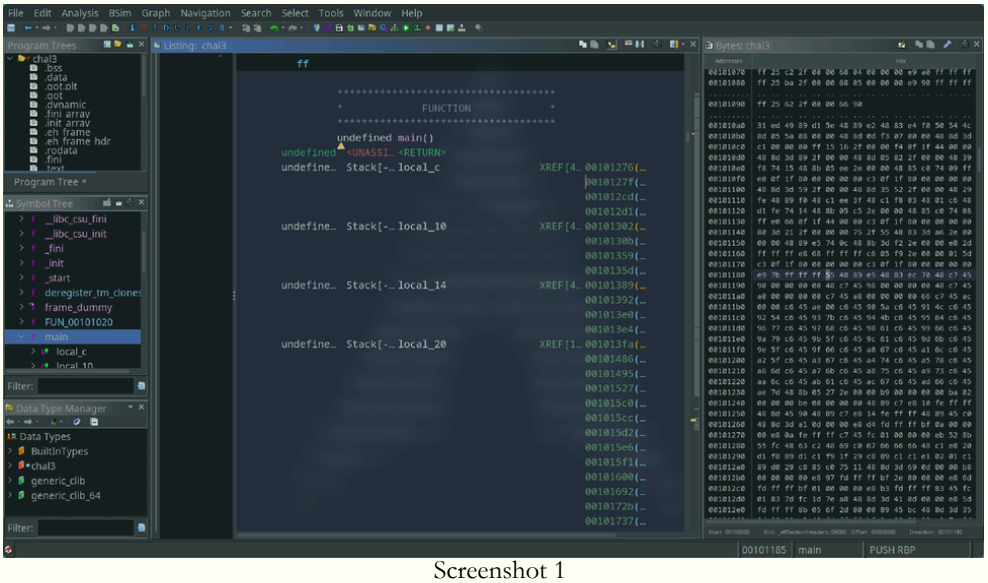
Importing “chal3” File To Ghidra

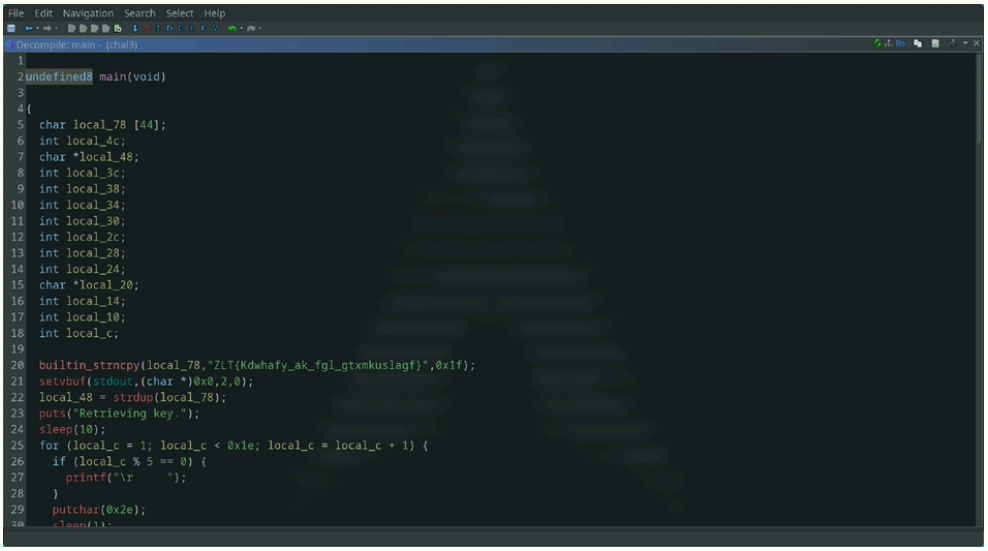
After importing the “chal3” file into Ghidra and it turns the machine code back into readable form i started to search for entry point and i found an interesting function called **main** showing in Screenshot 1.



Screenshot 1

Decompiled The Function

Using the Ghidra Decompiler, the assembly instructions were translated into C-like pseudocode for better readability as we can see in Screenshot 2.



