Language-Based Security Project Presentation

Analysis Tools for Race Detection

- **Group 11**
  Himanshu Chuphal
  Raya Altarabulsi

Agenda

- Project Goals
- Need for Tools?
- Race Detection Tools
- Tool Kit
- Evaluation and Results
- Tools Limitations and Comparison
- Insights
- Questions

Project Goals

- Experiment with tools for race detection in threaded programs
- Common Types of races by the tool
- Compare the different tools
- Installation and ease of Use
- Code Documentation
- Command Line Tools Kit

Need for Tools?

- Very hard to detect with traditional testing techniques
- Traditional software engineering testing methods are inadequate
- Difficult to find and reproduce
- Often happen under very specific circumstances
- Increase of reported vulnerabilities
- Scheduling nondeterminism and Programming errors

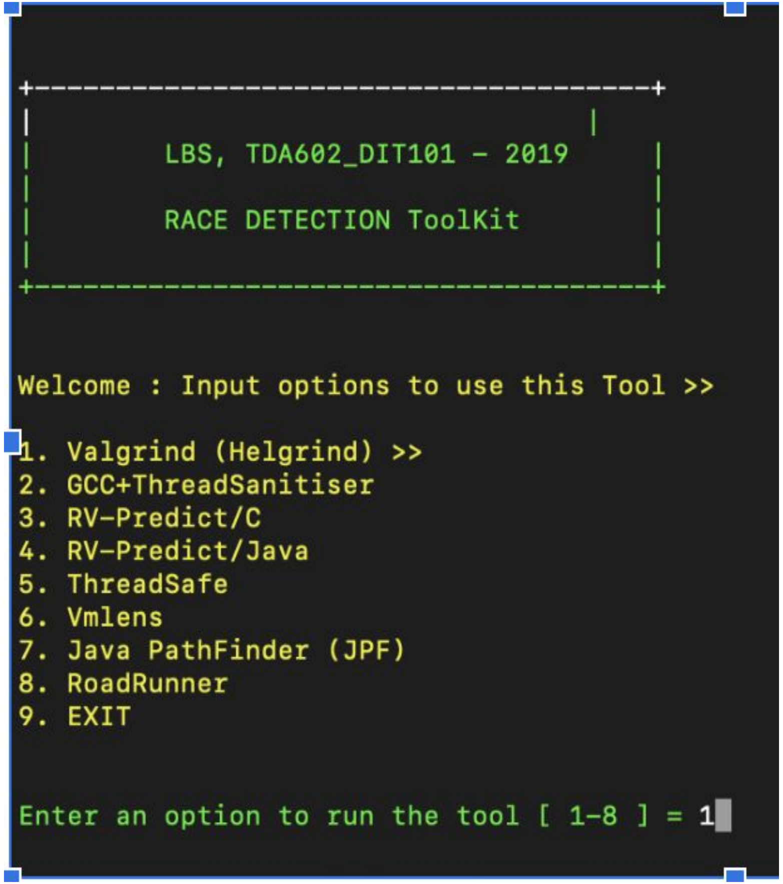
Race Detection Tools - 9+

- **9+ Tools and 40+ Use Cases (C/C++/Java)**
  - Valgrind (Helgrind)
  - GCC+ThreadSanitiser
  - Rv Predict
  - Vmlens
  - Java PathFinder
  - ThreadSafe
  - RoadRunner
  - FindBugs/SpotBugs
  - Coverity- Static Tool

Tool-Kit

- Use Cases : Shared variables access, Files read/Write, TOCTOU, other Multi-Threaded Programs
- Programming Language - C/C++ and Java
- Interface - Command line
- Information about :
  - Installation steps
  - Man-Page, URL
  - Programming language support
  - Use Cases etc.

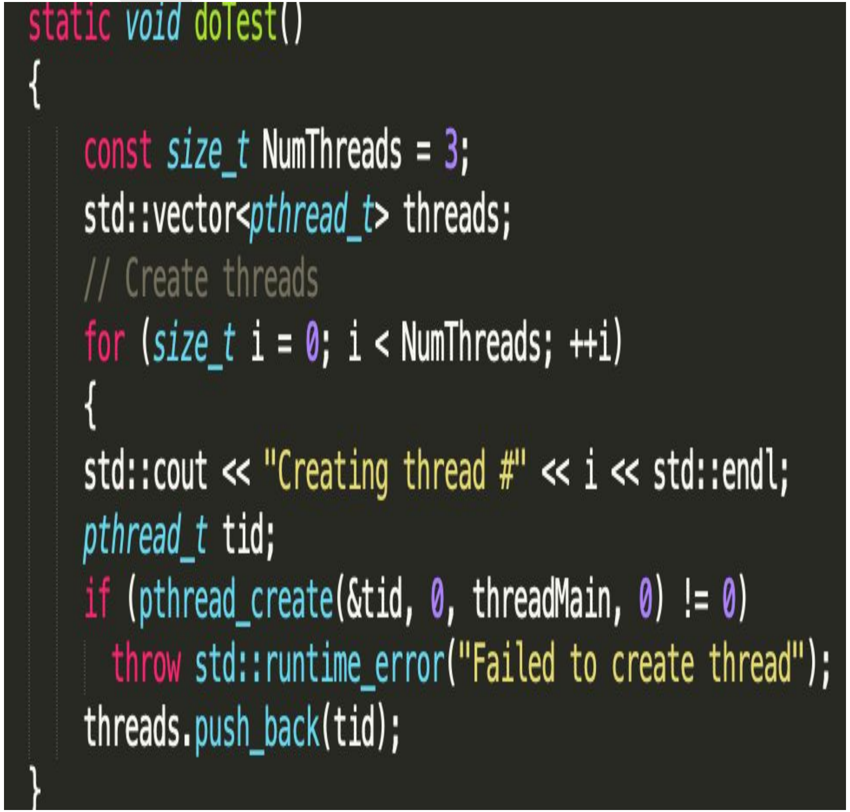
Note- Tools need to be installed 1st to use the kit.



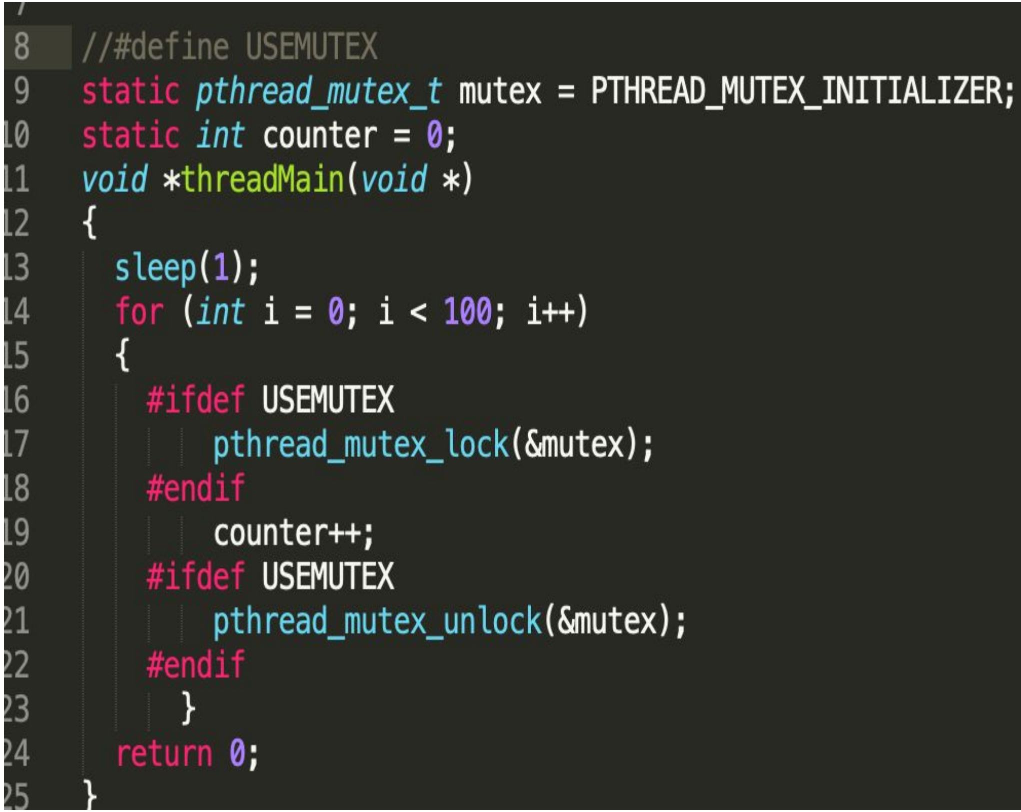
```
+----------------------------------------+
|                                     |
|        LBS, TDA602_DIT101 - 2019        |
|                                         |
|        RACE DETECTION ToolKit           |
|                                         |
+----------------------------------------+


Welcome : Input options to use this Tool >>

1. Valgrind (Helgrind) >>
2. GCC+ThreadSanitiser
3. RV-Predict/C
4. RV-Predict/Java
5. ThreadSafe
6. Vmlens
7. Java PathFinder (JPF)
8. RoadRunner
9. EXIT


Enter an option to run the tool [ 1-8 ] = 1
```

