SITCH

Situational Information from Telemetry and Correlated Heuristics

Premise: Inexpensive Situational Information Gathering for GSM Networks

In the last few years it has become easier and far less expensive to intercept and record cell phone conversations. From compromising trusted devices like femtocells to building your own base station for around $500, for relatively little money just about anyone with some basic Linux experience can do some really evil things. As security practices and techniques typically go, defensive measures develop after a legitimate avenue of attack has been demonstrated. Can we easily and inexpensively detect the presence of Man-In-The-Middle (MITM)-capable devices?

Results of a Successful MITM Attack

Loss of Anonymity:

A successful MITM attack can result in the re-registration of the mobile device. This causes the re-transmission of the International Mobile Subscriber Identity (IMSI), which is linked to the person owning the account the SIM is registered to. This information can be used to identify an individual, even if the SIM card is moved to a different phone.

Invasion of Privacy

A successful MITM attack can record phone calls and SMS messages.

How valuable are your phone calls? What is the potential fallout from leaked financial information? A well-placed MITM device in a CFO's office opens the door for insider trading, industrial espionage, and market manipulation.

Malicious Equipment

Evil Base Transceiver Station (BTS)

For around $600 you can build your very own Evil BTS, for recording IMSI numbers and intercepting calls and SMS messages. (https://www.evilsocket.net/2016/03/31/how-to-build-your-own-rogue-gsm-bts-for-fun-and-profit/)

Compromised Femtocell Device

With the investment of $250 and some of your free time, you may be able to hack a femtocell. You will, however, have to find a way other than the method used here: https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2013/august/femtocell-presentation-slides-videos-and-app/ ... Verizon issued a patch to correct this vector.

Existing Detection Methods:

Android App: AIMSICD

https://github.com/CellularPrivacy/Android-IMSI-Catcher-Detector

AIMSICD interacts with your cell phone's radio to detect behavioral anomalies.

Android App: Femto Catcher

https://github.com/iSECPartners/femtocatcher

Similar in some ways to AIMSICD, Femto Catcher specifically targets femtocells from Verizon

Shell Scripts: FakeBTS

http://fakebts.com/en/

Written by Pedro Cabrera, FakeBTS was an early source of inspiration for SITCH. FakeBTS is a collection of shell scripts that use Kalibrate, Airprobe, and Wireshark to analyze GSM networks.

Commercial Appliance: Pwnie Express

According to this article: http://arstechnica.com/information-technology/2015/04/this-machine-catches-stingrays-pwnie-express-demos-cellular-threat-detector/ the appliance is a Pwn Pro with a 4G cellular device installed. Base price for the Pwn Pro is $2,675.00. Price for Pwn Pro with GSM hardware is unannounced.

Commercial Appliance: Bastille Networks

Bastille Networks offers visibility of wireless network devices from 100kHz to 6GHz, which, among other things, includes GSM networks. As of July 7th 2016, I've been unable to locate specifics on hardware or pricing.

Signals from the Noise: Indicators of Attack

Undocumented BTS

A BTS which broadcasts unexpected network identification information, or network information that doesn't exist in a database of BTSes, is an undocumented BTS. For instance, a CellID which has not been previously observed, could indicate an attack.

Relocated BTS

A BTS with identifying information which is not supposed to be nearby can indicate a malicious (or poorly configured) BTS. For example, a BTS which has been observed and documented in Roanoke, VA but is now appearing in San Francisco, CA is, at best, poorly configured. At worst, in the process of registering with this BTS, your phone could leak your IMSI and cause a degradation in your call encryption settings, or the BTS could record your conversations and SMS messages.

Gratuitous BTS Change

A gratuitous BTS change can occur when a more powerful signal appears for your network, causing your phone to behave as if it were moving into range for another serving BTS. While this happens quite often for a mobile station, a stationary monitoring device should seldom, associate with a different BTS.

Rapid Signal Strength Increase

When monitoring the signal strength for observable Absolute Radio Frequency Channel Numbers (ARFCNs), after a short period of time, a noise floor can be established. This can be accomplished by setting an alerting threshold just above the maximum strength observed for the strongest ARFCN signal.

