

# **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**

August 23, 2021

## R

- Software environment for statistical computing and graphics.
- Used among statisticians and data miners for developing statistical software and data analysis.
- R package is a fundamental unit of shareable code.
- The standard distribution mechanism for R packages is the CRAN (Comprehensive R Archive Network).

## Rcpp

- Interfaces R to C/C++ code.
- A set of utilities to make writing C++ more convenient for R users.

Table: R vs C++

Features	R	C++
Garbage Collector	YES	NO
Fastest	NO	YES
Memory Efficient	NO	YES
Safe Language	YES	NO

- R memory is not freed when the data is removed.
- R is slow, often slower than native C/C++.
- R aggressively over-uses memory.

# Why is it needed to test Rcpp libraries

- Software system integrate libraries and testing these libraries is unlikely.
- The basic testing approach for these libraries is by a fixed set of input/output pairs.
- Such Manually written tests allow for subtle bugs, since they do not anticipate all possible inputs and miss important code paths.

## Rcpp library Structure

	Name	Size	▼ Modified
	..		
<input type="checkbox"/>	DESCRIPTION	1.8 KB	Apr 1, 2021, 7:25 AM
<input type="checkbox"/>	LICENSE	41 B	Apr 1, 2021, 7:25 AM
<input type="checkbox"/>	MD5	10.5 KB	Apr 1, 2021, 7:25 AM
<input type="checkbox"/>	NAMESPACE	1.1 KB	Apr 1, 2021, 7:25 AM
<input type="checkbox"/>	NEWS.md	4 KB	Apr 1, 2021, 7:25 AM
<input type="checkbox"/>	README.md	1.8 KB	Apr 1, 2021, 7:25 AM
<input type="checkbox"/>	data		
<input type="checkbox"/>	inst		
<input type="checkbox"/>	man		
<input type="checkbox"/>	R		
<input type="checkbox"/>	src		
<input type="checkbox"/>	tests		

## Problem : Subtle Bugs in the Code

### Rcpp function: read\_value()

```
1 #include <Rcpp.h>
2 using namespace std;
3
4 // [[Rcpp::export]]
5 int read_value(int size, int access_var){
6     int *read_array = new int[size];
7     for(int i = 0 ; i < size ; i++){
8         read_array[i] = 10;
9     }
10    return read_array[access_var];
11 }
```

### Output:

```
1 > read_array(100, 50)
2 [1] 10
3
4 > read_array(100, 1000) -- ??
5 [1] 1914846579
6
7 > read_array(100, -1) -- ??
8 [1] 32683
```

- read from an undeclared/inaccessible memory
- write to an index out of bounds/inaccessible memory
- use after free
- use after delete
- referencing to a uninitialized/null pointer

## Valgrind

- Valgrind Memcheck is a tool that detects memory leaks and memory errors.
- Like the debugger, Valgrind runs on your executable, so be sure you have compiled an up-to-date copy of your program.
- Run it like this, for example, if your program is named memoryLeak:

## Using Valgrind:

---

```
akhilachowdarykolla@AkhilaKollaMacBookPro%$ valgrind ./memoryLeak
==4649== ERROR SUMMARY: 0 errors from 0 contexts
==4649== malloc/free: in use at exit: 0 bytes in 0 blocks.
==4649== malloc/free: 10 allocs, 10 frees, 2640 bytes allocated.
==4649== For counts of detected errors, rerun with: -v
==4649== All heap blocks were freed – no leaks are possible.
```

---

## Fuzzing

- 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

- 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.
- Code for DeepState is available at: <https://github.com/trailofbits/deepstate>.



- Fuzzers provide standard inputs without type-specific values.
- DeepState provides C/C++ typed input functions (DeepState\_Int(), DeepState\_Double(), etc) which are useful for producing fuzzer specific datatypes in Rcpp.

### DeepState\_Int() :

```
akhilachowdarykollaAkhilaKollaMacBookPro inputs% cat integers  
-22198547 -1839144542 -1543701651 -640104457 686298858 1367200796  
-1318345480 1204187134 -1644300991 -101601582 -915027215 1591794863  
1588644879 460326988 999791065 1204187134 2143838281 1042954908 186010473  
-971540255 -2004865780
```

### DeepState\_Double() :

```
akhilachowdarykollaAkhilaKollaMacBookPro inputs% cat doubles  
-1.450026e-210 4.589782e+224 -7.722098e-23 5.902083e+79 -3.830617e-208  
1.865128e+23 [2] 16.419437e+30 9.461046e-28 1.295939e+277 -3.016591e-306  
-3.109608e-93 -1.662483e+24 2.579611e-78 4.736890e-82 -4.045699e+22  
-1.900983e-02 -1.523905e-231 -1.149843e+177
```

# 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>**

- 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.

## mi\_cmi.cpp : BNSL::mi

```
1
2 // [[Rcpp::export]]
3 double mi(NumericVector x, NumericVector y, int proc=0){
4     if(proc==0) return(Jeffreys_mi(x,y,0,0));
5     else if(proc==1) return(MDL_mi(x,y,0,0));
6     else if(proc==2) return(BDeu_mi(x,y,0,0,1));
7     else if(proc==3) return(empirical_mi(x,y));
8     else if(proc==9) return(empirical_mi(x,y));
9     else if(proc==10) return(cont_mi(x,y));
10    else return(Jeffreys_mi(x,y,0,0));
11 }
```

- 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")'**

```
1 mi> n=100
2
3 mi> x=rbinom(n,1,0.5); y=rbinom(n,1,0.5); mi(x,y)
4 [1] 0
5
6 mi> z=rbinom(n,1,0.1); y=(x+z); mi(x,y);
7 [1] 0.5204089
8
9 mi> x=rnorm(n); y=rnorm(n); mi(x,y,proc=10)
10 [1] 0
11
12 mi> x=rnorm(n); z=rnorm(n); y=0.9*x+sqrt(1-0.9^2)*z; mi(x,y,proc=10)
13 [1] 0.4281646
14 >
```

When run under Valgrind:

```
1 > res <- deepstate_read_valgrind_xml("mi_example.R")
2 > res
3 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, predefined inputs are unable to find subtle bugs and are unable to explore all the possible input paths.
- Is there a way to automatically run the code on various unexpected or randomized inputs? - Yes, using RcppDeepState.

## **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.

## mi\_DeepState\_TestHarness.cpp : Test Harness for BNSL::mi

```
1 #include <RInside.h>
2 #include <iostream>
3 #include <RcppDeepState.h>
4 #include <DeepState.hpp>
5
6 double mi(NumericVector x, NumericVector y, int proc);
7
8 TEST(BNSL_deepstate_test, mi_test){
9     RInside R;
10    NumericVector x = RcppDeepState_NumericVector();
11    NumericVector y = RcppDeepState_NumericVector();
12    IntegerVector proc(1);
13    proc[0] = RcppDeepState_int();
14    try{
15        mi(x,y,proc[0]);
16    }
17    catch(Rcpp::exception& e){
18        std::cout<<"Exception Handled"<<std::endl;
19    }
20 }
```

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



## **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

```
1 > pkg.path <- "/home/akhila/BNSL"
2 > fun <- "mi"
3 > RcppDeepState::deepstate_fuzz_fun(pkg.path,fun,time.limit.seconds=3)
4 [1] "mi"
```

## Analyzing the compile run for function

```
1 > path <- "/home/akhila/BNSL/inst/testfiles/mi"
2 > seed_analyze <- deepstate_fuzz_fun_analyze(path,time.limit.seconds=10)
3 running the executable ..
4
5 > seed_analyze
6
7         inputs          logtable
8 1: <list[3]> <data.table[1x5]>
9
10 > str(seed_analyze$logtable)
11 List of 1
12 $ :Classes   data   .table and 'data.frame':  1 obs. of  5 variables:
13 ..$ err.kind   : chr "InvalidRead"
14 ..$ message    : chr "Invalid read of size 8"
15 ..$ file.line  : chr "mi_cmi.cpp : 55"
16 ..$ address.msg : chr "Address 0x9d66358 is 0 bytes after a block of
    size 184 alloc'd"
17 ..$ address.trace: chr NA
18 ..- attr(*, ".internal.selfref")=<externalptr>
```

# Identifying Memory Issues in code using RcppDeepState

## Analyzing the compile run for function

```
1 > seed_analyze$inputs
2 [[1]]
3 [[1]]$proc
4 [1] -1178760776
5
6 [[1]]$x
7 [1] 5.111874e+27 -4.615293e-56 -4.504466e-15 -1.692003e-48
8 [5] -1.351623e-65 2.651995e-25 -1.223792e-18 1.335638e-244
9 [9] -1.507216e+20 9.426978e-46 1.040939e+14 -9.491862e-138
10 [13] -3.067944e-30 -1.303296e-27 4.310657e+29 3.034607e-186
11 [17] 5.375657e-03 -2.595575e-50 -2.613731e-19 -1.412455e+107
12 [21] 3.405566e-18 -8.424350e+19 6.909683e+25 -2.022733e-131
13
14 [[1]]$y
15 [1] NA -9.912803e-89 -5.524951e-177 4.628963e+15
16 [5] -1.249944e+271 -5.473829e-15 1.183622e+118 -4.707083e+33
17 [9] 2.053698e+222 -6.608259e+267 NA NaN
18 [13] 2.047837e+096 -2.521985e+209 Inf -1.832406e-20
19 [17] 0.000000e+00
```

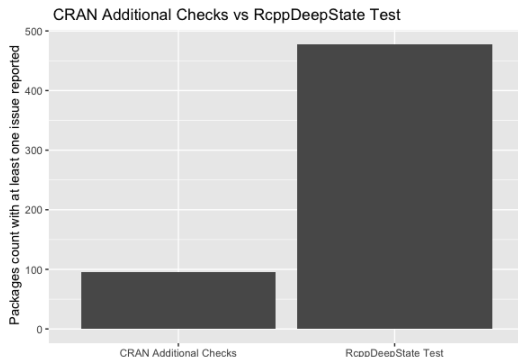
# Identifying Memory Issues in code using RcppDeepState

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 below 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](#) →
- [factorecpt](#)
- [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	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
BNSL::BDeu_mi	d m x m y x y	Invalid read of size 8	mi_cmi.cpp : 65	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
BNSL::Jeffreys_cmi	m x m y m z x y z	Invalid read of size 8	mi_cmi.cpp : 102	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
BNSL::Jeffreys_mi	m x m y x y	Invalid read of size 8	mi_cmi.cpp : 55	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
BNSL::MDL_cmi	m x m y m z x y z	Invalid read of size 8	mi_cmi.cpp : 77	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
BNSL::MDL_mi	m x m y x y	Invalid read of size 8	mi_cmi.cpp : 28	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
BNSL::aster_cpp					
BNSL::binary_search					
BNSL::cmi	proc x y z	Invalid read of size 8	mi_cmi.cpp : 102	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
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	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
BNSL::cont_mi	x y	42 bytes in 1 blocks are possibly lost in loss record 21 of 1,354	mi_cmi.cpp : 139	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
BNSL::empirical_cmi	x y z	Invalid read of size 8	mi_cmi.cpp : 77	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
BNSL::empirical_mi	x y	Invalid read of size 8	mi_cmi.cpp : 28	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
BNSL::ftable					
BNSL::gc					
BNSL::gc_all					
BNSL::intervals					
BNSL::kruskal					
BNSL::mi	proc x y	Invalid read of size 8	mi_cmi.cpp : 55	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>
BNSL::parent	dfo h proc tw	Invalid read of size 4	parent_set.cpp : 71	<a href="#">valgrind_log</a>	<a href="#">executable-test.file</a>

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

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).

Table: Valgrind Error Count

Error Type	DeepState	afl	libFuzzer
Invalid read	405	11	20
Invalid write	97	13	53
Use of uninitialized values	6	7	1
Conditional jump or move(s)	113	7	3
Argument size	16	0	0
Possibly lost data	274	34	16

- Invalid read: there was an attempt to read from an invalid memory address.
- Invalid write: there was an attempt to write to an invalid memory address.
- Use of uninitialised values: there was a read from an uninitialized value in memory.
- Conditional jump or move(s): a branch depends on uninitialized data.
- Argument size: an incorrect value was passed to a system call, e.g., `malloc(-1)`
- Possibly lost data: there is a potential memory leak, where allocated memory was not deallocated.



## 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](https://github.com/akhikolla/RcppDeepState)

email : [akolla1504@gmail.com](mailto:akolla1504@gmail.com)