REDIS COMMAND ANALIZER

A Sysdig Chisel to list Redis queries by parsing Sysdig event.

How the Chisel works

in order to explain better my project, I would like to introduce some basic concept about Sysdig and Redis.

Sysdig

Sysdig instruments your physical and virtual machines at the OS level by installing into the Linux kernel and capturing system calls and other OS events. Then, using sysdig's command line interface, you can filter and decode these events in order to extract useful information. Sysdig can be used to inspect systems live in real-time, or to generate trace files that can be analyzed at a later stage.

[from www.sysdig.org]

Another method by which, through Sysdig, you can capture and analyze events from the kernel Linux is represented by so-called Chisel, LUA scripts, through which you can filter and decode system calls in order to extract information.

Redis

Redis is an open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server since keys can containstrings, hashes, lists, sets and sorted sets.

[from www.redis.io]

Redis is a Client-Server application, the Client trought a GUI, a command lines interface or from other source get the Commands and send those to the server, is important to specificate, in order to understand better this document, that the redis-client speaks with the redis-server using read's and write's Unix System Calls.

Chisel General Description

As mentioned earlier, using a Lua script (Sysdig Chisel) is possible to filter and parse read's and write's system calls events in order to extract useful information like:

- Number of execution of a Redis Command.
- Command Rate per Second.
- Average of time spent serving request by Redis Server.

Assuming to use Redis Client to generate Redis Command, I will explain how my Chisel works in every particular.
In order to perform what I said before, I must intercept write's and read's system calls from "redis-server"; the read's will represents the command sent by a Redis-Client (that could be the owner client or a other client built using Redis Protocols rules) while the write's will indicate the result of the query over the data. Write's System Calls are able to indicate how long the command is performed by the server.

To make it possible I used this filter over Sysdig event:

```
chisel.set_filter(" proc.name = redis-server and ( evt.type = read or evt.type = write ) and evt.dir = < ")
```

This filter wraps the write's and read's system call of redis server with the < direction, that is Sysdig intercept System calls when those exit from the Kernel.
After specifying to the Sysdig Chisel the filter I can start to parse the command in order to store that into a LUA table (very nice tool) and calculate the statistics above that.

Number of execution of a Redis Command

I simply count the command once parsed.

Command Rate per Second

This stats represent the number of commands per second, I obtain this stats by dividing the Command's count for the time that the Chisel was up.

Average of time spent serving request by Redis Server.

Each time arrived a read Calls from the redis-server I save the current time until I get the corresponding write System Call, when it happens I do the substraction between the write time and the read times, then I add that result to the sum of previous differences and finally I dividing this for Command count.
As I do for Command count I use a LUA table to store the sum of difference of read's and write's arriving times.

Command vs Non Command

The most significant implementation choices that I' ve done in this project was "How to differentiate from command and not command" in fact, altought the Redis protocol establishes that RESP Array (The notation-way how command are sent) was coding with a string that starts with the * character, this specification allowed Inline Commands, that is the command was not sent with the starting * character but are human readable string.
In order to make the Chisel consistent with all non-proprietary Client that use, or not, Inline Command notation, I decide to established if a string is a Command throught the couple of read and write System calls, The first read system call is the command, other following read's could represent other non command strings.

How Run Redis Command Analizer

The Chisel could run into three different mode, Normal Mode, Verbouse Mode and Very Verbouse Mode, those three modality will be indicated by the following abbreviations N, V, VV.
First of all, In order to run the Chisel is necessary have an active Redis-Client that generates the Redis Command, I used Ntopng on wlan0 interface, after that it possible to start the Chisel by moving inside the folder where the file "RedisCommandAnalizer.lua" is located, and then running it by the command:

```
sysdig -c RedisCommandAnalizer.lua %i
```

The integer %i discriminate the modality into that you want to start the Chisel, I'll explain briefly those usage modality.

%i = 0 N mode

%i = 1 V mode

%i = 2 VV mode

Normal Mode

In the N mode all the command will be hidden and nothing will show, in this modality will only be possible to interrupt, whenever you want, the Chisel in order to see result.

Verbouse Mode

Into the V mode the Chisel will show you the Command at any time was receveid by the server, you could see an ouput like this:

LPOP
GET
SET

Similarly the normal mode you need to stop the script to see the results, the interruption is performed through the holds of the keys CTRL + C.

Very Verbouse Mode

In this modality the output is just like a command appear in a read request, it is useful to understand what I said previously about protocol RESP and Inline Comment (Try using in this modality both ntopng and the propietary redis-client to generate the commands).

The holds of CTRL+C interrupt the Chisel and show you result.