CS 4337.501

Homework 3

1. Use the Principle of Least Privilege and the principle Minimal Trusted Computing Base to describe how to make the following scenario more secure. What protocols can be put into place with regards to how information is handled and how departments interact with each other?
   1. The principle of least privilege can be applied to all the information about employees HR, Accounting, and Immediate supervisors have access to. For example, the HR department should only have information about an employee’s performance evaluation and formal reprimands because that is all they are evaluating the employee on for contract renewals. If they had access to other information about them, then it might cause biases in their evaluation even if they do not intend to. The principle of minimal trusted computing base can be used to ensure that only immediate supervisors are able to record information about the employees and no other people have access to edit the information. Because the immediate supervisors are the people who watch employees the closest, they are the most likely and most trusted to give correct assessments of an employee’s performance.
2. Compare and contrast Reference Monitors and Inline Reference Monitors. How are they implemented? What are the advantages and disadvantages? What are some motivations for using an inline reference monitor over reference monitors? Explain how the inline reference monitor is a language-based security technique.
   1. A reference monitor is a program that watches a program and makes sure it does not violate any of its policies. If it does, then the monitor crashes the program to ensure the program can’t do anything that the monitor doesn’t want it to do. An inline reference monitor means the monitor is written in the code you are trying to enforce policies on. While both are methods for enforcing policies on a program, there are advantages and disadvantages to both. A reference monitor can be used on a wider range of programs and resources and are usually more broad than inline. However, they can also cause performance issues because they are run separately from a program. Inline reference monitors have the advantage of being very specific to a program, potentially making them very secure and they don’t cause as many performance issues. A problem with inline is that if the checks are incorrectly done, it can cause vulnerabilities in a program so they must be very carefully implemented. Some motivations for using inline over reference monitors is if performance is a constraint or when specific policies need to be enforced within a program. Inline refence monitoring is a language-based security technique because it involves handling various security checks in order to make sure a program runs as intended.
3. Analyze the following fragment of C code. Is the code is well-typed? Is it is well-defined? Is it type-safe? Explain and support your answers.
   1. Yes, the code is well-typed and by definition is also well-defined. The program does not break the C type system and will compile and run successfully. It is well-defined because any program that is well-typed is also well-defined. The code is not type-safe because in the function f, takes in a pointer of type void but is given an input of a pointer of a pointer (the address of z which points to z which points to y). This function f sets the value the pointer points at to 0, setting the address of z to point to 0, or null, dereferencing z.
4. Analyze the following fragment of Java code. Is the code is well-typed? Is it is well-defined? Is it type-safe? Explain and support your answers. Assume doSomethingWithY is defined and makes a decision based on the value of y.
   1. This code fragment is not well-typed, well-defined, or type-safe. It is not well-typed because it is trying to assign a double to an int. When adding 1 and the double value 4.3, java implicitly converts 1 to a double, doing double addition on the operation which equals 5.3. The code tries to assign it to an int, which violates Java’s type system and all three definitions.
5. Analyze the following fragment of Python code. Is the code is well-typed? Is it is well-defined? Is it type-safe? Explain and support your answers.
   1. The code fragment is well-typed, well-defined, and type-safe. The code compiles and runs properly because Python allows for variables to change their type dynamically in runtime. The print function also can handle different variable types. This makes the program well-typed and thus well-defined. It is type-safe because there is nothing in the code fragments that could cause any type errors. The parameter val can take variables of any type and the print function can also handle variable types.
6. Consider your class schedule. You can view it as a string of some language. Describe this language. What is the alphabet? What strings are in the language?
   1. Given that the class schedule is made up of a class name, number, its time, days of the week, and classroom number, the language would be the set of class schedules. The alphabet would include the alphanumerical characters, so ‘A-z’ and ‘0-1’ along with symbols such as { “.”, “-“, “:”, “,”, “/”, “ “}. The letters would make up class names, for example “Programming Paradigms”, the number would be a concatenation of letters, numbers, and symbols – CS 4337.01, the time would be made up of numbers, symbols, and letters - 1:00-2:15 PM, etc. The strings would be the full class schedule for each unique class, for example a string in this language could be: Algorithms, CS 3345.01, 1:00-2:15, Tues/Thurs, Room SCI 2.201.
7. Consider the following grammar:
   1. Give a sentence in this language that is generated from at least 3 <prod>.
      1. myID -> a, &, % ENDLINE ID2 -> \*, b ENDLINE ID3 -> &, \*
   2. Show the left-most derivation for your sentence.
   3. Show the right-most derivation for your sentence.
   4. Show the parse tree for your sentence.
8. Consider the following grammar:
   1. Show that this grammar is ambiguous.
      1. This grammar is ambiguous because there are multiple parse trees for the same sentence. Taking the example sentence “ab”, which can be written with <S> -> <AB> -> “ab” and also <S> -> <As> <Bs> -> “a” <As> “b” -> “a”” b”.
   2. Rewrite the grammar to one that is unambiguous. The new grammar must generate the exact same language. HINT: This can be done by changing a single rule.
      1. Changing <S> -> <As> <Bs> | <AB> to: <S> -> <As> <Bs> | <As> <AB>. This works because <As> followed by <Bs> have higher precedence than <AB>, removing ambiguity from the language.
9. Create a grammar for the following language. (Note: We are ignoring whitespace and assume it is handled in the lexical analyzer stage.)
   1. S -> <tasks>

<tasks> -> <task> <tasks> | <task>

<task> -> <optional-int> <id> <optional-imp> <tags> ";"

<optional-int> -> <int>

<optional-imp> -> <imp>

<tags> -> <tag> <tags>

<tag> -> ID

<id> -> ID