

Buffer overflow string:

AA\x18\x0e\x40\x00

Explanation of how to find string:

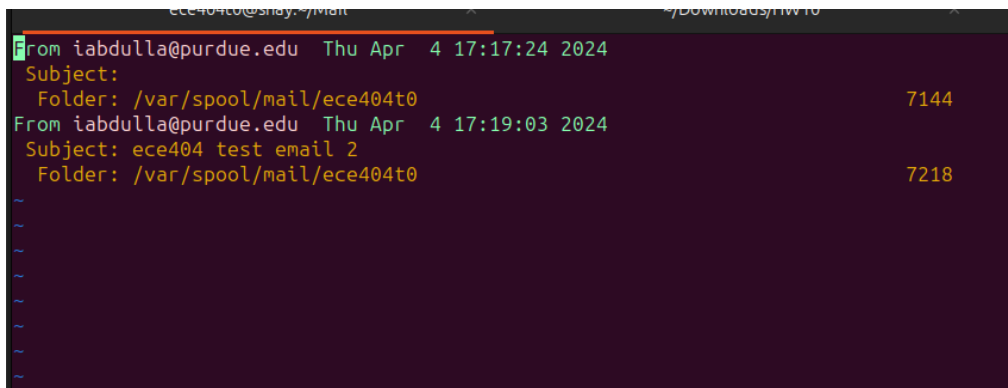
To orchestrate a buffer overflow that triggers a secretFunction, the exact payload string “AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x18\x0e\x40\x00” was crafted. This process began with a code review to locate the vulnerable buffer and the secretFunction within the server's code, identifying the latter's memory address using GDB. By determining the size of the buffer and the offset to the saved return address on the stack, a payload was constructed. This payload consisted of a series of "A" characters sufficient to fill the buffer completely, immediately followed by the little-endian representation of the secretFunction's address. When the client sent this string to the server, it filled the buffer and overwrote the return address on the stack with that of secretFunction. Upon the function's return, instead of following the original execution path, the program counter redirected to secretFunction, demonstrating a successful buffer overflow exploit.

Explanation of how to fix servers.c:

To mitigate buffer overflow vulnerabilities in `server.c`, I replaced the `strcpy()` function with `strncpy()` to prevent copying more data than the destination buffer can accommodate. The `strncpy()` function is used with explicit length checks, ensuring that the number of bytes copied does not exceed the buffer size. Additionally, I manually added null terminators after using `strncpy()` to ensure that strings are properly terminated, as `strncpy()` does not guarantee null termination if the source is longer than the specified length. The dynamic allocation of the receive buffer (`recvBuff`) is now followed by a boundary check against `MAX_DATA_SIZE` before echoing back to the client. These measures, along with freeing the dynamically allocated memory to avoid memory leaks and setting the return type to `void` since the allocated buffer is freed within the function, substantially increase the robustness of `server.c` against buffer overflows.

```
if (numBytes < MAX_DATA_SIZE) {
    strncpy(str, recvBuff, numBytes);
    str[numBytes] = '\0'; // Null-terminate the string
} else {
    strncpy(str, recvBuff, MAX_DATA_SIZE - 1);
    str[MAX_DATA_SIZE - 1] = '\0'; // Ensure string is null-terminated
}
```

Part 2:



The screenshot shows a terminal window with a dark background and light-colored text. The text displays email log entries. The first entry shows an email from iabdulla@purdue.edu received on Thursday, April 4, 2024, at 17:17:24. The subject is blank, and it is stored in the folder /var/spool/mail/ece404t0 with a size of 7144. The second entry shows an email from the same sender received at 17:19:03, with the subject 'ece404 test email 2', also in the folder /var/spool/mail/ece404t0, with a size of 7218. Below these entries are several tilde (~) characters, likely representing the continuation of the log file.

```
ece404t0@shay:~/mail  x  ~/Downloads/HW10  x
From iabdulla@purdue.edu  Thu Apr  4 17:17:24 2024
Subject:
Folder: /var/spool/mail/ece404t0                7144
From iabdulla@purdue.edu  Thu Apr  4 17:19:03 2024
Subject: ece404 test email 2
Folder: /var/spool/mail/ece404t0                7218
~
~
~
~
~
~
```

Screenshot of logfile contents.