martini
security

Martini Security
Deploying STIR/SHAKEN with Asterisk

Version 1.0

Deploying STIR/SHAKEN with Asterisk

Background

This document will describe our recommended strategy for deploying STIR/SHAKEN with Asterisk along with services offered by Martini Security.

Asterisk, as of version 18, natively supports STIR/SHAKEN via module res_stir_shaken. It does so with configuration added to the PJSIP endpoints in addition to configuration loaded from stir_shaken.conf. We will not be discussing this further; however, since the feature set is poorly documented / supported (author's opinion).

The preferred option is to implement Vermouth. This application runs as a separate service alongside Asterisk and offers a REST API for the signing and verification of SIP calls from within Asterisk dialplan configuration. It also implements its own lightweight ACME client for automatic cert renewal.

Sample Configuration Notes

All information provided was tested against Ubuntu Server 22 LTS with latest packages as of 2023-04-04. It is assumed (without guarantee) that other flavors of Debian or RHEL/Fedora could be supported as well.

Asterisk 18.10.0 was installed via `apt install asterisk`. Configuration was also tested against Asterisk 20.2.1 compiled from source.

Source code was cloned using:

```
git clone -b 20.2.1 https://github.com/asterisk/asterisk.git
```

It is assumed that the reader has adequate familiarity with Asterisk along with a solid understanding of SIP. Knowledge of the ATIS STIR/SHAKEN specifications is also helpful.

Finally, this document utilizes pjsip rather than chan_sip as chan_sip has been deprecated and will eventually be removed. The good news is that chan_sip still works just fine with vermouth, and much of the config can be adapted for legacy asterisk installs.

Getting Started

You will first need to create an account with Martini Security and get an API key. This process consists of four steps:

1. Enroll for an account with Martini Security and obtain an API key.
2. Provide your FCC 499 filer ID.
3. Choose a subscription plan and pay.
4. Have the registered 499 filer approve the request to represent their organization.

The process of completing the account creation normally takes just a few minutes.

Once enrolled, creating ACME credentials can be done in just a few steps. 1. Click ACME Clients on the left navigation bar. 2. Click to + to add a new client. 3. Give it a label meaningful to you like "signing-node-1" 4. Locate the sections labeled "Key ID" and "API Key" and copy these somewhere. You will need them soon regardless of which mechanism you will use to implement STIR/SHAKEN.

For more information, you can watch this video: https://youtu.be/CXvR-jyJVx4

You will also need your STI-PA API credentials and OCN. These are credentials you created at iConectiv's STIR/SHAKEN provider portal.

General Asterisk + STIR/SHAKEN Guidance

All STIR/SHAKEN implementations need information relating to:

- X5U
  - Public STIR/SHAKEN certificate HTTP URL
- Attestation Level
  - A, B, or C as defined by ATIS
- OrigTN
  - Calling party
- DestTN
  - Called party
- OrigID
  - Call identifier specified by signer (unspecified OrigIDs cause most signing engines to generate and use a UUID)
- Private Key Path
  - Signing key used to sign calls

Asterisk has a variety of ways of accessing variables relating to the calling and called parties. Built-in variables such as `${CALLERID(num)}`, `${CALLERID(ani)}`, and `${CALLERID(dnid)}` as well as accessing pjsip channel variables are used throughout for OrigTN and DestTN purposes.

Configuration and testing were performed with To/From and Caller ID information specific to our lab setup. Note that where the OrigTN and DestTN are derived from is highly specific to your VoIP environment. It is important to note that the OrigTN and DestTN are always formatted as E.164 without "+" (1NXXNXXXX).

The request to add the identity header should occur just before the INVITE is sent to an external peer. The request to check the identity should occur just as a SIP INVITE is received from an external SIP peer. These practices remain true across all STIR/SHAKEN deployments.

Vermouth

About

Vermouth is a system service that listens on a configured IP and port for signing and verification requests and responds appropriately. It performs a number of services including ACME operations for requesting and renewing signing certificates.

Installation

Vermouth can be installed from a package or built from source. After downloading the current release (or building the package by hand), install it with `dpkg -i vermouth-X.X.X.deb`. By default, /etc/vermouth is the YAML master configuration file which will need to be modified before vermouth can be started.

Vermouth Configuration

First, open /etc/vermouth in your preferred text editor. Most of the default fields already have the recommended values and can be left as-is. The following lines, however, must be modified:

```
sti-pa:
  api_user_id: <STI-PA API Username>
  api_password: <STI-PA API Password>
sti_as:
  carrier_ocn: <your OCN>
  acme_eab_key_id: <ACME Key ID from Martini Security>
  acme_eab_key_secret: <ACME Key from Martini Security>
```

After making the required changes, use `sytemctl start vermouth` to begin STIR/SHAKEN services.