Solution Constraints

Physical

* Small footprint favored for portability and placement versatility

Price

* Inexpensive sensors ($100 target price)

Sensor Management

* Fleet-wide central configuration management
* Hands-off software update

Input Devices

* Preference toward SDR, for flexibility

Data Representation

* JSON documents representing scan results
* Time-series measurements for specific metrics, like signal strength

Analysis

* ARFCN signal strength threshold exceeded
* ARFCN signal strength anomaly detection against forecast (Holt-Winters)
* Change in serving BTS
* Cell metadata change (MCC, MNC, LAC, etc...)
* Use of unassigned network identifiers (bad MCC, MNC)
* Appearance of network identifiers in wrong geolocation

Output and Integration

* Ability to deliver events to other log retention or SIEM systems
* Browseable/searchable event history
* Browseable time-series measurements
* Meaningful alerts (limit esoteric knowledge requirements)
* Alerts delivered through Slack

Situational Information from Telemetry and Correlated Heuristics (SITCH)

Hardware Platform Selection

While the Intel NUC and Odroid XU-4 platforms are quite capable, both have higher power requirements than the Raspberry Pi 2. The powerful NUC was priced quite a bit higher than the ARM platforms and the Odroid's availability and requirement for active cooling added unnecessary complexity. The Raspberry Pi 2 platform was chosen because its processing power was sufficient and the power and cooling requirements are such that it could be easily run off of a USB battery pack supplying 2A of power.

Device and Firmware Management

Resin.io is the platform used to manage all the physical nodes in the solution. The Resin OS runs Docker on your Raspberry pi. The SAAS side of Resin's offering contains a Docker registry, a build system, and a git service. When you push code to Resin it attempts to build a Docker image according to the instructions in the Dockerfile. If successful, it will add it to Resin's Docker registry. Your devices check every minute for updated container images and if a newer image exists, the device downloads it and restarts the application. Application configuration items (alerting threshold, target GSM bands, etc...) are managed through environment variables configured on a per-application or per-device basis, in the Resin UI.

Input Device Options

* NooElec NESDR (R820T)
  * ~$20
  * Fine for GSM-850
* NooElec NESDR (E4000)
  * ~$40
  * Broader range covers GSM all the way up to 1900MHz
* Fona SIM808 GSM breakout
  * ~$50 + ~$25 for accessories (battery, antenna, SIM)
  * Quad-band

Intelligence Feeds

* OpenCellId Database
  * Geo for BTS (MCC, MNC, LAC, CellID)

    * Effective range for BTS
  * Twilio API
    * Pricing API contains information to correlate MCC, MNC to provider name

Telemetry Delivery

  * Logstash forwarder
    * Manages its own cache
    * Minimal tweaking to build for ARM

Telemetry Aggregation and Storage

  * Elasticsearch
  * Graphite
  * Logstash

Analysis Tools

  * Graphite-beacon
    * Simple, straightforward setup
    * Alerts to Slack
  * Kibana
    * Search scan documents
    * Pretty dashboards, if you're into that kind of thing
  * Some custom code (SITCH enricher/alarms)

Alerts

  * Slack
    * Alerts from Logstash and Graphite-beacon
    * Apps for all the things

Design Principles

  * Push analysis and alerting as low in the stack as possible
    * Each sensor has its own feed DB, performs its own enrichment
      * What else are you going to do with all that compute?
      * Reserve resources on the service-side analysis tools for cross-device and cross-site correlation
  * Flexible Components
    * Use of queues as buffer between collection, enrichment, transmission processes
    * Build with the expectation of expansion to support other devices and detection methods

