Assignment 8, Fall 2015
CS 4630, Defense Against the Dark Arts
Detecting Virus Code Patterns

Purpose

This assignment will teach you to recognize virus code patterns unique to viruses using regular expressions. The regular expressions will be defined using flex, a modern version of the well-known utility lex.

Due Date

This assignment is due on Wednesday, 11/18/2015, at 10:00am

Virus Code Pattern

This assignment will focus on the recognition of the virus code that we just inserted into an executable —the "tricky jump." A typical example is:

```
push <virus_function_addr>
ret
```

We are not concerned with the particular function name. It would be unusual for a legitimate application to push an address immediately before returning, and most compilers have no occasion to generate this sequence of instructions. The presence of this sequence likely indicates that a virus has infected the executable.

Using flex

1. As initial test cases, use the infected files that you created in Assignments 4 and 5. An additional test case will be provided on the class Collab site (infected-target2.exe).

2. Because flex works on text files, we will need to write a simple filter that reads the binary file from standard input and writes a stream of ASCII bytes in hexadecimal to standard output. (Note: This type of "preprocessing" is a standard programming trick when you want to use a tool, but the tool doesn't quite handle the input that you have.) The hexadecimal bytes should be written in lowercase. Here is what filter produces for the first bytes of the file infected-target.exe when executed.

   ```
   $ ./filter <infected-target.exe
   7f454c4601010100000000000000000200030001000000030 .......
   ```

   Hint: You might find the default file pointer *stdio* useful. The program filter is small. Yous hould be able to implement it in fewer than 25 lines of C/C++ code.

3. Write a flex pattern file to scan the disassembled code and detect the virus code pattern discussed above. You should detect any tricky jump pattern regardless of the function address pushed by the first instruction. Keep track of the byte number in the input file as you process the file. When a virus is detected, print a warning message with the byte number and the name of the pattern. Start numbering at byte 0 and report the byte number of the first byte of the byte sequence that matches the pattern. After the warning message, print the bytes that matched the

pattern, followed by a blank line. Do not write anything else to the output stream. A sample run and output would be:

```
$ ./filter <infected-target.exe | ./scanner
WARNING! Tricky Jump: byte number: 2057
68f0144100c3
```

Note: The above output is for illustrative purposes. It is not accurate with regard to the line number or the matching code that is output. However, you output format should be EXACTLY the same as the output in this example.

It may be that a file contains multiple instances of a tricky jump. Your scanner should detect all instances of a tricky jump (i.e., do not terminate the scan after reporting the first occurence of a tricky jump).

4. Call your flex file virus_patterns.l. Compile and link the flex code into an executable. To jump start you, here are the commands necessary to create your scanner:

```
$ flex virus_patterns.l
$ gcc -o scanner lex.yy.c -lfl
```

The -lfl option to gcc tells the linker to search the flex library for any support routines necessary.

5. Run and test your solution. You might want to create some additional input files that contain sequences that the pattern should match as well as some sequences that are close but that your scanner should not match.

6. Note: we will be testing your code on a variety of inputs to see if you get false positives or false negatives. We will extend our pattern file to detect other viruses in a following assignment.

Submission Guideline

1. **WARNING: YOUR SUBMISSION ***MUST*** FOLLOW THIS GUIDELINE. THERE WILL BE 40% PENALTY FOR THOSE WHO FAIL TO FOLLOW THIS GUIDELINE.**
2. Submit your filter.c and *virus_patterns.l* to collab. **THE FILENAME MUST BE filter.c and virus_patterns.l .**
3. Your program will be compiled with the following command:

```
$ flex virus_patterns.l
$ gcc -o scanner lex.yy.c -lfl
```

You will receive 0 points if your program fails to compile with this command. Warning messages from the compilers are OK.

4. **Your filter program MUST read from stdio and output to stdout.**