA 054A7000 43090FF6 A62E05DD
 86452268 12EA8BF9 D7E63996 116426D5 02030100 01300D06 092A8648 86F70D01
```

```
 01040500 03818100 7DDBBA3F 2EF28952 6458090A E005C659 F26D690C 3CEB89A3
 B4C4BF49 8CA7B624 EF75AA02 3C723BCD 028C04FF 191EE516 49AE9092 CADED3F9
 D652EE75 E0BCF22E EBA6908F BD7D8248 F19F3BCE D06B0A26 5FADFA19 1C5E9721
 6BCD8EFA 249DD629 5024EA19 5B2B0732 CE5DF1DD 7758EB41 B3F3FE1C D0E34AAA
 5E3CA3D2 9FEA6CA2
quit
certificate ca 0171
 3082024C 308201B5 02020171 300D0609 2A864886 F70D0101 04050030 6E310B30
 09060355 04061302 55533110 300E0603 55040813 0747656F 72676961 31183016
 06035504 0A130F54 72616E73 4E657875 732C204C 4C433114 30120603 55040B13
 0B446576 656C6F70 6D656E74 311D301B 06035504 03131454 52414E53 4E455855
 53204245 54412043 41203130 1E170D39 39303332 32313334 3630395A 170D3030
 30333231 31333436 30395A30 6E310B30 09060355 04061302 55533110 300E0603
 55040813 0747656F 72676961 31183016 06035504 0A130F54 72616E73 4E657875
 732C204C 4C433114 30120603 55040B13 0B446576 656C6F70 6D656E74 311D301B
 06035504 03131454 52414E53 4E455855 53204245 54412043 41203130 819F300D
 06092A86 4886F70D 01010105 0003818D 00308189 02818100 B1B8ACFC D78F0C95
 0258D164 5B6BD8A4 6F5668BD 50E7524B 2339B670 DC306537 3E1E9381 DE2619B4
 4698CD82 739CB251 91AF90A5 52736137 658DF200 FAFEFE6B 7FC7161D 89617E5E
 4584D67F F018EDAB 2858DDF9 5272F108 AB791A70 580F994B 4CA54F08 38C32DF5
 B44077E8 79830F95 96F1DA69 4CAE16F2 2879E07B 164F5F6D 02030100 01300D06
 092A8648 86F70D01 01040500 03818100 2FDCB580 C29E557C 52201151 A8DB5F47
 C06962D5 8FDA524E A69DE3EE C3FE166A D05C8B93 2844CD66 824A8859 974F22E0
 46F69F7E 8027064F C19D28BC CA750E4E FF2DD68E 1AA9CA41 8BB89C68 7A61E9BF
 49CBE41E E3A42B16 AAEDAEC7 D3B4F676 4F1A817B A5B89ED8 F03A15B0 39A6EBB9
 0AFA6968 17A9D381 FD62BBB7 A7D379E5
quit
!
voice-port 1/0/0
 description ma bell 954 9173
!
voice-port 3/0/0
 input gain 14
!
voice-port 3/0/1
!
voice-port 3/1/0
!
voice-port 3/1/1
!
!
dial-peer voice 1 pots
 application clid_authen_collect
 incoming called-number 9549172
 port 3/0/0
!
dial-peer voice 767 pots
 destination-pattern 7......
 port 3/0/0
 prefix 7
!
dial-peer voice 513 pots
! The outgoing pots dial peer is associated with the default application and
! does not need an OSP application association.
 destination-pattern 1513.......
 port 3/0/0
!
dial-peer voice 1513 voip
! The incoming VoIP dial peer, which matches the outgoing VoIP dial peer on
! the originating gateway, must also point to a session target of settlement.
 application session
 incoming called-number 1513.......
 session target settlement
!
```

```
dial-peer terminator #
process-max-time 200
settlement 0
 type osp
 url https://10.144.152.17:8444/
 device-id 1140855798
 customer-id 805311438
 no shutdown
!
interface Ethernet0/0
 no ip address
 no ip directed-broadcast
 shutdown
!
interface Serial0/0
 no ip address
 no ip directed-broadcast
 no ip mroute-cache
 shutdown
!
interface Ethernet0/1
 description Transnexus enrollment
 ip address 10.24.141.35 255.255.255.240
 no ip directed-broadcast
!
ip classless
ip route 0.0.0.0 0.0.0.0 10.24.141.33
ni ip http server
!
line con 0
 exec-timeout 0 0
 transport input none
line aux 0
line vty 0 4
 no login
!
ntp clock-period 17180148
ntp source Ethernet0/1
ntp server 10.24.141.33
end
```