# CSCD 437
# Lab 4
# Code Vulnerabilities

## SPECIFICATIONS

This assignment is to critically examine some C code for flaws that could lead to buffer overrun or even other exploits. Your job is to identify the vulnerabilities then provide fixes.

Sunny Sunheim likes writing C code. That's because when Sunny writes it, the sun is always shining. If the code compiles, Sunny won the battle and it's sunny. If the program runs properly with input, Sunny won the war and it's sunny *and* warm. Sunny smiles broadly when the war is won. After winning the war, Sunny goes out into the sun for some serious R and R. Your job is to rain on Sunny's 'shining' C code and identify how it can be exploited (or that it can't be exploited -- maybe Sunny got lucky). Show that Sunny spends too much time in the sun!

For each of the following letters, identify problems/vulnerabilities and suggest how to avoid them (first) inside the function itself or (second - assuming first is not possible) how you could modify the parameters to the function to help. Since you have the code, compile and run to confirm any suspicions. All problems assume a command line argument is specified when the program is run.

Submit a PDF that contains your identified vulnerabilities and **specific** code fixes. More specifically, clearly identify the code that is vulnerable, including a sample run showing it is vulnerable, and then provide specific code fixes, including sample runs, that illustrate the vulnerability/vulnerabilities is/are fixed.

NOTE: You must use the gcc compiler for this lab, preferably from Ubuntu.

**a.**

```
int foo(char *arg, char *out)
{
  strcpy(out, arg);
  return 0;
}

int main(int argc, char *argv[])
{
  char buf[64];
  if (argc != 2)
  {
    fprintf(stderr, "a: argc != 2\n");
    exit(EXIT_FAILURE);
  }// end if

  foo(argv[1], buf);

  return 0;

}// end main
```

**b.**

```
int foo(char *arg)
{

  char buf[128];

  int len, i;



  len = strlen(arg);

  if (len > 136)

    len = 136;



  for (i = 0; i <= len; i++)

    buf[i] = arg[i];

  return 0;

}

int main(int argc, char *argv[])

{

  if (argc != 2)

  {

    fprintf(stderr, "b: argc != 2\n");

    exit(EXIT_FAILURE);

  }

  foo(argv[1]);

  return 0;

}// end main
```

**c.**

```
int bar(char *arg, char *targ, int ltarg)
{

  int len, i;

  len = strlen(arg);

  if (len > ltarg)

    len = ltarg;

  for (i = 0; i <= len; i++)

    targ[i] = arg[i];

  return 0;

}

int foo(char *arg)
{
  char buf[128];

  bar(arg, buf, 140);

  return 0;

}

int main(int argc, char *argv[])

{

  if (argc != 2)

  {

    fprintf(stderr, "c: argc != 2\n");

    exit(EXIT_FAILURE);

  }

  foo(argv[1]);

  return 0;

}// end main
```

**d.**

```
int foo(char *arg, short arglen)

{

  char buf[1024];

  int i, maxlen = 1024;

  if (arglen < maxlen)

  {

    for (i = 0; i < strlen(arg); i++)

      buf[i] = arg[i];

  }

  return 0;

}

int main(int argc, char *argv[])

{

  if (argc != 2)

  {

    fprintf(stderr, "d: argc != 2\n");

    exit(EXIT_FAILURE);

  }

  foo(argv[1], strlen(argv[1]));

  return 0;

}
```

## WHAT TO TURN IN:

Your group will submit a single PDF containing in this order:

- The names of each teammate in alphabetical order by last name.
- The letter and the original code clearly noting where the vulnerability is and what the vulnerability is.

  Example strcpy(buf, input); <- strcpy does not check if input is larger than buf, and it writes the \0 no matter what the size of input and buf are.

- Output capture(s) that show you exploited the vulnerability.
- The coded fix to fix the vulnerability with explanation. NOTE: just changing the solution in the example above from strcpy to strncpy does not illustrate it is fixed. You need an explanation about why you think that solves the problem, and possibly a comment about the vulnerabilities from the solution.

  Example: strncpy can solve this problem; however, its vulnerability is it does not write the \0 so the string may not be terminated.

- Output capture(s) that show you the vulnerability is fixed.

NOTE: this is much more involved than just changing a few lines of code. You need to clearly address the original vulnerabilities and also address added vulnerabilities by your fixes.

Name your PDF your team number lab4.pdf (Example: team16lab4.pdf)