Vulnerable Program and Demo

Use-Case 1:



```
static void doTest()
{
    const size_t NumThreads = 3;
    std::vector<pthread_t> threads;
    // Create threads
    for (size_t i = 0; i < NumThreads; ++i)
    {
    std::cout << "Creating thread #" << i << std::endl;
    pthread_t tid;
    if (pthread_create(&tid, 0, threadMain, 0) != 0)
      throw std::runtime_error("Failed to create thread");
    threads.push_back(tid);
}
```



```
//#define USEMUTEX
static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
static int counter = 0;
void *threadMain(void *)
{
  sleep(1);
  for (int i = 0; i < 100; i++)
  {
    #ifdef USEMUTEX
        pthread_mutex_lock(&mutex);
    #endif
        counter++;
    #ifdef USEMUTEX
        pthread_mutex_unlock(&mutex);
    #endif
      }
  return 0;
}
```

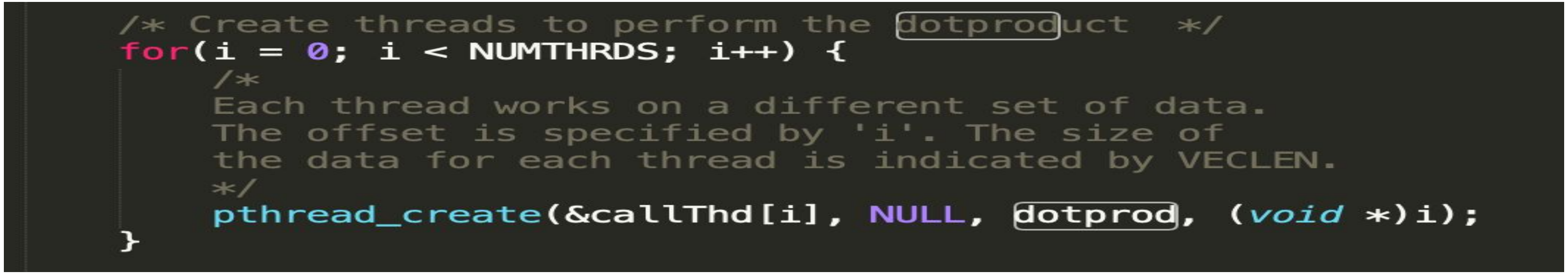
Evaluation and Results

- Valgrind (Helgrind) ::

```
==3301== Possible data race during read of size 4 at 0x30C2A8 by thread #3
==3301== Possible data race during write of size 4 at 0x30C2A8 by thread #2
```

- GCC++ThreadSanitiser by Google
  - WARNING: ThreadSanitizer: data race (pid=3377)
  - SUMMARY: ThreadSanitizer: data race (/media/sf_project/tools/valgrind/program/example+0x1455) in threadMain(void*)
- RV-Predict C/C++-- None
  - Creating thread #0
  - …….
  - pausing...
  - ...paused.
  - Stopping thread #0
  - …….
  - counter=298
  - Killed