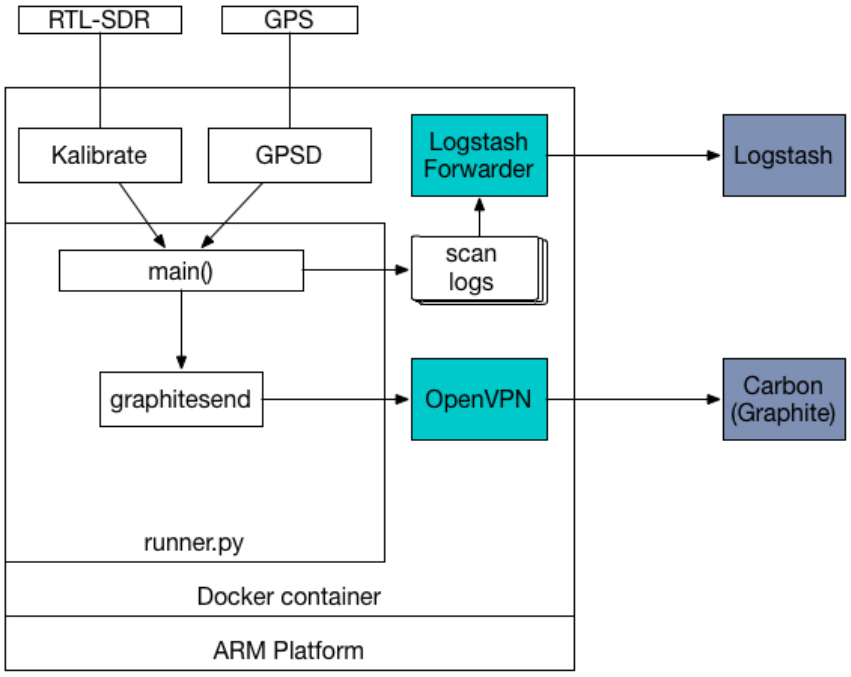
SITCH Sensor: MkI

Sensor hardware

    * Raspberry Pi 2
    * NooElec NESDR R820T
    * GlobalSat USB GPS

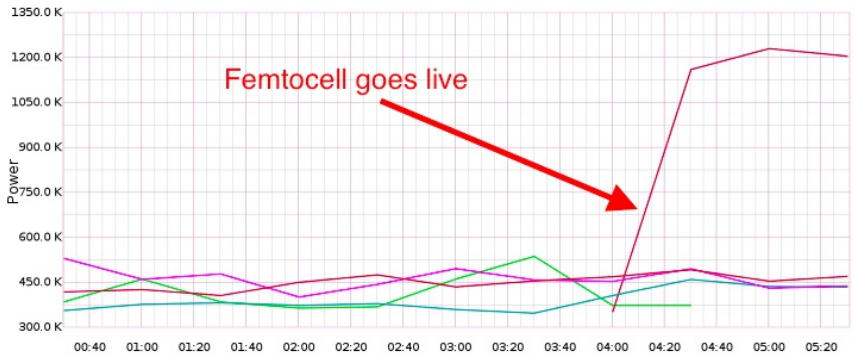
Sensor Software

Single threaded collection process. Sensor logs are transmitted through Logstash and time-series information is sent over OpenVPN to Graphite.



Results

The MkI sensor positively detected the presence of a femtocell using Kalibrate's ARFCN power detection



Target Functionality	MkI
Signal over threshold	Y
Signal outside forecast	Y
Unknown BTS	N
Primary BTS Change	N
Tower beyond range	N

Lessons Learned and Path Forward

* What you can do with a little ARM board, and what you can't...yet:
  * Gnuradio + GR-GSM are too heavy for R-Pi 2
  * Kalibrate takes around seven minutes to scan GSM-850
* Desired results for the next iteration:
  * Cell tower metadata for enrichment and correlation
  * Improve resolution (Less than seven minutes for results)

SITCH Sensor: MkII

Sensor Hardware

* Raspberry Pi 2
* Fona SIM808 (GSM modem)
* NooElec E4000 (SDR device, capable of 1900 MHz)

