Exercise 5: Program Vulnerabilities. For the following code, assume an attacker can control the value of basket passed into search_basket. The value of n is constrained to correctly reflect the number of dogs in the basket. The code includes several security vulnerabilities.

Some reminders:

- **snprintf(buf, len, fmt, ...) works like printf, but instead writes to buf, and wont write more than len - 1 characters. It terminates the characters written with a '0'.**
- **System runs the shell command given by its first argument.**

```
struct dog {
   char name [1024];
   int age;
};

/*
Searches through a basket of dogs of size at most 32.
Returns the number of puppies or -1 if there are no puppies .
*/

int search_basket (struct dog basket[], size_t n) {
   struct dog puppies [32];
   char puppy_names [1024], cmd [1024];
   int i, total_puppies = 0, names_size = 0;

   if (n > 32) return -1;
   for (i = 0; i <= n; i++) {
      if (basket[i].age < 12) {                                  1024
         size_t len = strlen (basket[i].name);
         snprintf(puppy_names + names_size, len , "%s", basket[i].name)
         puppies[total_puppies] = basket[i];
         names_size += len;   1024
         total_puppies += 1;
      }
   }

   if (total_puppies > 5) {
      const char *fmt= " adopt - puppies -- num_puppies %d --names %s";
      snprintf(cmd, sizeof cmd, fmt, total_puppies, puppy_names);
      system(cmd);
   }
   return total_puppies;
}
```

(a) Circle three such vulnerabilities in the code and briefly explain each of the three.

Vulnerability I: dog age integer overflow – although this isn't a huge vulnerability, this demonstrates how the attacker can bypass the age check for each dog inside of the search_basket function by passing in a value that is greater than the max integer size, given that a signed integer is used to denote the dog's age. In this case, the integer overflow can cause the value contained in the age variable to ultimately be less than 12, giving the attacker access to write to the buffer within this block of code.

Vulnerability II: puppy_names buffer overflow (smash attack) – the name of each dog can be up to 1024 chars, while the buffer allocated to store each puppy name is also 1024 chars. Since there is no check to determine how much has already been written to the buffer, the buffer will overflow when the names_size counter has surpassed the size limit of the buffer. Consequently, the attacker can modify the contents of the stack beyond the buffer itself and manipulate variables such as the return address to intentionally point it towards planted malicious code.

Vulnerability III: command injection – puppy_names is a buffer that stores the string values provided to the name of each dog in the basket. It is added to the end of a formatted string that is passed into a system call – this can easily be exploited as it is possible to execute multiple commands in a single line. We can use an ampersand (&) in Windows, or a semicolon (;) in Linux between two commands in the same line to do exactly that. Not only that, but the actual command performed by this function can fail and the injected command will still be executed. The attacker can use this to perform a plethora of malicious attacks, as they effectively have access to the command line of the computer upon which this code is being executed.

(b) Describe how an attacker could exploit these vulnerabilities to obtain a shell.

See Vulnerability III from part (a). By doing this, the attacker can use something like the wget command on Linux to download and execute a script from a malicious source on the web, all in a single line.