Lab06 - Static Analysis

Dr. David J. Coe
Software Safety and Security Engineering Lab
Electrical and Computer Engineering Department

Sponsored by National Security Agency Grant H98230-17-1-0344

Outline

- Introduction to Static Analysis
- Coding Standards
  - Sample Rules
- Static analysis with open source programs
  - Using Cppcheck
- Lab06
  - Static Analysis of ec3.cpp
- Discussion
- Summary

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Introduction to Static Analysis

Sponsored by National Security Agency Grant H98230-17-1-0344

Introduction to Static Analysis - 1

- What is **`Static Analysis`**?
  - The methodology used when analyzing software without actually running the program
  - Requires access to the source code
- The purpose of static analysis is to identify common types of defects that may impact correct operation or security

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Introduction to Static Analysis - 2

- Static analysis uses various methods to accomplish its goal including:
  - Adherence to coding standards
  - Data flow analysis
  - Control flow analysis
  - Input validation
  - Error handling
- These areas can help identify coding errors that could lead to undesirable and unsafe output conditions

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Coding Standards

Sponsored by National Security Agency Grant H98230-17-1-0344

Coding Standards - 1

- Historically, ***coding conventions*** have focused on improving ***readability*** and ***maintainability*** by advocating
  - Consistency of presentation
  - Consistency of naming of variables, constants, functions, etc.
  - Consistency of documentation
  - Computational complexity: cyclomatic complexity
  - Etc.
- Coding conventions have evolved into ***coding standards*** to address ***defects*** that may manifest during operation
  - Runtime issues: use of uninitialized variables
  - Security issues: buffer overflow
  - Safety issues: unreachable code
- Some coding standards are intended for new development only, others may be applied to clean up legacy code

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Coding Standards - 2

- ***Motor Industry Software Reliability Association***
  - MISRA C and MISRA C++ coding standards
  - MISRA C goals
    - Safety, security, portability, and reliability of code implemented in C (according to Wikipedia)
- ***Joint Strike Fighter C++ Air Vehicle***
  - JSF++AV
  - http://www.stroustrup.com/JSF-AV-rules.pdf
- **High Integrity C++**
  - https://www.perforce.com/blog/qac/high-integrity-cpp-hicpp
- **CERT C / CERT JAVA**
  - https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard
  - Security-focused coding standards
- ***MathWorks Automotive Advisory Board***
  - MAAB Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow
- Like programming languages themselves, these standards evolve over time

Sponsored by National Security Agency Grant H98230-17-1-0344

Sample Rules

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

Sample Rules - 1

MISRA C

- MISRA C Rule 14.1
  - There shall be no unreachable code
- MISRA C Rule 14.7
  - A function shall have a single point of exit at the end of the function
- MISRA C Rule 9.1
  - All automatic variables shall have been assigned a value before being used
- MISRA C Rule 20.4
  - Dynamic heap allocation shall not be used

https://pubweb.eng.utah.edu/~cs5785/slides/08.pdf

Sponsored by National Security Agency Grant H98230-17-1-0344

Sample Rules - 2

MISRA C

- MISRA C Rule 5.2
  - Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier

```
int total;

int  foo(int total) {
   return 3 * total;
}
```

https://pubweb.eng.utah.edu/~cs5785/slides/08.pdf

Sponsored by National Security Agency Grant H98230-17-1-0344

Sample Rules - 3

MISRA C

- MISRA C Rule 17.6
  - The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist

```
int *  foo(void)  {
    int x;
    int *y = &x;
    return y;
}
```

https://pubweb.eng.utah.edu/~cs5785/slides/08.pdf

Sponsored by National Security Agency Grant H98230-17-1-0344

Sample Rules - 4

JSF++ AV

- AV Rule 1
  - Any one function (or method) will contain no more than 200 logical source lines of code

    (L- SLOCs).

- Rationale: Long functions tend to be complex and therefore difficult to comprehend and test.

http://www.stroustrup.com/JSF-AV-rules.pdf

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Sample Rules - 5

JSF++ AV

- AV Rule 206 (MISRA Rule 118, Revised)
  - Allocation/deallocation from/to the free store (heap) shall not occur after initialization.

