

# **Ghidra SRE Challenge**

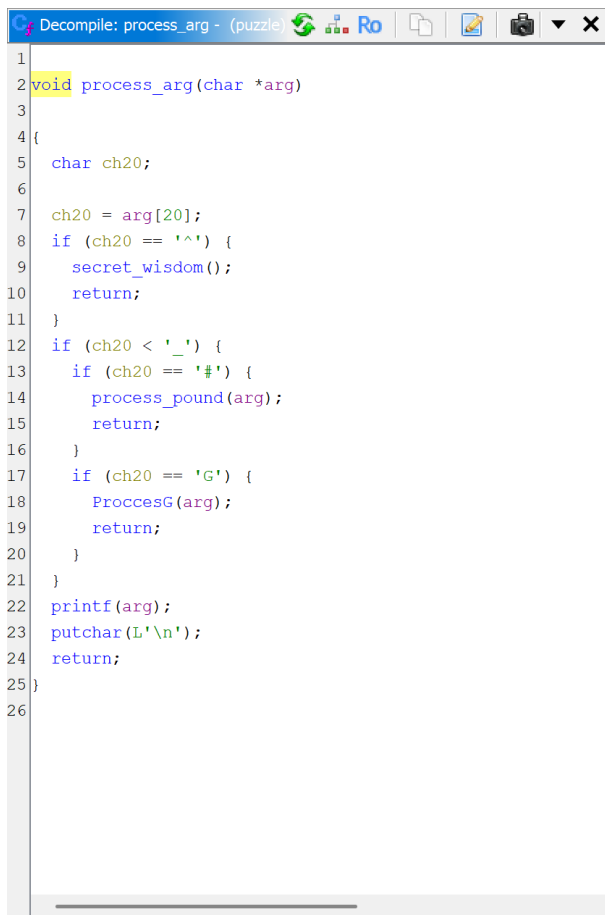
**By Sebastian Medina**

## **Content:**

**Walk though of Hands-on Demo.....pg1**

## Ghidra SRE Challenge

The first thing that we did during the session examined the “puzzle” binary within Ghidra. While in Ghidra we learned how to navigate the binary like how function looks how to change names of function or variables as well change the value of a variable. During the demo we learned what to look for and then changed the names and values to make the code a lot easier to read. Once we did that, we entered the main function then we entered another function that we called process arg that took us too the if statement to enter the three secrets we had to find. The photo is shown below.

A screenshot of the Ghidra decompiler interface. The title bar reads "Decompile: process\_arg - (puzzle)". The code is displayed in a monospaced font with syntax highlighting. The function signature is "void process\_arg(char \*arg)". The code includes several conditional statements: an if statement for 'ch20 == '^'', an if statement for 'ch20 < '\_' containing an if for 'ch20 == '#', and an if statement for 'ch20 == 'G'. The function concludes with a printf statement and a putchar statement. Line numbers 1 through 26 are visible on the left margin.

```
1
2 void process_arg(char *arg)
3
4 {
5     char ch20;
6
7     ch20 = arg[20];
8     if (ch20 == '^') {
9         secret_wisdom();
10        return;
11    }
12    if (ch20 < '_') {
13        if (ch20 == '#') {
14            process_pound(arg);
15            return;
16        }
17        if (ch20 == 'G') {
18            ProccesG(arg);
19            return;
20        }
21    }
22    printf(arg);
23    putchar(L'\n');
24    return;
25 }
26
```

Once we have entered the process\_arg function and found the three secrets the next step is to view the if statement to see what the requirements had to be met

to enter the function and then what we must meet within the function we just entered to reveal the secret of the program. The first if statement was easy all you had to do to was enter in 20 character and the last one had to be a ^ character. This allowed us to enter the function that we called secret wisdom. In the function we did not have to meet any requirements so once you entered 20 characters and the last character being a ^ the program printed out a message that said, “Secret Wisdom”. Below you will see a snapshot of the secret and what is inside the secret wisdom function.

```
(base) └─(ssmedina@sebshome)-[~/School/A4_Ghidra/ghidra-ils]
└─$ ./puzzle 01234567890123456789^
secret wisdom

(base) └─(ssmedina@sebshome)-[~/School/A4_Ghidra/ghidra-ils]
└─$
```

```
Decompile: secret_wisdom - (puzzle)
1
2 void secret_wisdom(void)
3
4 {
5     undefined4 local_1e;
6     undefined4 local_1a;
7     undefined4 local_16;
8     undefined2 local_12;
9     uint local_10;
10
11     local_1e = 0x81909680;
12     local_1a = 0x84d38796;
13     local_16 = 0x9c97809a;
14     local_12 = 0xf39e;
15     for (local_10 = 0; local_10 < 0xe; local_10 = local_10 + 1) {
16         *(byte *) ((int)&local_1e + local_10) = *(byte *) ((int)&local_1e + 1);
17     }
18     puts((char *)&local_1e);
19     return;
20 }
21
```

For the next secret we had to enter the function that we called process\_pound. To do this, we had to meet the requirements of the if statement

which was the 20 characters must be less than ‘\_’ and the 20 character has to be equal to the # sign. Once these requirements were met then the program takes you into the process\_pound function where another set of requirements must be met to see the secret. The requirement that had to be met was that the character that you entered in spaces [0], [4], [8], [12] and you add up their ascii values it will equal 260. So, a quick recap you must make sure that you have entered in at least 20 characters, the 20<sup>th</sup> character must be a # and then the character positioned in [0], [4], [8], [12] ascii values must be equal to 260 to get the secret that prints out “Welcome to the inner chamber”. Below is the function of process\_pound and the second secret.

```
20: parse_error: need ...  
(base) └─(ssmedina@sebshome)─[~/Schhol/A4_Ghidra/ghidra-ils]  
└─$ ./puzzle A123B563C712\>3345678#  
welcome to the inner chamber  
  
(base) └─(ssmedina@sebshome)─[~/Schhol/A4_Ghidra/ghidra-ils]  
└─$
```

```
Decompile: process_pound - (puzzle)  
1  
2 void process_pound(char *arg)  
3  
4 {  
5     if ((int)arg[12] + (int)*arg + (int)arg[4] + (int)arg[8] == 260) {  
6         inner_chamber();  
7     }  
8     return;  
9 }  
10
```

Now all that is left is the last secret that we must find. The requirement that had to be met to enter the last secret function called ProcessG was that the 20 characters that you input must be equal to G and the program will take you inside function ProcessG. Once you have entered the ProcessG function to get the secret you have to meet the requirements of the new bool variable that is set to true must be true and that the 7<sup>th</sup> character of your input must be a @. However, there is a for loop before the if statement that takes you to the secret that checks if you 20 characters you inputted is a palindrome because if it is not then the proceed variable get turned to false and the if statement will then not meet the requirements to see the secret. So, in order to see the third and final secret you must have a palindrome with the first and last character G, and the 7<sup>th</sup> and 13<sup>th</sup> character spots are @ in order to see the last secret that an ascii art of a dragon. Below are some snapshots of the secret and its function.

[illegible]