Sensor Software

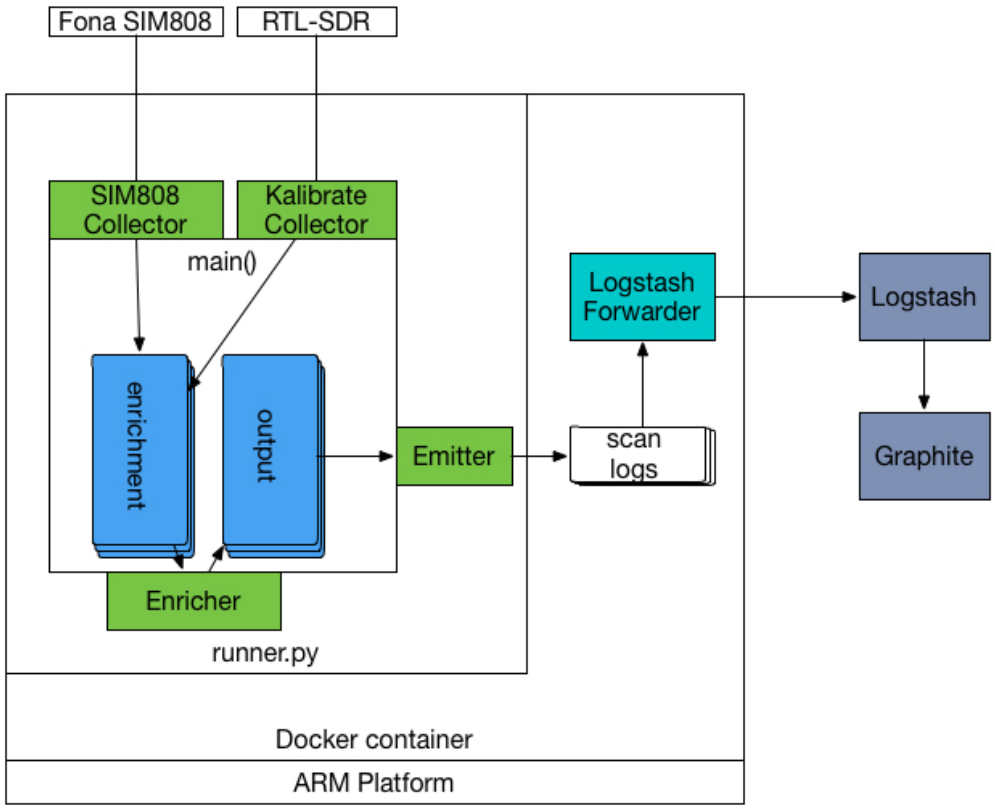
Separate threads for managing each input device, enrichment, and event delivery. All telemetry flows through Logstash, and service-side the time-series information is extracted and sent to Graphite.

All information produced by the attached hardware goes into an enrichment queue, where it is retrieved by the enricher process. The enricher process works by adding data from the intelligence feed, splitting channels into individual messages, making a determination on whether to create alarm events based on changes in primary cell metadata or ARFCN signal strength being beyond threshold, or other anomalies in observed behavior.

Information Lifecycle

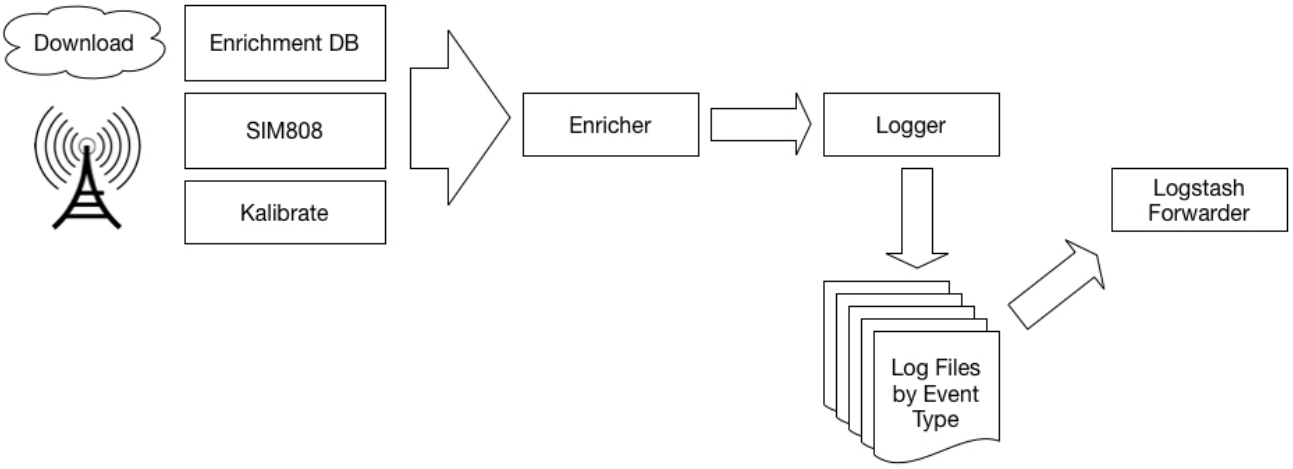
There is a separate process that runs once daily to retrieve the OpenCellID database and enrich it with Twilio's indication of who owns the MCC/MNC of each result in the DB. This job is run in a Docker container and publishes the resulting database in files containing all known cells for an MCC on a specific band, to an Amazon S3 bucket. The Enricher process on the individual sensors will periodically attempt to download the newest database files for its target MCCs.

The Sensor runs a thread for each collector- one for Kalibrate (RTL-SDR) and one for a small library that consumes troubleshooting information from the SIM808 device.



