CIS4361 – Programming Assignment 2
Part A Virus Characteristics (total 50pts)
Part B Antivirus Characteristics (total 50pts)

Due on 4/20/2012
Group Work (individual groups)
Requires: **only** A **or** B (recommended)
Extra Credit: A and B

Part A:

As seen in class, viruses have 4 different stages: dormant, propagation, trigger, and payload. This assignment concentrates only on the execution stage and part of the trigger stage, and attempts to model a virus behavior considering the infection, trigger, and payload attributes. For simplicity, the assignment will be done in Java (some code will be provided). You will also need to download the tool Cavaj, a Java decompiler that will help with the infection characteristics. Inside of the Cavaj installation directory you will find a directory named jad, with a jad.exe command inside; such command may be used for decompiling a Java class file from command line. You will also be provided with a class file (Victim.class), with only a "Hello World" inside. Victim.class will be used as the victim file to attack. Note that if this was a real virus you only needed to select a file at random.

You are to complete the code provided to implement the following:

Infection and Propagation: this is not the core of this assignment; hence, a simple initial infection mechanism will be used. Initial infection will be made via USB drive with ***autorun*** feature. Autorun is a file in removable media, which runs automatically upon opening the drive. Newer versions of OS ask you if you want to run the autorun, I will say answer yes to that question to verify your project. Once the autorun runs it will simply use the current

user and copy the virus to its startup directory, and the Victim.class to the Desktop directory as an infected Victim.class and a backup of its previous state Victim.class.bak#, where # is the number of successful infections. This will be used to check for the hashes of each copy, and verify that it indeed all copies (with different hashes) perform the same action as the originals.

Payload: during infection the following payload behavior should be inserted into Victim.class. The virus-like program is to insert text into Victim.class, by decompiling with jad.exe (a Java command line decompiler), insert the text, and compile back the new code. You may need to add javac into your path variable, or compile the new code manually.

Every .class file in Java contains the definition of the class, which is **always** named after the file; thus, the file at some point should start with class Victim text. The line which comes after the first "{" that is encountered after Victim (e.g. public class Victim **{**) tells you where to start infecting the file (the next line after that one), since it would be the first line of the class definition. Similarly, you are to insert the payload required to mutate the file after main(String[] args) {.

You are to turn in:

- Your code (25 pts)
  - README file (5 pts)
  - Comments (5 pts)
  - Compiles/Runs (15 pts)
- A 2 page (minimum) report (25 pts)
  - what you learn (5 pts)
  - How Victim.class is changed with screenshots, garbage code generated, and examples of file hashes (15 pts)
  - how would you improve it (5 pts)

Submit your report in pdf format to hector.lugo@knights.ucf.edu

Part B

One of the most important characteristics of Anti-Viruses is their ability to check for integrity in files, to identify files which may contain virus signatures. For this assignment, you are to complete the code of a Java which checks for integrity of files. After you complete the code, copy and paste different files into the folder where your binary (IntegrityChecker.class) runs. Add the files to the IntegrityChecker, using the *add* command recently implemented. Run the *list* to verify that all the files are within the IntegrityChecker. Next make a modification to the file and use *check* command to see if the IntegrityChecker detects the illegal modification; use *update* command to make the modification legal from the IntegrityChecker point of view, and re-run *check* to verify that the update was done successfully.

I will prepare a windows version of cron, such that you can run the IntergityChecker, as if it where a windows scheduled task, using the format of cron (Linux).

At the end, you are to turn in:

- Your code (20 pts)
  - README file (5 pts)
  - Comments (5 pts)
  - Compiles/Runs (10 pts)
- A 2 page (minimum) report (20 pts)
  - what you learn (5 pts)
  - IntegrityChecker runs with screenshots, hashes generated, and examples of file modifications (10 pts)
  - how would you improve it (5 pts)
- A crontab file set to run IntegrityChecker each minute (10 pts)

Submit your report in pdf format to hector.lugo@knights.ucf.edu