

# CS 6V81-002: Quiz 11

March 3, 2008

Name: \_\_\_\_\_

1. Which of the following are true of well-typed Mobile programs? (circle all that apply)
  - (a) **They can be executed as .NET programs without modification.**
  - (b) Their non-security-relevant behavior is equivalent (up to stutter steps) to the original, unmodified code.
  - (c) **They always satisfy memory-safety and control-flow safety.**
  - (d) **They never commit a policy violation at runtime.**

*Mobile programs are just .NET programs with typing annotations. The typing annotations are implemented as .NET attributes (which are ignored by the .NET virtual machine) so they can be run without modification. The .NET type system enforces memory safety and control-flow safety and the Mobile type system enforces policy-adherence; however there is no guarantee that the rewritten code is behaviorally equivalent to the original.*

2. Which of the following are true of “packages” in Mobile? (circle all that apply)
  - (a) **Pointers to them may alias.**
  - (b) **Their methods are synchronized, preventing multiple threads from executing the same package’s methods concurrently.**
  - (c) **Packages may contain only unique pointers to security-relevant objects.**
  - (d) **Unpacking a package might throw an exception at runtime.**

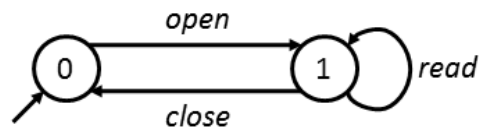
*Packages may alias arbitrarily but they wrap unique pointers to security-relevant classes. Their methods are implemented as atomic operations so that policies are enforced even for multithreaded code. Unpacking a package throws an exception at runtime if the package is empty.*

3. In the policy  $(\mathbf{0}(\mathbf{G} \cup \mathbf{G}^2 \cup \mathbf{G}^3)\mathbf{C})^\omega$ , the  $\omega$  is... (circle one)
  - (a) a typing variable for a polymorphic type
  - (b) a typing variable for a dependent type
  - (c) a runtime variable that gets injected into the untrusted code during rewriting
  - (d) **a regular expression operator denoting finite or infinite repetition**
4. *Prefix-adherence* is different from *policy-adherence* in that... (circle one)
  - (a) prefix-adherence is a property of heaps, whereas policy-adherence is a property of programs

- (b) prefix-adherence implies policy-adherence
- (c) .NET's type system guarantees prefix-adherence, but Mobile's type system guarantees policy-adherence
- (d) the proof of policy-adherence only applies to terminating programs**

*Policy-adherence and prefix-adherence are both defined in the paper as properties of heaps. They say, respectively, that all the objects in the heap satisfy the security policy, and that there exists a sequence of operations that can return every object to a policy-satisfying state. Thus, policy-adherence implies prefix-adherence but not vice versa, making (b) incorrect. The .NET type-system guarantees neither property, so (c) is incorrect. Mobile's type-system proves that if a program terminates normally then the resulting heap is policy-adherent, making (d) the correct answer.*

5. To implement security automata, Mobile policies define for every integer  $i$  the set of histories that cause an object to be in state  $i$ . For example, in the automaton



the set of histories that cause objects to be in state 0 is  $(open\ read^\omega\ close)^\omega$ . What set of histories cause objects to be in state 1?

*The set of histories that put objects in state 1 consist of those histories that put the object in state 0 followed by an open operation and any number of read operations. Thus, the correct set is:  $(open\ read^\omega\ close)^\omega\ open\ read^\omega$ .*