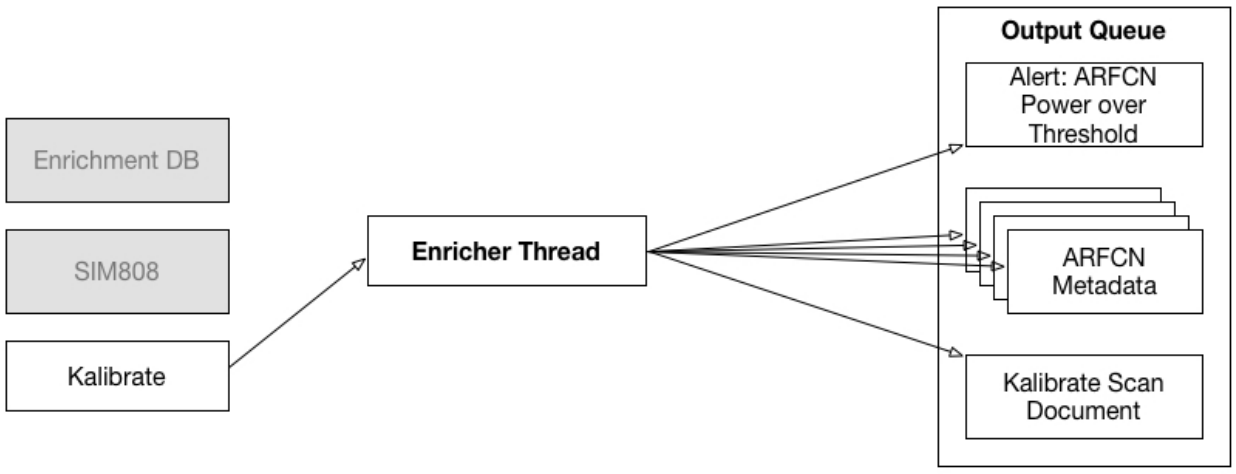
As results are produced by the collector threads, they are queued up for the enricher component. The enricher breaks down scan documents into messages for specific metrics, like ARFCN power, and puts them into the outbound message queue. The enricher also looks for anomalies in the stream of data passing through the sensor. It calculates the distance between the Sensor (location determined by GeoIP)

