Casr-Cluster: Crash Clustering for Linux Applications

Georgy Savidov, Andrey Fedotov

3 december 2021

ISP RAS

Idea

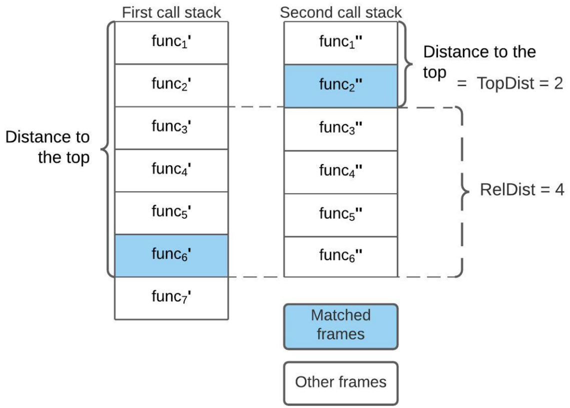
- Linux application development is closely related with debugging and fixing various bugs. Our tool was created to help developers in this difficult task.
- We propose a tool Casr-cluster, that groups "similar"crashes (information about which is contained in Casr reports) together as clusters.
- First of all, our tool is focused on clustering crashes obtained by fuzzing programs / libraries written in the C programming language.

Metrics

The main component of the crash, on the basis of which we determine how much crash reports differ, is the call stack. We used some approaches to calculate the similarity and clustering similar to Microsoft's ReBucket method. We use the following two pseudometrics in our algorithm:

- *TopDist* - the minimal position offset of the current frame relative to the topmost one.
- *RelDist* - distance between matched frames in two call stacks.

Example



First call stack
Second call stack
func1'
func2'
func3'
func4'
func5'
func6'
func7'
func1''
func2''
func3''
func4''
func5''
func6''
Distance to the top
Distance to the top = TopDist = 2
RelDist = 4
Matched frames
Other frames

About algorithm in general

Based on these pseudometrics, we make the following assumptions regarding the similarity of the two crashes:

- The closer the matching frames are to the top of the call stacks, the greater the *TopDist* weight.
- The smaller the distance between the matching frames in call stacks, the greater the weight of *RelDist*.

In our method of calculating the similarity, we also used a dynamic programming algorithm to find the largest common subsequence of two sequences(LCSP*).

*Longest common subsequence problem

Longest common subsequence

How to find the length of the longest common subsequence?
Let's say we are looking for a solution for the case (n_1, n_2), where n_1, n_2 are the lengths of the first and second lines. Let there already exist solutions for all subproblems (m_1, m_2) less than a given one. Then problem (n_1, n_2) is reduced to smaller subproblems as follows:

$$f(n_1, n_2) = \begin{cases} 0, & n_1 = 0 \vee n_2 = 0 \\ f(n_1 - 1, n_2 - 1) + 1, & s_1[n_1] = s_2[n_2] \\ max(f(n_1 - 1, n_2), f(n_1, n_2 - 1)), & s_1[n_1] \neq s_2[n_2] \end{cases} \quad (1)$$

The complexity of the algorithm is $O(n1 * n2)$.

		A	B	C	D
	0	0	0	0	0
D	0	← 0	← 0	← 0	↖ 1
C	0	← 0	← 0	↖ 1	← 1
D	0	← 0	← 0	↑ 1	↖ 2
A	0	↖ 1	← 1	← 1	↑ 2

Equivalence of frames

Two frames (call sites) are matched

- If the frames have the same **module** and the same **offset in this module** (+ the same **offset in line** if presents). These attributes can be obtained from debug information or mappings.
- If the frames have the same **linear addresses**, if we have not any other information (module + offset). But in this case, we do not take into account the ASLR, which is often present in the system.

This mechanism for stack traces comparison was implemented in an open source library*.

*https://github.com/xcoldhandsx/gdb-command

Clustering algorithm

First stage.

$$M[i][j] = max \begin{cases} M[i][j-1], \\ M[i-1][j], \\ M[i-1][j-1] + addition(i,j) \end{cases} \quad (2)$$

$$addition(i,j) = \begin{cases} e^{-r*|i-j|-a*min(i,j)}, & f_{1,i} \ = \ f_{2,j} \\ 0 & otherwise \end{cases} \quad (3)$$

$$dist(a,b) = 1 - similarity(a,b)$$

Where:

- `'r'` - *RelDist coefficient*.
- `'a'` - *TopDist coefficient*.

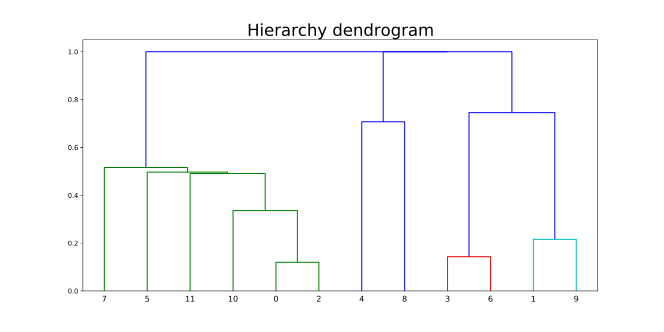
A compressed distance matrix is formed, composed of pairwise distances between the crash stack traces.

