**Question 1.** The following code snippet is described in Peter Szor’s “Attacks on Win32 II”. What does it do and why?

**Address Opcode Mnemonic**

**0040601A E800000000 call 0040601F**

**0040601F 5E pop si**

**Question 2.** (Follow-up to Question 1). Szor claims in the discussion of the code from Question 1 that “[s]ince, in normal circumstances, code similar to the above is not generated by compilers, the use of E800000000 opcode is a suspicious activity.” Explain what he means by this statement.

**Question 3.** The following code snippet also occurs in Peter Szor’s “Attacks on Win32 II”. What does it do and why?

**0040601A E807000000 call 00406026h**

**0040601F 34F4 xor al,F4**

**00406021 F0A4 lock movsb**

**00406023 288C085EB934AC sub [eax+ecx-53CB46A2],cl**

**0040602A 0200 add al,[eax]**

**Question 4.** (Follow-up to Question 3.). Szor claims that “Dynamic heuristics are necessary to see if the CALL instruction points to an actual POP.” Explain what a dynamic heuristic is. Why is a dynamic heuristic necessary?

**Question 5.** An armored virus may try to make the disassembly of the viral code difficult. Describe a technique that virus writers use to make code difficult to disassemble and explain how it makes the code difficult to disassemble.

**Question 6.** Virus researchers often use emulators to “watch” virus code execute. One technique that virus writers use to thwart anti-virus emulators is to use special processor instructions such as MMX (multi-media extension) or SSE (SIMD extensions). Explain how the use of special processor instructions can thwart emulation-based virus detection techniques.

**Question 7. Short definitions: Define each of the following in 1-3 sentences.**

1. **Oligomorphic virus.**
2. **Retrovirus.**
3. **Dynamic Malware Analysis.**
4. **Code Integration.**
5. **Multipartite virus.**

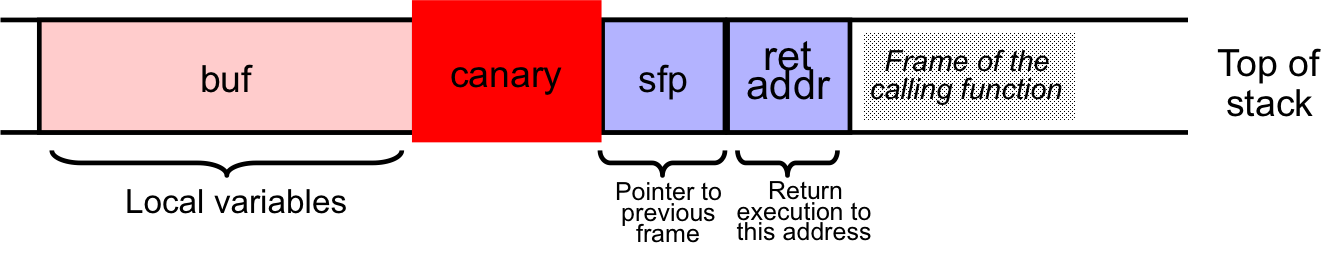
**Question 8.** Tunneling viruses modify a key operating system data structure. What data structure is modified?

**Question 9.** We read two papers this semester that relied heavily on the concept of “Turing completeness”; these were:

* “**mov** *is Turing-complete*”, by Stephen Dolan, and
* “*The Geometry of Innocent Flesh on the Bone : Return-into-libc without Function Calls (on the x86)*” by Hovav Schacham.

Explain (briefly) what a Turing complete language is. Also explain its significance with respect to return-oriented programming (i.e., why should we care that ROP is a Turing complete way to implement programs?).

**Question 10.** In class, we briefly discussed the use of “canaries”. They are described in Wikipedia as “*Canaries or canary words are known values that are placed between a buffer and control data on the stack to monitor buffer overflows. When the buffer overflows, the first data to be corrupted will usually be the canary, and a failed verification of the canary data is therefore an alert of an overflow, which can then be handled, for example, by invalidating the corrupted data.*” Here is an illustration of a canary.



Describe how a canary is used to monitor buffer overflows (hint: consider what happens when ‘buf’ is overflowed). What kinds of changes are necessary to language compilers to use canaries?