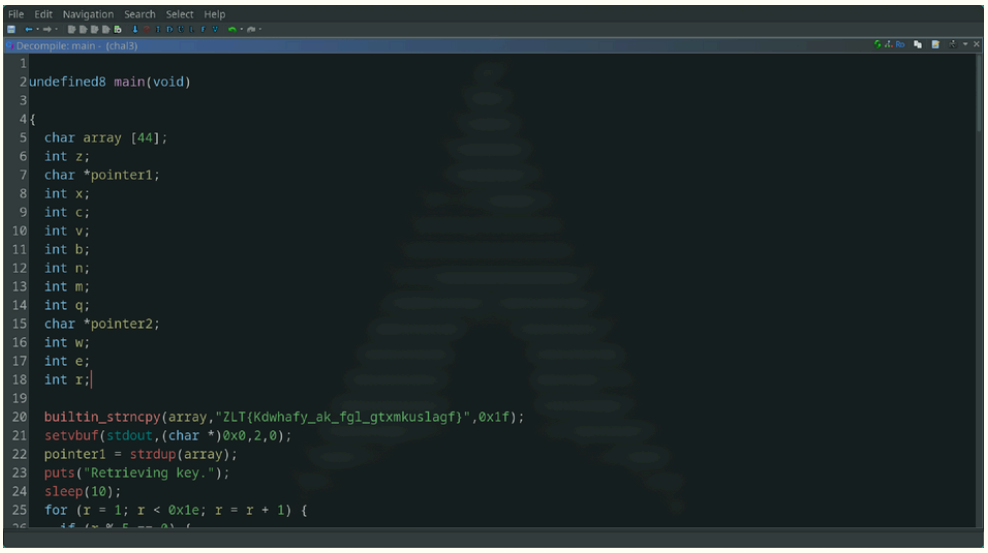
```
undefined8 main(void)

{
  char local_78 [44];
  int local_4c;
  char *local_48;
  int local_3c;
  int local_38;
  int local_34;
  int local_30;
  int local_2c;
  int local_28;
  int local_24;
  char *local_20;
  int local_14;
  int local_10;
  int local_c;

  builtin_strncpy(local_78,"ZLT{Kdwhafy_ak_fgl_gtxmkuslagf}",0x1f);
  setvbuf(stdout,(char *)0x0,2,0);
  local_48 = strdup(local_78);
  puts("Retrieving key.");
  sleep(10);
  for (local_c = 1; local_c < 0x1e; local_c = local_c + 1) {
    if (local_c % 5 == 0) {
      printf("\r    ");
    }
    putchar(0x2e);
    sleep(1);
```

Screenshot 2

Inside the **main** function it was a 150 line so its little long so to make it more easier for me to read i started by renaming the variables by looking at the type as showing in Screenshot3



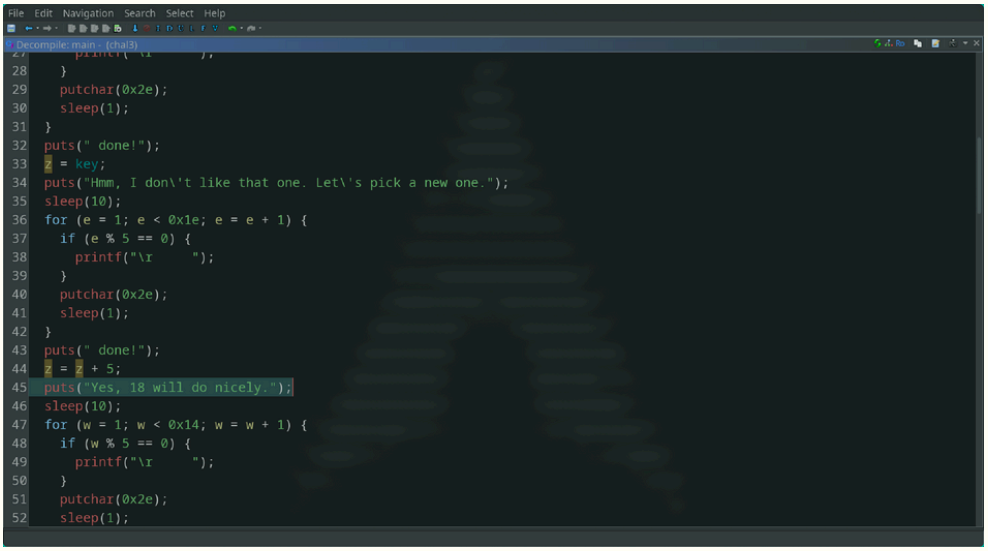
```
undefined8 main(void)

{
  char array [44];
  int z;
  char *pointer1;
  int x;
  int c;
  int v;
  int b;
  int n;
  int m;
  int q;
  char *pointer2;
  int w;
  int e;
  int r;

  builtin_strncpy(array,"ZLT{Kdwhafy_ak_fgl_gtxmkuslagf}",0x1f);
  setvbuf(stdout,(char *)0x0,2,0);
  pointer1 = strdup(array);
  puts("Retrieving key.");
  sleep(10);
  for (r = 1; r < 0x1e; r = r + 1) {
```

