Hacking in C

Assignment 5, Thursday, Februari 28, 2018

Handing in your answers: Submission via Brightspace (http://brightspace.ru.nl)

Deadline: **Wednesday, March 14**, 23:59:59 (midnight)

1. In this assignment you will perform a remote exploit on an echo server we provide. The echo server echoes input, and as attackers you can repeatedly supply format strings or long inputs to reveal or corrupt data on the stack. Please be considerate of your fellow students: If you intentionally break the server, they cannot complete the exercise until we fix it.

   Remember that these exercises do not count to your final grade. However, you should make an effort to at least partially answer all questions If you get stuck on a question, clearly explain why you cannot continue.

   The hostname of the echo server is `hackme.cs.ru.nl`, the port is `2266`. If you connect to this address with e.g. `netcat` aka `nc` (`nc hackme.cs.ru.nl 2266`, see also `man nc`) it will simply print everything you send back to you.

   ```
   ~$ echo "Hello there!" | nc hackme.cs.ru.nl  2266
   Hello there!
   ```

   The command `netcat` takes input from files, commands or scripts if you simply redirect the standard input, and send them over a network connection. For example, `nc hackme.cs.ru.nl 2266 <file` will establish a connection to the echo server and send the contents of `file` to it. As another example, `./exploit.sh | nc hackme.cs.ru.nl 2266` will send the output of the script `exploit.sh` to the echo server. Since you are not giving the input on the command line, all you see is what the server echoes back.

   For the purpose of this exercise, we went out of our way to do this insecurely. We have manually disabled several security mechanisms on the program you connect with. There is no reason to ever do this on modern systems, so you are unlikely to encounter programs in the wild which are exploited as easily as this one.

   For this exercise you will need to figure out what the program does *without access to the program*. The source code or binary executable are not provided. Rather, you must use the knowledge gained from the previous lectures and exercises to figure out how to exploit this program with the shellcode we provide.

   Preparation

   (a) Download and extract the assignment code package from https://rded.nl/hic/assignment5.tar.gz.

   (b) Read the source code of `shellcode.c` to learn what `shellcode.c` does and compile it using `make`.

   (c) You will also find a helper program `reverseaddr` that helps with reversing an address and printing it in little-endian byte form.

   Note: you do not need to use either program (you may write your own), but they will probably be quite helpful.

   Gathering intelligence

   (a) Create a file called `exercise5` to do your notetaking in. This will be one of the files you will hand in.

   (b) There is a buffer overflow vulnerability in the program. Figure out how many bytes you need to send before the program crashes. Why is the program likely to be crashing when you send that many bytes? Write your answer down.
   Hint: It is probably easiest to write a shell script that prints however many characters as you want. The code package contains `attack.sh` which you could use as a starting point.

(c) There is also a vulnerable `printf` statement. Use this to read the stack. Add the format string you've used and the memory contents you obtained to `exercise5`. Try to identify certain items in the output, think of:

- Stack addresses
- Return addresses
- The frame pointer
- The contents of your buffer

Add your guesses to your notes.
Hint 1: It is probably again easiest if you add some functionality to your attack script to print your format string.
Hint 2: Remember that addresses pointing to values on the stack usually start with `0x7f`.

(d) If you've found the frame pointer, you can try to estimate where the start of the vulnerable buffer may be. Write down what you think, and add an explanation.

(e) Recall the layout of the stack: there may be other local variables in the memory output between the buffer and the return address. You may be able to guess what sort of values are there. Write down your thoughts.

Exploitation You should now have a *rough* idea of the following:

- The (relative) location of the return address on the stack
- Some idea of where the start of the buffer is

If you're unsure, it is prefectly valid to apply some amount of brute force to figure out the exact addresses; adding a for loop to your script to try out some values may be very useful.

(a) Combine all the information you've obtained to obtain a shell on the server.

i. Add the shellcode to your attack script. The provided `shellcode` program should be helpful. It can also print however many NOPs you want.
Hint 1: Start out with `./shellcode list <nops>`. This shell code will execute `/bin/ls`, which immediately gives you output.
Hint 2: to add the output of `./shellcode list <nops>` to your attack string in `attack.sh`, write `attackstr="${attackstr}$(./shellcode list <nops>)"`.

ii. Add the address you want to overwrite the return address with to your attack script.
Hint 1: Don't forget that addresses should be supplied little-endian!
Hint 2: You may use the supplied `reverseaddr` program to help you reverse and print the bytes of an address. If it seems you don't get visible output when run by itself, that may be because the characters printed are not valid ASCII. You could check that by doing `./reverseaddr 0xAAAAAA | xxd` to have `xxd` print the output in hexadecimal.

iii. Figure out how many bytes you should print to overwrite the return address and execute your shell code. If your script by this point expects any arguments, add the correct way to invoke it to your notes.

iv. Once you have obtained the `/bin/ls` output, switch the `./shellcode list` command over to the `./shellcode shell` command. **Also**, switch the command that executes the attack (the ones with `echo $attackstr`) to the one that says "uncomment if you want to run the shell code". If your script by this point expects any arguments, add the correct way to invoke it to your notes.
Hint: If it appears the command hangs, try typing `whoami`. You will not get the layout of a normal command prompt.

v. Once you've obtained a shell, run the command `proof`, and follow the instructions on-screen. Do not forget to send the e-mail.

vi. Please do not try to further exploit the machine. Do not run anything that may hinder other students.

2. Place the file with your notes and everything you used to exploit the machine in a folder called `hic-assignment5-STUDENTNUMBER1-STUDENTNUMBER2`. Crucially, include your attack script. Do not forget to add any instructions on how to run your script. Again, add in your student numbers and create a tar archive of the folder. Hand it in on Brightspace.