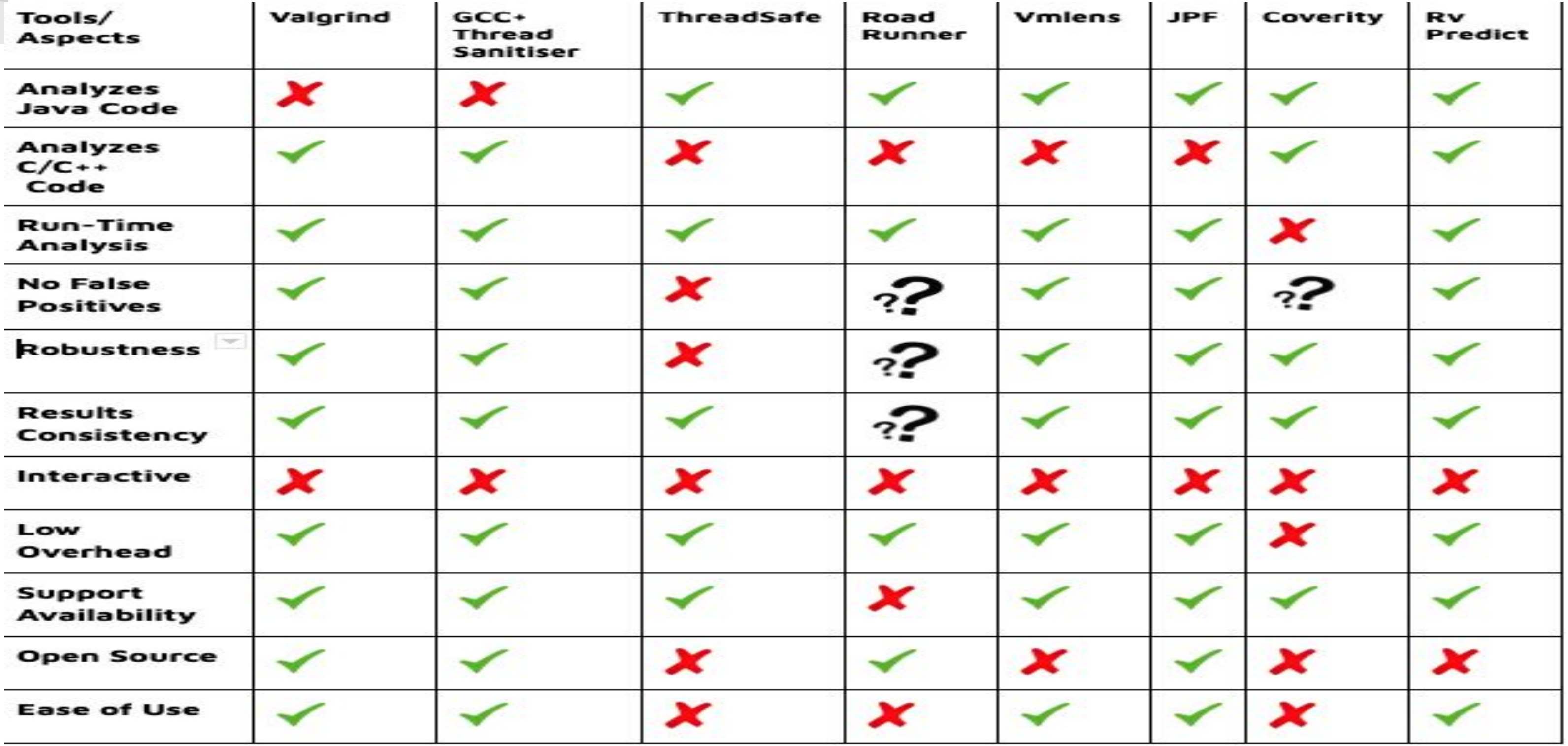
Use Case 2



```
/* Create threads to perform the dotproduct  */
for(i = 0; i < NUMTHRDS; i++) {
    /*
    Each thread works on a different set of data.
    The offset is specified by 'i'. The size of
    the data for each thread is indicated by VECLEN.
    */
    pthread_create(&callThd[i], NULL, dotprod, (void *)i);
}
```

- Rv-Predict C
    - Predict correctly *predicts* two data races. The first report describes the case where there can be a concurrent write at line 62, and a concurrent read in the printf statement ending at line 64:
    - Line 62, data race occurs because two threads concurrently read and write the shared variable dotstr.sum
- **ThreadSanitizer**
    - reports only one data race, specifically, a case where there are two concurrent writes to dotstr.sum
    - misses the race between lines 62 and 64 entirely.
- **Helgrind**
    - detect two data races related to concurrent writes or a concurrent read and a concurrent write at line 62,
    - is not able to predict a concurrent write at line 62 and a concurrent read at line 64.

Tools Comparison



Tools/ Aspects	Valgrind	GCC+ Thread Sanitiser	ThreadSafe	Road Runner	Vmlens	JPF	Coverity	Rv Predict
Analyzes Java Code	✗	✗	✓	✓	✓	✓	✓	✓
Analyzes C/C++ Code	✓	✓	✗	✗	✗	✗	✓	✓
Run-Time Analysis	✓	✓	✓	✓	✓	✓	✗	✓
No False Positives	✓	✓	✗	??	✓	✓	??	✓
Robustness	✓	✓	✗	??	✓	✓	✓	✓
Results Consistency	✓	✓	✓	??	✓	✓	✓	✓
Interactive	✗	✗	✗	✗	✗	✗	✗	✗
Low Overhead	✓	✓	✓	✓	✓	✓	✗	✓
Support Availability	✓	✓	✓	✗	✓	✓	✓	✓
Open Source	✓	✓	✗	✓	✗	✓	✗	✗
Ease of Use	✓	✓	✗	✗	✓	✓	✗	✓

Tools Limitations

- Predefined known vulnerability database detection
- Predictions, No in depth analysis
- Might generate false positives
- Not an Open source
- No active tools community
- Lack of support
- Dynamic Tools can’t detect all data races

Insight for Developers

- Tools are simple to use
- Use Multiple tools to be sure
- Good starting point to do manual security audits
- Command Line Interfaces
- Link with Continuous Integration (CI) of the Project
- Publish reports for developers
- Check Compiler warnings
- Contribute to the community
- Share Knowledge!

Q/A

Happy Coding

Any Questions?