CERIAS

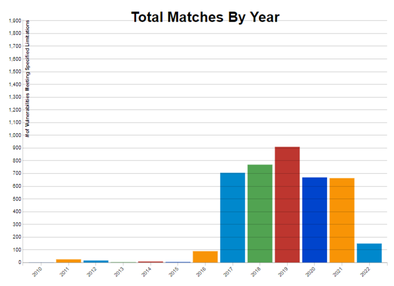
The Center for Education and Research in Information Assurance and Security

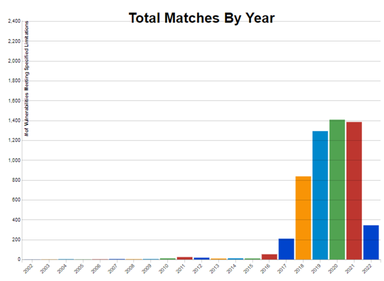
CheckCBox: Automated and Zero Cost Spatial Memory Safety

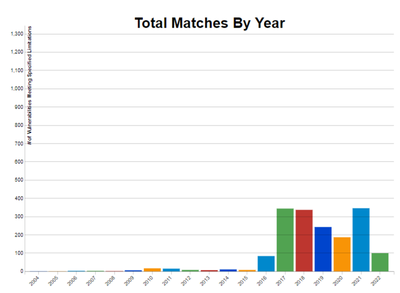
Arun Kumar, Aravind Machiry
Purdue University

PurS3

The Never Ending Trend of Spatial Safety Violations







Out-of-bounds Read | Out-of-bounds Write | NULL Pointer Dereference

Spatial Safety Violations still are the Major class of vulnerabilities in Low-level system software.

Existing Approaches Have High Overhead (Porting and Performance)

Safe languages

LEGACY CODE

Safe by design: Prevents memory corruption vulnerabilities.

Rust Programming Language | Golang

What about **Legacy code**? Not feasible to rewrite.

Retrofitting Techniques

Address Sanitizer (ASan)

SoftBound CETS

Slow (>= 50%).

Not backward compatible and need runtime changes.

Completely rewriting existing legacy code in Safe languages is not viable.

ASAN and SoftBound CETS □ High Performance Overhead
No Backward Compatibility and needs runtime changes

Checked C to Rescue

Checked C

Fast

Backward compatible

_Ptr
_Array_ptr
_Nt_array_ptr

Name	Code Changes LM %	Code Changes EM %	Code Changes LU %	Observed Overheads RT ±%	Observed Overheads CT ±%	Observed Overheads ES ±%
bh	10.0	76.7	5.2	+0.2	+23.8	+6.2
bisort	21.8	84.3	7.0	0.0	+7.3	+3.8
em3d	35.3	66.4	16.9	+0.8	+18.0	-0.4
health	24.0	97.8	9.3	+2.1	+18.5	+6.7
mst	30.1	75.0	19.3	0.0	+6.3	-5.0
perimeter	9.8	92.3	5.2	0.0	+4.9	+0.8
power	15.0	69.2	3.9	0.0	+21.6	+8.5
treeadd	17.2	92.3	20.4	+8.3	+83.1	+7.0
tsp	9.9	94.5	10.3	0.0	+47.6	+4.6
anagram	26.6	67.5	10.7	+23.5	+16.8	+5.1
ft	18.7	98.5	6.3	+25.9	+16.5	+11.3
ks	14.2	93.4	8.1	+12.8	+32.3	+26.7
yacr2	14.5	51.5	16.2	+49.3	+38.4	+24.5
Mean:	17.5	80.1	9.3	+8.6	+24.3	+7.4

Table 2

Benchmark Results. Key: *LM %*: Percentage of Source LoC Modified, including Additions; *EM %*: Percentage of Code Modifications deemed to be Easy; *LU %*: Percentage of Lines remaining Unchecked; *RT ±%*: Percentage Change in Run Time; *CT ±%*: Percentage Change in Compile Time; *ES ±%*: Percentage Change in Executable Size (`.text` section only). *Mean*: Geometric Mean.