Asterisk Configuration

Asterisk (by way of `extensions.conf`) will be configured to make synchronous HTTP requests to the vermouth API endpoints. The returned values can be used for further decisioning.

Your VoIP environment will determine which values are needed and how those values must be cleaned up before making requests to vermouth. For example in signing requests, you might want to use the outbound P-Asserted-Id instead of the From. The reason is that you might have Anonymous in the From with the Privacy:id header set. Here, the P-Asserted-Id has the actual origTN to be used in a signing request.

extensions.conf

Our extensions.conf file contains just the configuration necessary to demonstrate call signing and verification.

```
; This context assigned to the internal SIP endpoint
[from-internal]
exten = _+1X.,1,NoOp(OB Call to ${EXTEN})
 ; pjsip only allows a header to be added to an outbound INVITE as a result
 ;  of a pre-DIAL subroutine — hence the b(sub-sign... section.
 same = n,Dial(PJSIP/${EXTEN}@pstn,,b(sub-sign-call,sign,1(${CALLERID(num)},${CALLERID
 ↪  (dnid)})))
 same = n,HangUp()

; Vermouth accepts 2 possible origTNs and 2 possible DestTns to match
;  against when determining whether the signature and call data are aligned
;  For this reason, we pull various variables and normalize them as 11
;  digits using a subroutine
; This context is assigned to the external PSTN SIP endpoint
[from-external]
exten = _+1X.,1,NoOp(About to perform STIR/SHAKEN Validation for call to ${EXTEN})
 same = n,Gosub(sub-clean-tn,clean,1({CALLERID(ani)}))
 same = n,Set(ANI=${GOSUB_RETVAL})
 same = n,Set(RURI=${CHANNEL(pjsip,request_uri)})
 same = n,Set(RURI=${CUT(RURI,@,1)})
 same = n,Set(RURI=${CUT(RURI,:,2)})
 same = n,Gosub(sub-clean-tn,clean,1(${RURI}))
 same = n,Set(RURI=${GOSUB_RETVAL})
 same = n,Gosub(sub-clean-tn,clean,1(${CALLERID(dnid)}))
 same = n,Set(DNID=${GOSUB_RETVAL})
 same = n,Gosub(sub-clean-tn,clean,1(${CALLERID(num)}))
 same = n,Set(CIDNUM=${GOSUB_RETVAL})
 same = n,Set(SIGNATURE=${PJSIP_HEADER(read,Identity)})
 ; Here we just print the status, but you could choose alternate handling
 ;  depending on the result.
 same = n,NoOp(STIR/SHAKEN Status: ${CURL(http://127.0.0.1:8085/ezstir/v1/verification
 ↪  /${ANI}/${CIDNUM}/${DNID}/${RURI}/${SIGNATURE})})
 same = n,Gosub(to-internal,${EXTEN},1)

; Always called from context 'from-external'
; In this example, all dialed numbers originating from the PSTN and beginning '+1X' are
```

```
;  sent to the internal endpoint
[to-internal]
exten = _+1X.,1,Dial(PJSIP/6001)
 same = n,HangUp()

; Vermouth' s signing engine will clean up most formats that present as a
;  valid US phone number. Still, the called and calling info could be
;  tidied up before the signing request is made.
[sub-sign-call]
exten = sign,1,Set(ORIGTN=${ARG1})
 same = n,Set(DESTTN=${ARG2})
 same = n,Set(SIGNATURE=${CURL(http://127.0.0.1:8085/ezstir/v1/signing/${ORIGTN}/${DES
 ↪  TTN})})
 ; If successful, the return value will also begin 'Identity'.
 ; PJSIP has no function to add a raw SIP header. We need to strip part of
 ;  the returned string and then add the Identity header through the
 ;  supported mechanism: 'Set(PJSIP_HEADER(add,hdr_name)=${VALUE})'
 same = n,ExecIf($["${SIGNATURE:0:8}"="Identity"]?Set(PJSIP_HEADER(add,Identity)=${SIG
 ↪  NATURE:10}))
 same = n,Return()

; Simple phone number clean-up
[sub-clean-tn]
exten = clean,1,ExecIf($[${LEN(${ARG1})} < 10 | ${LEN(${ARG1})} > 12]?Return())
 same = n,ExecIf($["${ARG1:0:2}"="+1"])?Return(${ARG1:1})
 same = n,ExecIf($["${ARG1:0:1}"="1"])?Return(${ARG1})
 same = n,Return(1${ARG1})
```