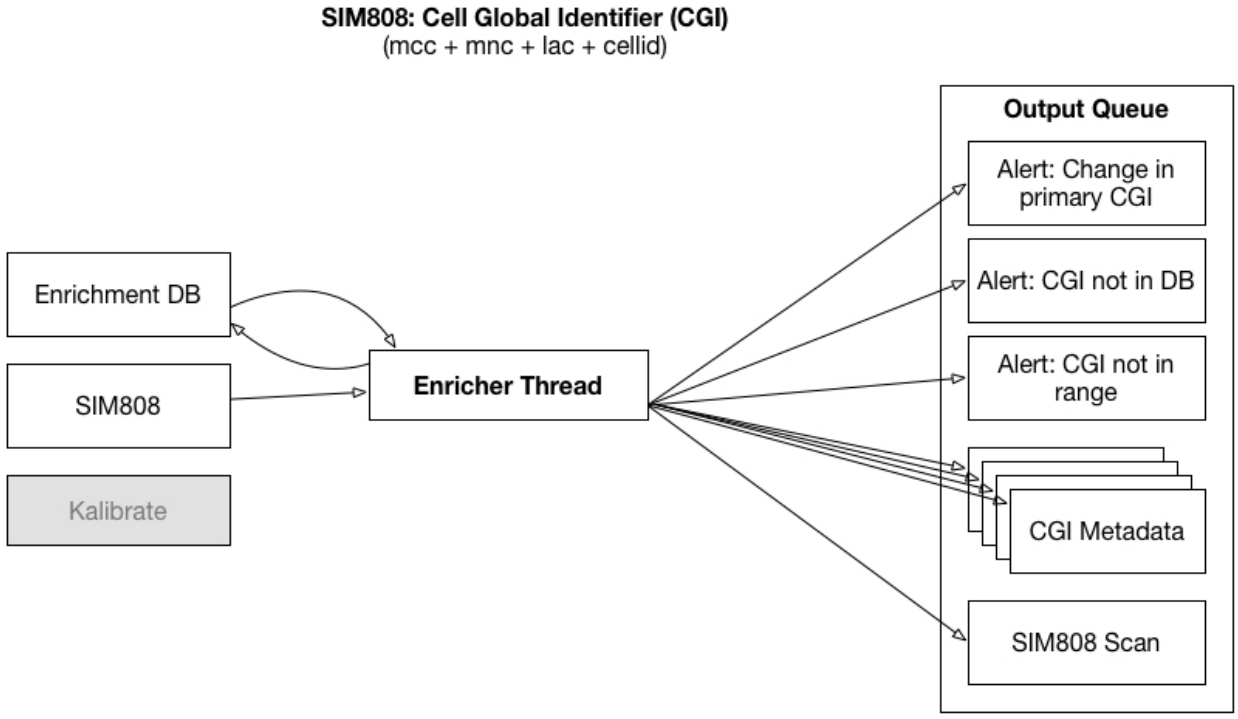
Information Flow Within Sensor

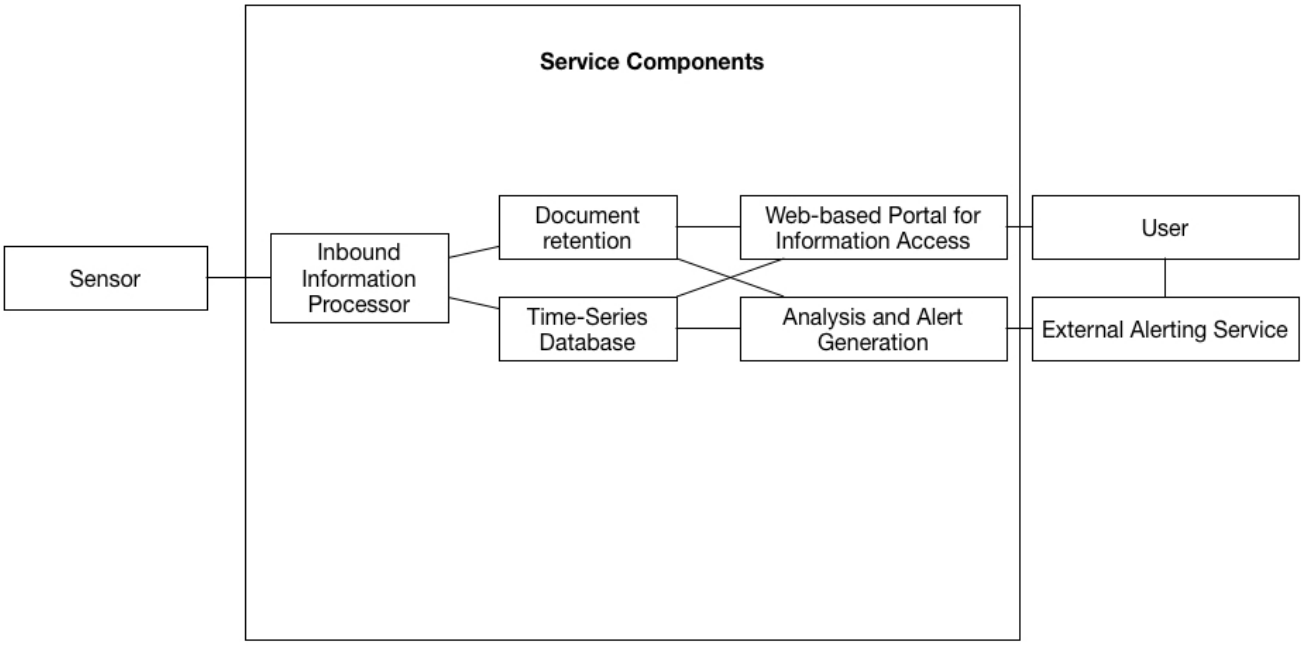


and the observed cell towers (location from OpenCellID), and will alert if the observed distance is beyond what is expected. The enricher also tracks the current BTS and if the cell metadata changes, it will put an alert message in the outbound message queue. The enricher is also capable of generating alerts if the observed signal strength is over the configured threshold.

