RcppDeepState, a simple way to fuzz test Rcpp packages

Akhila Chowdary Kolla,
Joint work with Dr.Toby Dylan Hocking and Dr.Alex Groce
Sponsored by R Consortium

July 5, 2021

Problem : Subtle Bugs in the Code

Rcpp function: read_value()

```
#include <Rcpp.h>
using namespace std;

// [[Rcpp::export]]
int read_value(int size, int access_var){
     int *read_array = new int[size];
     for(int i = 0 ; i < size ; i++){
         read_array[i] = 10;
     }
     return read_array[access_var];
}
```

Output:

```
> read_array(100, 50)
[1] 10

> read_array(100, 1000) -- ??
[1] 1914846579

> read_array(100, -1) -- ??
[1] 32683
```

Introduction to Fuzzing and DeepState

DeepState

- Fuzzing is the generation of invalid, unexpected, or random data as inputs to a computer program expecting it to crash, fail, or generate errors.
- DeepState is a means to run unit tests with a lot of fuzzers. Unit tests in DeepState are called as test harnesses.
- Normally, DeepState's test harnesses are written in C++ providing a common interface to various symbolic execution tools, fuzzing engines, such as Angr, AFL, HonggFuzz, LibFuzzer, Eclipser.
- Fuzzers provide standard inputs without type-specific values and generate a stream of random data whereas DeepState provides an advantage over fuzzers by providing C/C++ typed input functions (DeepState_Int(), DeepState_Double(), etc) which are useful for producing fuzzer specific datatypes in Rcpp.
- Code for DeepState is available at: **https://github.com/trailofbits/deepstate**.

Purpose of RcppDeepState

- We have developed the RcppDeepState tool, an easy-to-use fuzzing system for R packages using the Rcpp framework.
- Used for memory debugging and memory leak detection.
- Increases speed and efficiency by using C++ test harnesses in R libraries.
- It generalizes and automates the test harness generation.
- Easy interfacing with fuzzers like AFL, LibFuzzer for input generation.
- It does the hard work of automating the matching of multiple function parameters with type-specific data generators.
- Code for RcppDeepState is available at: **https://github.com/akhikolla/RcppDeepState**

Related Work

- Inspiration from vroom standalone package which follows a do-it-by-hand approach to compile and run the AFL fuzzer on the package. No external frameworks are implemented to provide an interface with an Rcpp package and fuzzers.
- RUnit, testthat, tinytest, and unitizer conventional unit test frameworks that work on predefined assertions on provided inputs.
- CRAN checks perform automatic code analysis and run tests under sanitizers (e.g., clang/LLVM's ASAN) and even under Valgrind on manually included tests.
- Fuzzr, R package that fuzz tests R functions on predefined inputs or default suites. Doesn't support full-featured automated test generation for the R code.
- autotest, R package that mutates the inputs for only the documented functions in the R packages.

Testing BNSL::mi

mi_cmi.cpp : BNSL::mi

```

// [[Rcpp::export]]
double mi(NumericVector x, NumericVector y, int proc=0){
  if(proc==0)return(Jeffreys_mi(x,y,0,0));
  else if(proc==1)return(MDL_mi(x,y,0,0));
  else if(proc==2)return(BDeu_mi(x,y,0,0,1));
  else if(proc==3)return(empirical_mi(x,y));
  else if(proc==9)return(empirical_mi(x,y));
  else if(proc==10)return(cont_mi(x,y));
  else return(Jeffreys_mi(x,y,0,0));
}
```

- x and y are numeric vectors.
- proc estimation is based on Jeffrey's prior, the MDL principle, BDeu, and empirical principle.
- If the argument proc is missing, proc=0 (Jeffreys') is assumed.

Existing R examples

command : 'example("mi",package="BNSL")'

```
mi> n=100

mi> x=rbinom(n,1,0.5); y=rbinom(n,1,0.5); mi(x,y)
[1] 0

mi> z=rbinom(n,1,0.1); y=(x+z); mi(x,y);
[1] 0.5204089

mi> x=rnorm(n); y=rnorm(n); mi(x,y,proc=10)
[1] 0

mi> x=rnorm(n); z=rnorm(n); y=0.9*x+sqrt(1-0.9^2)*z; mi(x,y,proc=10)
[1] 0.4281646
>
```

When run under Valgrind:

```
> res <- deepstate_read_valgrind_xml("mi_example.R")
> res
Empty data.table (0 rows and 5 cols): err.kind, message, file.line,
     address.msg, address.trace
```

Why RcppDeepState Here?

