A Tale of Two Clients

Plot summary. In which a client Chuck (C) is having a pleasant exchange of requests and responses with server Sam (S), when Mary (M) suddenly appears and attacks Sam.

The scenario unfolds exactly as in the lab/project p2. Sam's server runs the "BuggyServer" code you studied for p2, ready to receive connections on port 9000. All processes involved are single-threaded. All participants run "classic Unix". All processes use the various Unix system calls as discussed in class.

The scenarios involve two simple and familiar Unix utility programs. The echo program simply writes its arguments to stdout. The nc program is netcat, which opens a socket and then uses read and write system calls to transfer data from its stdin to the socket and from the socket to its stdout.

Instructions. Answer all questions. Please attempt to confine your answers to the boxes provided. The small boxes are worth 10 points. The large boxes are worth 20 points.

Many questions on this exam ask you to explain process states of Chuck, Mary, and Sam at various stages in the narrative. For those questions, the state of a process is the state of its thread: either Running, Exited, or Blocked. If a process is Blocked, name the system call in which it is blocked and the object it is blocked on. Please assume that each process has had sufficient time to execute to reach a stable state.

As always: "Any kind of pseudocode is fine as long as its meaning is clear. You may assume standard data structures, e.g., linked lists: don't write code for those." If you don't know the answer to a question, then just say something else relevant for partial credit.

There are no "April Fools" jokes on this exam, with the possible exception of this statement.
Part 1. In which Chuck sends a request. (50 points)

Chuck types the following command to his shell to issue a valid HTTP request to Sam:

```
/bin/echo "GET /file HTTP" | nc sam 9000
```

(a) What is the state of Sam’s server process before the request is sent?

(b) Which of Chuck’s processes creates a pipe? How does another process gain access to the pipe?

(c) What is the state of each of Chuck’s processes after the request is sent, but before Sam’s reply is received?

(d) List the key system calls that the nc process issues before arriving at the state in (c), in the order it issues them.

(e) What causes Chuck’s nc process to (eventually) exit?
Part 2. In which Mary attacks Sam. (60 points)

Mary types the following command to her shell to issue a valid HTTP request to Sam. The request contains a malicious attack string with an embedded shellcode to inject into Sam's server. Sam's server receives the attack string and copies it into a local buffer variable of its handle() procedure, overflowing the buffer.

```
/bin/echo -e "GET <attack string> HTTP" | nc sam 9000
```

(a) Why is it necessary for the copy to overflow the buffer in order for the attack to be successful?

(b) Mary must guess the location of the buffer in the server's virtual address space, and include a stack address within the attack string. Why is this necessary in order for the attack to be successful?

(c) Draw a picture of Sam's server stack after the handle() procedure is called but before the copy operation occurs. Be sure to illustrate the key elements that are relevant to the attack.

(d) Draw a picture of Sam's server stack after the copy operation but before handle() returns. Be sure to show the key regions of the attack string and their locations relative to the elements in (c).
Part 3. In which Sam executes Mary's shellcode. (40 points)

The shellcode is a sequence of machine instructions with the usual behavior: create a fresh socket, bind the socket to a port number (the shell port), listen/accept a connect request from Mary on the shell port, and use the execve system call to run a shell program that receives commands from Mary and sends their output back to her.

(a) One possible outcome is that the server process generates a fault when it executes its first instruction from Mary's attack string. (Let's say the instruction is a NOP.) What is the most likely cause of the fault?

(b) If the attack succeeds in running a shell, what user identity does the attack shell run with? How does Sam's kernel determine what user identity to place in the security label of the shell's process?

(c) The nc process that Mary used to launch the attack (in Part 2) does not exit after the attack. Why not? How is it different from Chuck's nc process in Part 1?

(d) What is the state of Sam's server process after Mary connects to the shell port, but before she sends a shell command?
Part 4. In which Chuck is out of luck. (30 points)

Chuck repeats the command of Part 1 after Mary initiates the attack, but before Mary disconnects from her attack shell.

```
/bin/echo "GET /file HTTP" | nc sam 9000
```

Chuck types the command, hits the return key, and....nothing happens.

(a) What is the state of Sam's server process at this point?

(b) What is the state of each of Chuck's processes at this point?

(c) What is the state of Sam's server process after Mary disconnects from the attack shell by exiting her client program?
Part 5. In which Mary gets smarter and avoids detection. (20 points)

One weakness of Mary's shellcode is that Chuck might report his problem to Sam, who might then discover Mary's attack.

If Mary is busted, she could forfeit her property and be imprisoned for 20 years under US Federal Penal Code 18 §1030 (CFAA, the Computer Fraud and Abuse Act), and regret this sorry episode for the rest of her miserable life.

Propose a solution for Mary. Of course, Mary should not mount the attack at all, but if she is committed to this path of darkness, how could she modify her shellcode to hide her attack from Chuck? How could she avoid disrupting the functioning of Sam's webserver? Discuss any practical difficulties of your idea and any assumptions needed to overcome them.