Pointers annotated with Checked C types are guaranteed to not have any spatial violations

Can We Automatically Convert C to Checked C?

```
int main(int argc, char **argv) {
  int *b, *c;
  char *str = argv[0];
  b = &global;
  baz(b);
  c = b;
  printf("%d", strlen(str));
  return 0;
}
```

```
int *efunc();
int global;
int baz(int *p) {
   *p = 1;
    return 0;
}
```

```
int bar() {
   int *a = efunc();
   baz(a);
   return 0;
}
```

3C

```
int main(int argc, _Array_ptr<_Nt_array_ptr<char>> argv) {
  _Ptr<int> b;
  _Ptr<int> c;
  _Nt_array_ptr<char> str = argv[0];
  b = &global;
  baz(b);
  c = b;
  printf("%d", strlen(str));
  return 0;
}
```

```
int *efunc();
int global;
int baz(int *p: itype(_Ptr<int>)) {
   *p = 1;
    return 0;
}
```

```
int bar() {
   int *a = efunc();
   baz(a);
   return 0;
}
```

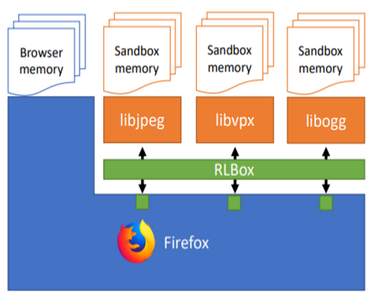
Complete Automated Conversion is *not Feasible*
Some regions of code will be still **unchecked**

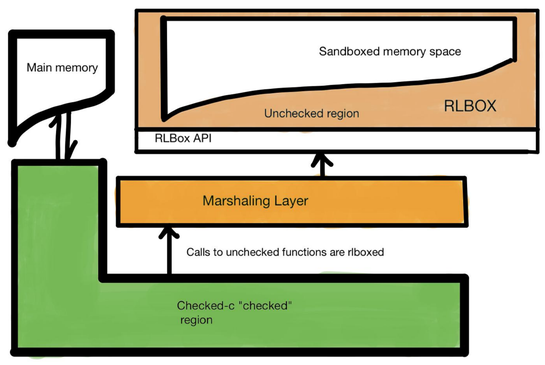
CheckCBox

RLBox

A C++ library that:

1. Abstracts isolation mechanism
   - Sandboxing with chosen isolation mechanism
   - Process, Native Client, WebAssembly, etc.
2. Mediates app-sandbox communication
   - APIs for control flow in/out of sandbox
   - `tainted` types for data flow in/out of sandbox






```
int main(int argc, _Array_ptr<_Nt_array_ptr<char>> argv) {
  _Ptr<int> b;
  _Ptr<int> c;
  _Nt_array_ptr<char> str = argv[0];
  b = &global;
  baz(b);
  c = b;
  printf("%d", strlen(str));
  return 0;
}
```

```
int *efunc();
int global;
int baz(int *p: itype(_Ptr<int>)) {
   *p = 1;
    return 0;
}
```

```
int bar() {
   int *a = efunc();
   baz(a);
   return 0;
}
```

Let's use RLBox to encapsulate unchecked regions and add marshalling between the regions.

CheckCBox: High level Idea

Challenges

- Automatically generating marshalling layers - Interaction between checked/unchecked/tainted types
- Handling Callbacks from “unchecked” region to “checked” region

Progress

- We were able to successfully encapsulate unchecked regions using RLBox and create required marshalling stubs
- Working on formalizing Checked C semantics with RLBox
- [On going] Working on automated encapsulation of unchecked regions into RLBox

Are you Curious?

- Open Source: https://github.com/purs3lab/CheckC-Box

Contact:

- Arun Kumar (bhattar1@purdue.edu)
- PurS3 Lab: https://purs3lab.github.io/

PURDUE UNIVERSITY Discovery Park

CERIAS