IERG4130: Introduction to Cyber Security **Fall 2017**

Buffer Overflow Solution

Question: Buffer Overflow

(a) Client-side validation, like HTML input attribute in this particular example, should never be trusted, because they can be easily bypassed. An attack can simply increase the value or even delete the attribute completely (try it out in your browser!) or just tailor-make an HTTP packet. In this way, the attacker can place arbitrary length string as input.

(b) The code is still vulnerable to buffer overflow attacks. The reason is as follows: Each `memcpy()` writes 8 bytes (because of `sizeof(unit64_t)`), which will write past the end of `buf` if `i` is 11 or larger. The call to `memcpy()` overwrites `buf[i..i+7]`, and if `i+7` is 18 or larger, this writes outside the bounds of `buf`. As a result, as long as `length` is 12 or larager, buffer overflow occurs.

(c) There are many mitigations. Here we only list three methods as examples.

- Non-executable memory: marking of memory regions as non-executable, such that an attempt to execute machine code in these regions will cause an exception.
- Address Space Layout Randomization (ASLR): randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries, making exploitation of a buffer overflow more difficult.
- Stack canary: embed "canaries" in stack frames and verify their integrity prior to function return. A failed verification of the canary data terminates the execution of the affected program.