Intro to payloads

The goal of this assignment is to gain familiarity with the environment in which stage 0 payloads run by writing a basic payload that will be executed within the memory space of some process.

Your assignment is to craft a 64-bit Linux x86 assembly language program that implements the algorithm described in Listing1 below.

Your payload may not invoke any library functions from any shared libraries. You are permitted to make system calls. You may also decompose the code below into smaller functional units containing helper functions if you feel that will help organize your code. NOTE that the following code is not intended to compile or run as is, it is merely a description of the algorithm that you are expected to implement. Execution is to begin at _start

```
void _start() {
   int in_pipe[2];
   int out_pipe[2];
   pipe(in_pipe);
   pipe(out_pipe);
   if (fork()) {
      close(in_pipe[0]);
      close(out_pipe[0]);
      close(out_pipe[1]);
      encode(0, in_pipe[1]);
   }
   else if (fork()) {
      close(out_pipe[1]);
      close(in_pipe[0]);
      close(in_pipe[1]);
      encode(out_pipe[0], 2);
   }
   else {
      close(in_pipe[1]);
      close(out_pipe[0]);
      dup2(in_pipe[0], 0);
      dup2(out_pipe[1], 1);
      dup2(out_pipe[1], 2);
      execve("/bin/sh", {"/bin/sh", NULL}, {NULL});
   }
}

def encode(fd_in, fd_out):
   k = 0xc9
   while True:
       val = fd_in.read(1)
       if not val:
           break
       val = chr(ord(val) ^ k);
       k += 1
       k = k & 0xff
       fd_out.write(val)
```

Listing 1 – Pseudocode description of required payload

Your payload will be tested by assembling it with nasm using "-f bin", a program will then load your payload into a random location in its memory space and transfer control to it. No parameters will be passed into your payload.

You must also create a second program, written in C whose purpose is to load the binary machine code blob of the first program into memory and then transfer control to the newly loaded bytes. This program will receive two command line arguments. The first argument will be either "-h" or "-s" indicating that you are required to load and execute the payload in either the heap or stack respectively. The second argument will be the name of the assembled binary blob file. The purpose of this program is to help you understand how your payload will be graded and to become familiar with the idea of running payloads within an existing process. The best way for you to proceed is probably to write a standalone assembly language program that performs the required task that makes no use of any fixed addresses (no .data or .bss). Once you have a working program, create a second program using C to read your payload from disk and execute it. There are any number of ways to transfer control to arbitrary locations in memory from a C program. If you are not familiar with them, you may want to look into function pointers. The following represents basic build and use of the two programs:

```
# nasm -f bin assn1.asm
# gcc -o assn1_harness assn1_harness.c
# ./assn1_harness –s assn1
<program interaction happens here>
#
```

Deliverables

1. You must turn in two source files, the source for your assembly language payload which **MUST be named exactly** assn1.asm, and the source for the testing harness that you create which **MUST be named exactly** assn1_harness.c
2. Create and turn in a third file named **exactly** readme.txt in which you describe what the code in Listing 1 is doing and why it might be useful.
3. Upload your three files, packaged in a zip file named **exactly** assn1.zip to Sakai NLT 1700 on the due date.

Notes

1. Depending on your Linux setup and how you write your test harness, you may need to make use of the execstack (research it) program to make the stack and heap executable.
2. When debugging, it is useful to have your program load at the same address every time you run it, so you may also wish to play around with the kernel variable: kernel.randomize_va_space which is available on linux systems and which, when set to zero, disables address space layout randomization. Research the sysctl command to learn how to manipulate such variables.