- Rationale: repeated allocation (new/malloc) and deallocation (delete/free) from the free store/heap can result in free store/heap fragmentation and hence non-deterministic delays in free store/heap access.

http://www.stroustrup.com/JSF-AV-rules.pdf

Sponsored by National Security Agency Grant H98230-17-1-0344

Static Analysis with Open Source Programs

Sponsored by National Security Agency Grant H98230-17-1-0344

Static Analysis with Open Source Programs - 1

- While LDRA is a useful tool, it is one of many commercial products that are not free.
  - Luckily there are many open source tools available as well for the frugal developer.
- Be aware, commercial or otherwise, no one tool will always find all of the errors in a program.
  - It is advisable to develop testing plans that make use of several different tools.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Static Analysis with Open Source Programs - 2

- Some of the available open source static analysis tools include:
  - Cppcheck
    - Has a focus on detecting undefined behavior
  - Clang
    - Compiler that includes a static analyzer
  - Eclipse
    - IDE that includes static analyzers
      - including cppcheck with a plugin called cppcheclipse

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Static Analysis with Open Source Programs - 3

- We will use Cppcheck in the following example.
  - The developers of Cppcheck warn that it focuses on bugs over stylistic issues and tries to avoid false positives.
    - False positives are errors that are reported but are not actually errors on review.
  - This means it is likely to not report some of the more questionable bugs.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Static Analysis with Open Source Programs - 4

- The bugs that Cppcheck focuses on include:
  - Dead pointers
  - Division by zero
  - Integer overflows
  - Invalid bit shift commands
  - Invalid conversions
  - Memory management
  - Null pointer dereferences
  - Out of bounds checking
  - Invalid usage of STL
  - Uninitialized variables
  - Writing const data

Sponsored by National Security Agency Grant H98230-17-1-0344

Lab06 – Static Analysis of ec3.cpp

Sponsored by National Security Agency Grant H98230-17-1-0344

Lab06 Overview

- For Lab06, you will analyze sample code using two different compilers and an open source static analysis tool
- Detailed instructions for Lab06 are available within Canvas under the Modules tab

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Lab06 – ec3.cpp Source Code

```
void f(int a);
bool g(int& b);
int main(int argc, char* argv[])
{
   char a[10];
   // Array bounds error
   a[10] = 0;
   // Array bounds error via loop
   for(int k = 0; k <= 10; k++)
   {
      a[k] = k;
   }
   // Array bounds error via loop
   int m = 0;
   while (true)
   {
      a[m] = m; m++;
   }
   // Pointer variable p uninitialized
   int* p;
   *p = 5;
   // Null pointer dereference
   int* x = 0;
   *x = 1;
   // Trigger path in function g that does not return Boolean
   int c = -1;
   if ( g(c) ) c = 2;
   else c = 3;
   // Memory leak
   int* q = new int;
   // Memory leak
   int* r = new int[4];
   delete r;
   return 0;
}
```

```
// Unused function f
void f(int a)
{
   a = a + 1;
   // No side effect outside function
}
```

```
// Function g has one path that does not return a Boolean value
bool g(int& b)
{
   if (b > 0) return false;
   b = 0; // Exit without returning a Boolean
}
```

- The source code for ec3.cpp
- Static analysis does not require compilation

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Discussion - 1

- The static analysis of code is critical during code review to help ensure the proper operation of the program.
  - However, no single tool is guaranteed to find all bugs existing in a program.
  - Be prepared to use multiple tools to help find bugs and still need to perform additional testing to find them all.

Sponsored by National Security Agency Grant H98230-17-1-0344

Discussion - 2

- Strict adherence to coding standards can help avoid many bugs and errors.
- A lot of static analyzers focus on checking for compliance with these coding standards, while others focus on various types of bugs.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Discussion - 3

- False positives are very common with static analysis.
  - Many tools are considered ‘noisy’ and err on the side of caution when reporting errors.
  - It is up to the developer to sort through the reported errors and determine if they are valid or not.
- False negatives occur when a bug exists, but is not reported.
  - These are a more severe problem and more motivation behind using multiple tools for analysis.

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Summary

- Static analysis
  - Code review with access to source code that helps ensure it is conforming to coding standards
  - Tests the robustness of code to safeguard against attacks and failure
- Commercial static analysis tools
  - LDRA is one of many available to help automate code review and standards compliance
  - Help to prove to customers that product requirements have been met
- Open source static analysis tools
  - Free options for code review that are useful with larger projects with fewer of the bells and whistles

Sponsored by National Security Agency Grant H98230-17-1-0344

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE