Secure Coding Principles **Name________________________**
Midterm **100 points**
Due Tuesday, February 20 by 8am on Blackboard
Place work in a zip/rar and name as follows: lastNamefirstNamemidterm.zip

(18 points - 3 points per problem) Sunny Sunheim likes writing C code. That's because when he writes it, the sun is always shining. If his code compiles, he won the battle and it's sunny. If his program runs properly with his input, he won the war and it's sunny *and* warm. Sunny smiles broadly when he wins the battle and the war, and then goes out into the sun for some serious R and R. For problem 1, 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 (the sun was shining on him)). Show Sunny that he 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, you might try and compile and run to confirm any suspicions…All problems assume a command line argument is specified when the program is run.

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);
    }
  foo(argv[1], buf);
  return 0;
}
```

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;
}
```

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;
}
```

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;
}
```

e.

```
FILE * foo(char *arg)
{

  return fopen(arg, "w");
}

int main(int argc, char *argv[])
{
  FILE *fp;
  if (argc != 2)
    {
      fprintf(stderr, "e: argc != 2\n");
      exit(EXIT_FAILURE);
    }
  fp = foo(argv[1]);
  fprintf(fp, "Files are a piece of cake in C\n");
  fclose(fp);

  return 0;
}
```

f.

```
long foo(int one, int two)
{

  return long(one) * two;
}

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

  if (argc != 3)
    {
      fprintf(stderr, "f: argc != 2\n");
      exit(EXIT_FAILURE);
    }

  printf("%d", foo(atoi(argv[1]), atoi(argv[2]));

  _exit(0);
  /* not reached */
  return 0;
}
```

1. (10 points) You have a program that requires the user to enter a password. Describe as many things as you can (you should come up with at least 5) to help avoid having this password discovered by an attacker.

2. (15 points) Read the article "Smashing the Stack for Fun and Profit, by Aleph One." Answer the following *based on discussion in the article*.
    3. (3 points) What is smashing the stack?
    4. (2 points) What routines in C provide the ability to implement this attack?
    5. (10 points) Describe specifically what must be done in memory to execute malicious code.

6. (20 points) Describe completely each of the threats represented in STRIDE. Include a real world example of each type of threat *and* what tool(s) can be used to mitigate the threat.

7. (10 points) Design a regular expression to handle/validate each of the following (specify which language you are using)
    8. (5 points) a 64 bit real number -- it can be positive or negative, it may or may not have a decimal point, it will not be in scientific notation (meaning it will contain only digits), it will not contain commas
    9. (5 points) a legal variable name in Java

10. (25 points) Code Monkeys Inc. has hired you, a security 'expert', to provide security recommendations for a payroll system they have just won a bid for from a local credit union. List **ten** things they should (a) be aware of or (b) should do or (c) should not do when building the software. Provide a brief explanation with each thing you list that justifies your recommendation. Include suggestions on how to mitigate possible threats you identify. NOTE: The folks at Code Monkeys Inc. understand programming languages, databases, and networking and protocols, but don't really know the security issues involved with them (they were never taught that stuff in school!).