Screenshot 3

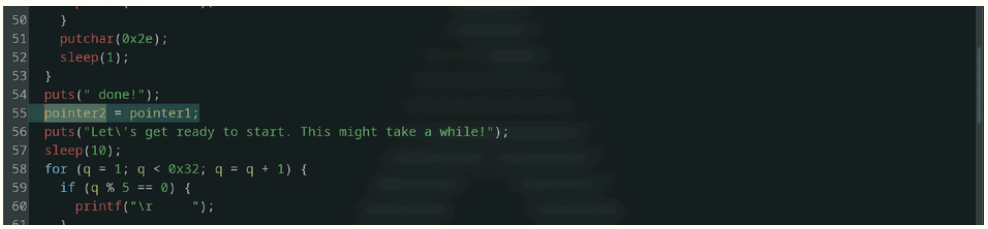
So i read the code and found that there is an "**arry**" that stores "**ZLT{Kdwhafy_ak_fgl_gtxmkuslagf}**" looks like encrypted string, and "**pointer1**" is pointing to "**array**" and there is alot of "**for loop"** The programmer added several features to make this code take a very long time to finish, Anyways i also found an interesting variable "**z**" that stores "**key**" an External data maybe its something to use for decryption, Also after few lines i found that its called again and adds 5 to it "**z = z + 5**" and there is a print just after it that says "**Yes, 18 will do nicely.**" as showing in Screenshot 4, so maybe its related so maybe z = 18 after the previous operation, im not sure but i kept it in mind.



```
  }
  putchar(0x2e);
  sleep(1);
}
puts(" done!");
z = key;
puts("Hmm, I don\'t like that one. Let\'s pick a new one.");
sleep(10);
for (e = 1; e < 0x1e; e = e + 1) {
  if (e % 5 == 0) {
    printf("\r     ");
  }
  putchar(0x2e);
  sleep(1);
}
puts(" done!");
z = z + 5;
puts("Yes, 18 will do nicely.");
sleep(10);
for (w = 1; w < 0x14; w = w + 1) {
  if (w % 5 == 0) {
    printf("\r     ");
  }
  putchar(0x2e);
  sleep(1);
```