Kalibrate: signal strength



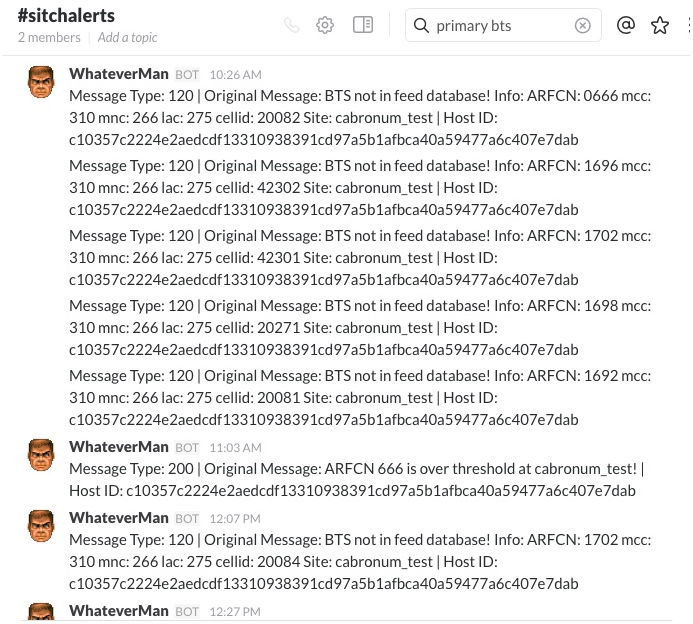

SIM808: Cell Global Identifier (CGI)
(mcc + mnc + lac + cellid)
Output Queue
Alert: Change in primary CGI
Alert: CGI not in DB
Alert: CGI not in range
CGI Metadata
SIM808 Scan
Enrichment DB
SIM808
Kalibrate
Enricher Thread

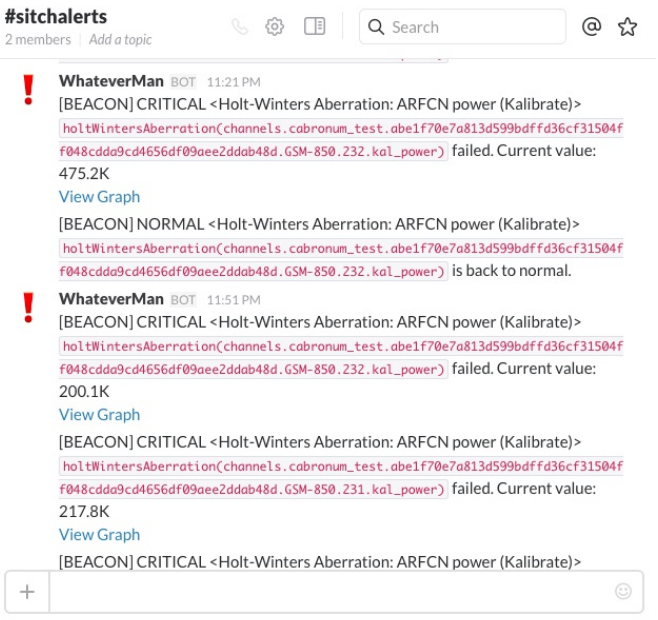

Service Components
Sensor
Inbound Information Processor
Document retention
Time-Series Database
Web-based Portal for Information Access
Analysis and Alert Generation
User
External Alerting Service

Alerting

Alerts are sometimes generated on the Sensor and passed up to the Service for delivery to the user, and sometimes the Service itself makes an observation and alerts the user. Primary alerts are generated on the Sensor and delivered to the user after being processed by the Service.

Secondary alerting occurs when the Service observes an anomaly. For instance, if a sensor's observed signal strength is outside of what's forecasted or a sensor does not send any information for too great a period of time.





Retention

Scan and alert events are retained in an Elasticsearch system, and are searchable using Kibana. Time-series data is generated by Logstash and sent to Graphite for retention and processing. Tessera is the dashboard chosen for presenting this information to the user. Graphite Beacon handles alerting on statistical anomalies in time-series data (specifically Holt-Winters forecast) as well as Sensors going silent.

Summary

Target Functionality	MkI	MkII
Signal over threshold	Y	Y
Signal outside forecast	Y	Y
Unknown BTS	N	Y
Primary BTS Change	N	Y
Tower beyond range	N	Y

I've shown that with around $100 of equipment one can easily detect the presence of a femtocell or other GSM device, using only signal strength threshold monitoring (MkI).

For around $150 a sensor can be built which is capable of making more granular observations, and alert more meaningfully on anomalies detected in GSM networks (MkII).

Source code will be released at https://sitch.io after DEF CON 24 concludes.

Looking Forward:

If Gnuradio and GR-GSM can be optimized to run on ARM devices, GR-GSM's scanner can be used to get results similar to what we're getting from the SIM808 GSM modem. This will allow us to go back to a strictly SDR platform, which offers much more flexibility for adding other RF monitoring capabilities without adding more specialized radio hardware.

U.S. Patent Pending