Clustering algorithm

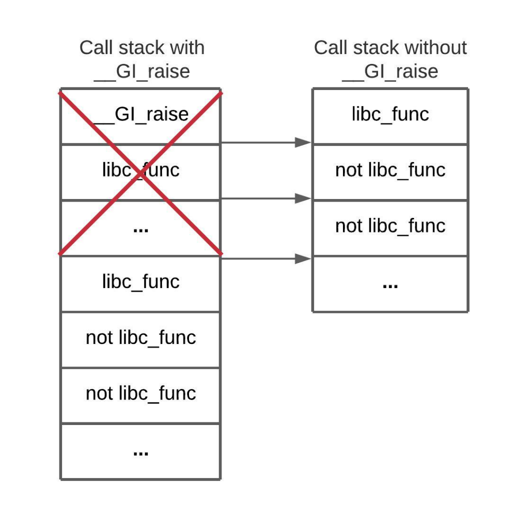
Second stage.

$$CLdist(CL_i, CL_j) = \max_{a \in CL_i, b \in CL_j}(dist(a, b)) \quad (4)$$

Hierarchical clustering is started based on the distance matrix obtained in the first stage. The distance between two clusters is defined as the maximum of the pairwise distance between crashes retrieved from the two clusters.

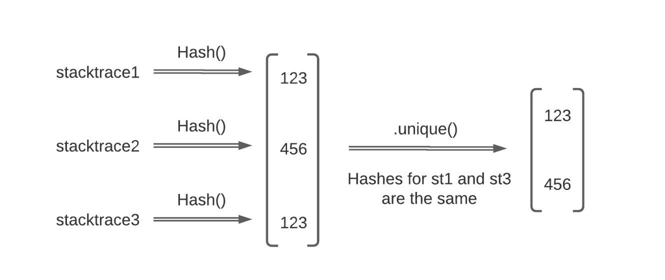


Removing "noise"from stack trace



Deduplication

- We also implemented a method for reports deduplication. two crashes are considered the same if $\mathit{similarity}(a, b) = 1$. For this, all frames in both call stacks must be equal.
- We can hash each frame and the entire Call Stack using attributes such as name of the file to which frames belong and offset from its beginning.



Result table

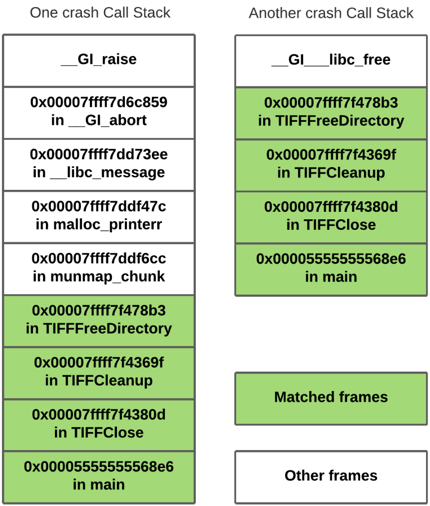
Casr-cluster Testing Results

Library	Crash reports	Unique crashes	Number of clusters	Average number of reports in cluster	Execution time(sec)	Deduplication time(sec)
libxml2	49	10	9	1	1.3	0.08
jasper	231	55	26	2	3.8	0.21
lame	74	15	7	2	1.4	0.06
openjpeg	264	72	36	2	6.5	0.24
libtiff	155	56	40	1	4.0	0.14
libarchive	306	5	3	2	1.2	0.24
lrzip	38	12	9	1	1.3	0.05
poppler	763	29	15	2	2.3	1.14
TOTAL:	1880	253	154	2		

Results

- Clustering was performed with the coefficients $a = 0.04$, $r = 0.13$ and the threshold value $d = 0.3$.
- The number of crash reports has decreased by about an order of magnitude, they were replaced by clusters (with deduplication by about 1.7 times).
- Some clusters have more than 3 crashes. In the poppler library, one of the clusters contains 6 similar (but not the same) crashes.
- The clustering time is at an acceptable level.

Example



One crash Call Stack
__GI_raise
0x00007ffff7d6c859 in __GI_abort
0x00007ffff7dd73ee in __libc_message
0x00007ffff7ddf47c in malloc_printerr
0x00007ffff7ddf6cc in munmap_chunk
0x00007ffff7f478b3 in TIFFFreeDirectory
0x00007ffff7f4369f in TIFFCleanup
0x00007ffff7f4380d in TIFFClose
0x00005555555568e6 in main
Another crash Call Stack
__GI___libc_free
0x00007ffff7f478b3 in TIFFFreeDirectory
0x00007ffff7f4369f in TIFFCleanup
0x00007ffff7f4380d in TIFFClose
0x00005555555568e6 in main
Matched frames
Other frames

Conclusion

- We propose crash clustering method, based on call stack comparison. Method could be applied to crash reports, collected via Casr tool for Linux systems.
- We use optimization for call stack comparison when call stack has libc abort function call.
- Before applying the clustering algorithm, you should first deduplicate the crashes for best performance.