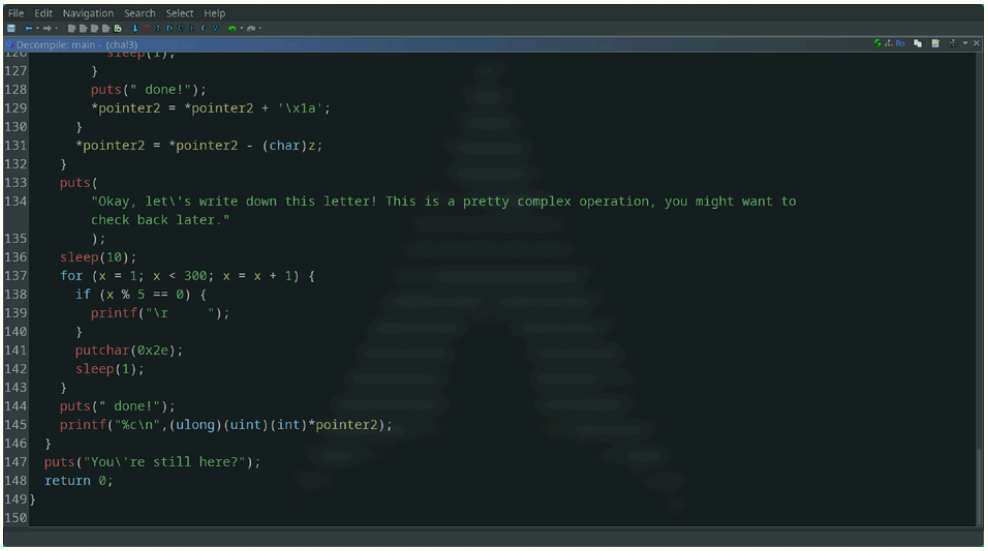
Screenshot 4

At line 55 pointer2 is equal pointer1 as showing in Screenshot5 so pointer2 now is = array



```
  }
  putchar(0x2e);
  sleep(1);
}
puts(" done!");
pointer2 = pointer1;
puts("Let\'s get ready to start. This might take a while!");
sleep(10);
for (q = 1; q < 0x32; q = q + 1) {
  if (q % 5 == 0) {
    printf("\r     ");
```

Screenshot 5



```
File  Edit  Navigation  Search  Select  Help
Decompile: main - (chal3)
127         }
128         puts(" done!");
129         *pointer2 = *pointer2 + '\x1a';
130       }
131       *pointer2 = *pointer2 - (char)z;
132     }
133     puts(
134         "Okay, let\'s write down this letter! This is a pretty complex operation, you might want to
            check back later."
135         );
136     sleep(10);
137     for (x = 1; x < 300; x = x + 1) {
138       if (x % 5 == 0) {
139         printf("\r     ");
140       }
141       putchar(0x2e);
142       sleep(1);
143     }
144     puts(" done!");
145     printf("%c\n",(ulong)(uint)(int)*pointer2);
146   }
147   puts("You\'re still here?");
148   return 0;
149 }
150
```

Screenshot 6

Ok! now in Screenshot 6 there it is using the "**z**" again on the "***pointer2 = *pointer2 - (char)z;**" so here where the decryption happens and after that at line 145 it prints the first letter from **ZLT{Kdwhafy_ak_fgl_gtxmkuslagf}** which is **"Z"** but after decryption so it might be another letter so lets run the program and only check the first letter to print as we can see in Screenshot 7.

```
➜ 0s ⚙ ./chal3                                                    □ Reverse/Target py ll  16:57
Retrieving key.
     ..... done!
Hmm, I don't like that one. Let's pick a new one.
     ..... done!
Yes, 18 will do nicely.
     ..... done!
Let's get ready to start. This might take a while!
     ..... done!
This one's an uppercase letter!
     ..... done!
Okay, let's write down this letter! This is a pretty complex operation, you might want to check back later.
     ..... done!
H
This one's an uppercase letter!

➜ 8m27s ○                                                         □ Reverse/Target py ll  17:05
```

Screenshot 7

After waiting 5min finally it prints the first letter "H" and i stoped the program cause it will take approx 5 minutes per character to print them all which is alot so lets use math to make a little python script to decode all letters in less than 1 second, So now we know that Z converted to H and Z in alphabet is numb 26 and H is 8 so 26 - 8 = 18 now i remember when it said "**Yes, 18 will do nicely**" so the "**key**" was equal **13** so the operation is "**z = 13 + 5**" pretty nice now as we know that we can create a simple python code to give us the whole decrypted string:

```python
def decode_flag():
    # The encoded string from the C code
    encoded = "ZLT{Kdwhafy_ak_fgl_gtxmkuslagf}"

    # The key determined in the code (original key + 5 = 18)
    z = 18
    decoded = ""

    for char in encoded:
        if 'a' <= char <= 'z':
            # Decode lowercase
            # We subtract the shift, then handle the wrap-around using modulo 26
            new_char = chr(((ord(char) - ord('a') - z) % 26) + ord('a'))
            decoded += new_char
        elif 'A' <= char <= 'Z':
            # Decode uppercase
            new_char = chr(((ord(char) - ord('A') - z) % 26) + ord('A'))
            decoded += new_char
        else:
            # Leave symbols like { } and _ alone
            decoded += char

    return decoded

print(f"Decoded Key: {decode_flag()}")
```

After running the code here is the output showing in Screenshot 8 :

Screenshot 8

As we can see the output is : "**HTB{Sleping_is_not_obfuscation}**"