17-654/17-754: Analysis of Software Artifacts
Jonathan Aldrich, Nels Beckman, Kevin Bierhoff
Assignment 5 (Tools): PREfast and SAL

Due Tuesday, February 27, 10:30am

Turn in a zip file named <username>-17654-A5.zip, where username is your Andrew id, electronically through Blackboard. The zip file should contain the following files:

- *Answers.txt*
- Your modified C++ file, *analysis_example.cpp*.

50 points

Assignment Objectives:

- Run a commercially important static analysis tool to find defects in C code that is synthetic but uses some realistic coding idioms.
- Gain an understanding of the benefits and drawbacks of automated static analysis.
- Explore the tradeoffs inherent in annotation-based analysis tools.

Tool Installation Instructions:

1. Install Microsoft Visual C++ 2005. You can get this as part of Microsoft Visual Studio 2005 through CMU's participation in the Microsoft Academic Alliance (contact Ed Walter). Or, download the express edition from http://msdn.microsoft.com/vstudio/express/visualc/

2. Download and install Microsoft Windows SDK for Vista from http://www.microsoft.com/downloads/details.aspx?familyid=c2b1e300-f358-4523-b479-f53d234cdccf

3. Unzip *asst5.zip* and open *hello.sln* it as a solution in Visual C++

4. Set up Visual C++ to use the Windows SDK compiler instead of the normal one:
   - In Tools | Options | Projects and Solutions | VC++ Directories add C:\Program Files\Microsoft SDKs\Windows\v6.0\VC\Bin (or similar) to the Executable files directory. Add C:\Program Files\Microsoft SDKs\Windows\v6.0\VC\Include (or similar) to the Include files directory.
   - In project Properties | Configuration Properties | C/C++ | Command Line add /analyze as an additional option

5. If your setup is correct, you should get 8 warnings, with types 4313, 3996, 6011, 6031, 6217, 6273, 6282, and 6386.

Run the Static Analysis on analysis_example.cpp:

1. Add annotations and/or fix defects in the code to eliminate the warnings that PREfast gives you.

2. Add annotations to express the sizes of all char * arguments and results in the program. If the analysis points you to a defect, fix it. As a reference for the annotations, see: http://msdn2.microsoft.com/en-us/library/ms235402(VS.80).aspx

3. **IMPORTANT: don't do this until the 2 tasks above are complete, as the annotations you will add here appear to mask some of the bugs you are trying to find above.** Add annotations to express taintedness. A string is tainted if it was read from an untrusted user, and has not been tested with a validation function.
   - You should annotate the first argument to read_input and the second argument to copy_data as [SA_Post(Tainted=SA_Yes)], meaning that the data left in the buffer after the call is tainted (here SA_Post means a postcondition).
   - You should annotate the first argument to copy_data as [SA_Pre(Tainted=SA_Yes)], as this function copies tainted data to the result (SA_Pre means precondition).
   - Calling system() on tainted data is dangerous because it potentially allows the user providing input to run any command line program they wish. So, annotate the output function as [SA_Pre(Tainted=SA_No)].
   - Note that validate is already annotated; do not change its annotation.
   - When you compile, you should see a warning about tainted data being passed to the output function. Fix this warning by adding a call to validate in the appropriate place.

Relevant resources you may need in the process above:

1. Michael Howard's blogs on getting started with Microsoft's Standard Annotation Language:
   - http://blogs.msdn.com/michael_howard/archive/2006/05/19/602077.aspx
   - http://blogs.msdn.com/michael_howard/archive/2006/05/23/604957.aspx

2. General documentation on Microsoft's Standard Annotation Language:
   - http://msdn2.microsoft.com/en-us/library/ms235402(VS.80).aspx

3. Documentation on printf: http://www.cplusplus.com/reference/clibrary/cstdio/printf.html

4. Documentation on the system() function (basically, this takes a command as a string and runs it in the shell): http://www.cplusplus.com/reference/clibrary/cstdlib/system.html

5. Documentation on gets, including relevant security issues and an appropriate workaround: http://www.codecogs.com/reference/stdio.h/fgets.php?alias=gets

6. Errors involving HRESULT and the FAILED macro:
   http://msdn2.microsoft.com/en-us/library/z5aa1ca1(VS.80).aspx

7. Errors involving = vs. ==:
   http://msdn2.microsoft.com/en-us/library/esa6csd7(VS.80).aspx

8. Information on the taintedness annotation in SAL:
   http://msdn2.microsoft.com/en-us/library/ms182047(vs.80).aspx

Questions (answer in Answers.txt):

1. Consider the code in wrap2 and wrap3, which is a variant of the example given in class. Is this code correct, and did PREfast find any issues with it? How does this affect your opinion of PREfast?

2. Are SAL annotations closer to Spec# annotations or to types? Compare them to each in terms of engineering costs and benefits.

3. Can you think of a way in which annotations might help make testing and/or inspection more effective?

4. Before you added ecount/bcount annotations, there were multiple bugs but PREfast reported no warnings. Discuss the benefits and drawbacks of this tool design.

5. Of all the types of errors that PREfast helped you find, which do you believe are the most important in a practical engineering setting and why?

Don't forget to also turn in your modified analysis_example.cpp file!