- Is this code bug-free as we didn't see any issues when we run the predefined examples under Valgrind? - No.
- Is testing the code on predefined inputs enough?- No, predefined inputs are unable to find subtle bugs.
- Is there a way to automatically run the code on various unexpected or randomized inputs? - RcppDeepState.

mi_DeepState_TestHarness.cpp : Test Harness for BNSL::mi

```cpp
#include <RInside.h>
#include <iostream>
#include <RcppDeepState.h>
#include <DeepState.hpp>

double mi(NumericVector x, NumericVector y, int proc);

TEST(BNSL_deepstate_test,mi_test){
  RInside R;
  NumericVector x  = RcppDeepState_NumericVector();
  NumericVector y  = RcppDeepState_NumericVector();
  IntegerVector proc(1);
  proc[0]  = RcppDeepState_int();
  try{
    mi(x,y,proc[0]);
  }
  catch(Rcpp::exception& e){
    std::cout<<"Exception Handled"<<std::endl;
  }
}
```

- Input values for numeric vectors x and y are obtained from RcppDeepState's fuzzer functions RcppDeepState_NumericVector().

Test Harness Generation & Compilation & Execution

deepstate_fuzz_fun(package_path,fun,time.limit.seconds=3)

- We parse the src/RcppExports for the prototypes, the list of parameters of the provided function.
- Allows test harness generation of provided Rcpp function.
- Compiles and runs the test harness for the provided Rcpp function.
- Generates crash/fail/pass extension files depending upon the type of response obtained by running the inputs on the executable.
- time.limit.seconds allows users to specify the fuzzing time on the executable.

Test Harness Analysis

deepstate_fuzz_fun_analyze(fun_path,seed,time.limit.seconds)

- Allow users to analyze the binary files of the provided Rcpp function under the presence of Valgrind. It also looks for errors/bugs if there are any.
- Allows users to specify a seed value that can serve as the start point of the fuzzing.
- time.limit.seconds allows users to specify the time limit to analyze the executable.
- Returns a data table with the inputs and the error messages and the position where the error occurred.

Identifying Memory Issues in code using RcppDeepState

Test Harness generation & Fuzz run for function

```
> pkg.path <- "/home/akhila/BNSL"
> fun <- "mi"
> RcppDeepState::deepstate_fuzz_fun(pkg.path,fun,time.limit.seconds=3)
[1] "mi"
```

Analyzing the compile run for function

```
> path <- "/home/akhila/BNSL/inst/testfiles/mi"
> seed_analyze <- deepstate_fuzz_fun_analyze(path,time.limit.seconds=10)
running the executable ..

> seed_analyze

      inputs          logtable
1: <list[3]> <data.table[1x5]>

> str(seed_analyze$logtable)
List of 1
 $ :Classes  data .table and 'data.frame':  1 obs. of  5 variables:
  ..$ err.kind      : chr "InvalidRead"
  ..$ message       : chr "Invalid read of size 8"
  ..$ file.line     : chr "mi_cmi.cpp : 55"
  ..$ address.msg   : chr "Address 0x9d66358 is 0 bytes after a block of
    size 184 alloc'd"
  ..$ address.trace: chr NA
  ..- attr(*, ".internal.selfref")=<externalptr>
```

Identifying Memory Issues in code using RcppDeepState

Analyzing the compile run for function

```
> seed_analyze$inputs
[[1]]
[[1]]$proc
[1] -1178760776

[[1]]$x
 [1]  5.111874e+27 -4.615293e-56 -4.504466e-15 -1.692003e-48
 [5] -1.351623e-65  2.651995e-25 -1.223792e-18  1.335638e-244
 [9] -1.507216e+20  9.426978e-46  1.040939e+14 -9.491862e-138
[13] -3.067944e-30 -1.303296e-27  4.310657e+29  3.034607e-186
[17]  5.375657e-03 -2.595575e-50 -2.613731e-19 -1.412455e+107
[21]  3.405566e-18 -8.424350e+19  6.909683e+25 -2.022733e-131

[[1]]$y
 [1]             NA  -9.912803e-89 -5.524951e-177   4.628963e+15
 [5] -1.249944e+271  -5.473829e-15  1.183622e+118  -4.707083e+33
 [9]  2.053698e+222  -6.608259e+267             NA            NaN
[13]  2.047837e+096  -2.521985e+209            Inf  -1.832406e-20
[17]  0.000000e+00
```

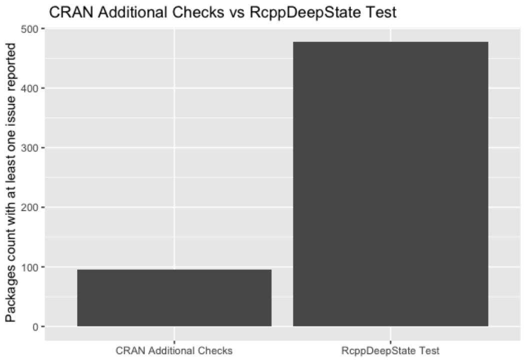
Identifying Memory Issues in code using RcppDeepState

mi_cmi.cpp : line 55

```
// [[Rcpp::export]]
double Jeffreys_mi(NumericVector x,NumericVector y,int m_x=0,int m_y=0)
{
  IntegerVector c_x=table(x),c_y=table(y),c_xy=table(x+1000*y);
  if(m_x==0)m_x=c_x.size(); if(m_y==0)m_y=c_y.size();
  int n=x.size();
  double S_x=gc(n,0.5*m_x)-gc_all(c_x,0.5),S_y=gc(n,0.5*m_y)-gc_all(c_y
     ,0.5),S_xy=gc(n,0.5*m_x*m_y)-gc_all(c_xy,0.5),S=(S_x+S_y-S_xy)/n;
  if (S<=0) return(0); else return (S);
}
```

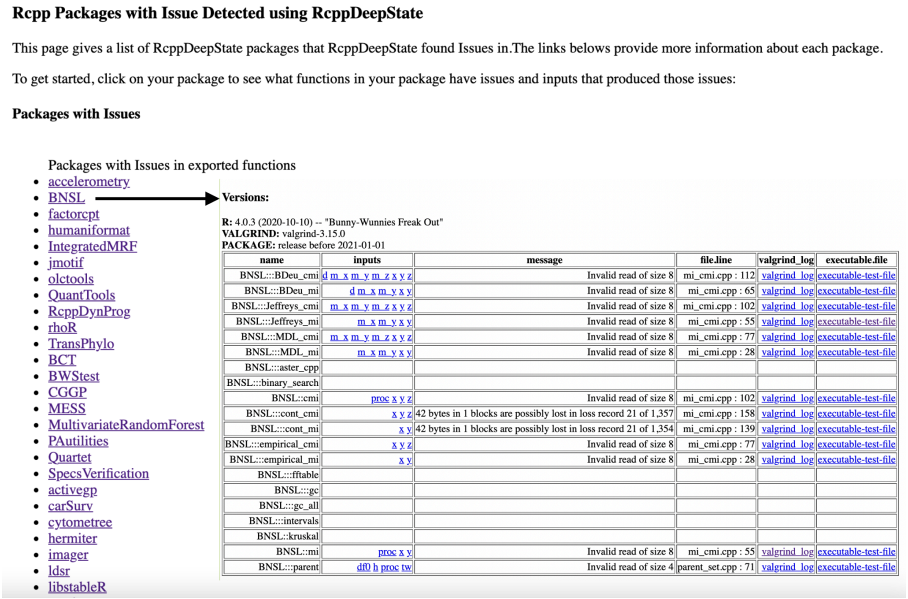
- As proc value is -1178760776 the mi function makes a call to Jeffreys_mi(x,y,0,0).
- The issue occurs when c_xy generates a table from both x and y vectors.
- If the size of vectors x and y are not equal the system generates an issue, that causes an invalid read. Therefore we need to specify a condition to check if the size of the vectors x and y are equal.

Results of RcppDeepState



- This analysis was made on packages that were downloaded as of **2020-12-20**.
- These additional CRAN checks include clang-UBSAN, gcc11, M1mac, donttest, gcc-UBSAN, rchk, clang-ASAN, ATLAS, gcc-ASAN.
- We fuzz tested 1,185 Rcpp packages and it reported issues for more than 1,000 functions (over nearly 412 packages) which were not detected using standard CRAN checks run on manually specified test/example inputs.

Web page for RcppDeepState Test Results



Rcpp Packages with Issue Detected using RcppDeepState

This page gives a list of RcppDeepState packages that RcppDeepState found Issues in.The links belows provide more information about each package.

To get started, click on your package to see what functions in your package have issues and inputs that produced those issues:

Packages with Issues

Packages with Issues in exported functions

- accelerometry
- BNSL
- factorcpt
- humaniformat
- IntegratedMRF
- jmotif
- olctools
- QuantTools
- RcppDynProg
- rhoR
- TransPhylo
- BCT
- BWStest
- CGGP
- MESS
- MultivariateRandomForest
- PAutilities
- Quartet
- SpecsVerification
- activegp
- carSurv
- cytometree
- hermiter
- imager
- ldsr
- libstableR

Versions:

R: 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
VALGRIND: valgrind-3.15.0
PACKAGE: release before 2021-01-01

name	inputs	message	file.line	valgrind_log	executable.file
BNSL:::BDeu_cmi	d m_x m_y m_z x y z	Invalid read of size 8	mi_cmi.cpp : 112	valgrind_log	executable-test-file
BNSL:::BDeu_mi	d m_x m_y x y	Invalid read of size 8	mi_cmi.cpp : 65	valgrind_log	executable-test-file
BNSL:::Jeffreys_cmi	m_x m_y m_z x y z	Invalid read of size 8	mi_cmi.cpp : 102	valgrind_log	executable-test-file
BNSL:::Jeffreys_mi	m_x m_y x y	Invalid read of size 8	mi_cmi.cpp : 55	valgrind_log	executable-test-file
BNSL:::MDL_cmi	m_x m_y m_z x y z	Invalid read of size 8	mi_cmi.cpp : 77	valgrind_log	executable-test-file
BNSL:::MDL_mi	m_x m_y x y	Invalid read of size 8	mi_cmi.cpp : 28	valgrind_log	executable-test-file
BNSL:::aster_cpp					
BNSL:::binary_search					
BNSL::cmi	proc x y z	Invalid read of size 8	mi_cmi.cpp : 102	valgrind_log	executable-test-file
BNSL:::cont_cmi	x y z	42 bytes in 1 blocks are possibly lost in loss record 21 of 1,357	mi_cmi.cpp : 158	valgrind_log	executable-test-file
BNSL:::cont_mi	x y	42 bytes in 1 blocks are possibly lost in loss record 21 of 1,354	mi_cmi.cpp : 139	valgrind_log	executable-test-file
BNSL:::empirical_cmi	x y z	Invalid read of size 8	mi_cmi.cpp : 77	valgrind_log	executable-test-file
BNSL:::empirical_mi	x y	Invalid read of size 8	mi_cmi.cpp : 28	valgrind_log	executable-test-file
BNSL:::fftable					
BNSL:::gc					
BNSL:::gc_all					
BNSL:::intervals					
BNSL::kruskal					
BNSL::mi	proc x y	Invalid read of size 8	mi_cmi.cpp : 55	valgrind_log	executable-test-file
BNSL:::parent	df0 h proc tw	Invalid read of size 4	parent_set.cpp : 71	valgrind_log	executable-test-file

Available at : https://akhikolla.github.io./packages-folders/root.html

Experimental Results

Table: Summary Of Results

Type	# Packages	# Functions
Total Rcpp Packages	2149	19735
No RcppExports	247	NA
At least one unfuzzable type	717	12,875
Executed under RcppDeepState	1,185	6,860
afl valgrind issues (corpus)	27	59
libFuzzer valgrind issues	73	117
DeepState Fuzzer valgrind issues	478	911
Package (`test/example`)	0	0
Exported	74	156
Unexported	406	755

- RcppDeepState's default fuzzer identified issues in 478 packages showing a better performance compared to well known mutation based fuzzers like AFL and libfuzzer.
- RcppDeepState identified issues in 156 exported functions (over 74 packages) and 755 unexported functions (over 406 packages).

Conclusion

Future Work

- Improved fuzz testing with more realistic random inputs.
- Extending Random generation functions for SEXP, List datatypes.
- Include RcppDeepState as part of CRAN checks.

Thanks

- R Consortium for funding RcppDeepState.
- Dr.Toby Dylan Hocking and Dr.Alex Groce for mentoring this project.

Contact Information
github : github.com/akhikolla/RcppDeepState
email : akolla1504@gmail.com