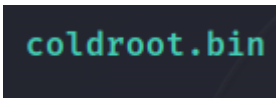
Reverse Lab Writeup

Coldroot: Part 1

So you managed to download the executable "coldroot.bin", now what?

1. FIrstly, we have to make sure it can be executed so we run:

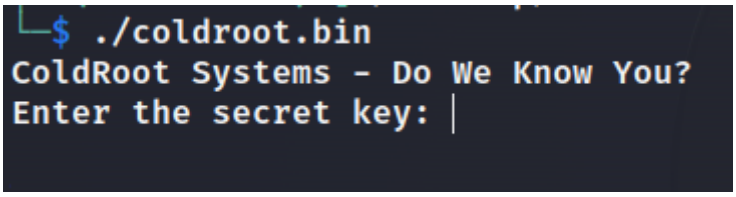
```
chmod +x coldroot.bin
```



It should change from white color to this green color, this indicated it can be executed.

2. Now, we try to run it to maybe understand what is the main idea behind it, or even look for clues:
   `Note: Make sure you are in the same directory as the file ./ means same folder`

```
./coldroot.bin
```



Ok so its asking for some secret key, now our next goal is to find this secret key.
3) Now, Lets start doing static analysis and check what is happening behind the scenes when we run the file.

```
strings coldroot.bin
```

The strings commands shows us what words are hidden/used by this file, so we scrolled through some system strings, then we found something interesting:

```
httx:/P@H
steL1nkxH
kx.com/
3Xfat9zbH
hiddenTrH
easure
ColdRoot Systems - Do We Know You?
Enter the secret key:
%49s
sysexit
Unauthorized access! Reporting to HQ.
Welcome to ColdRoot Systems.
Security level increased. Data encryption in process ...
Data Encrypted Successfully, you appear to be a stranger!
Here is a message for you:
aHR0cHM6Ly9wYXN0ZWJpbi5jb20vUWhxcmdSYkE=
- ColdRoot Operators
```

So, here is the summary of these findings:

- There is an "htt", so this indicated that there is a hidden URL that leads us to something.
- A base64 encoding that is sus and worth decoding and knowing whats behind it.
- There is a message saying data is encrypted, maybe this will appear when we put the right secret key?

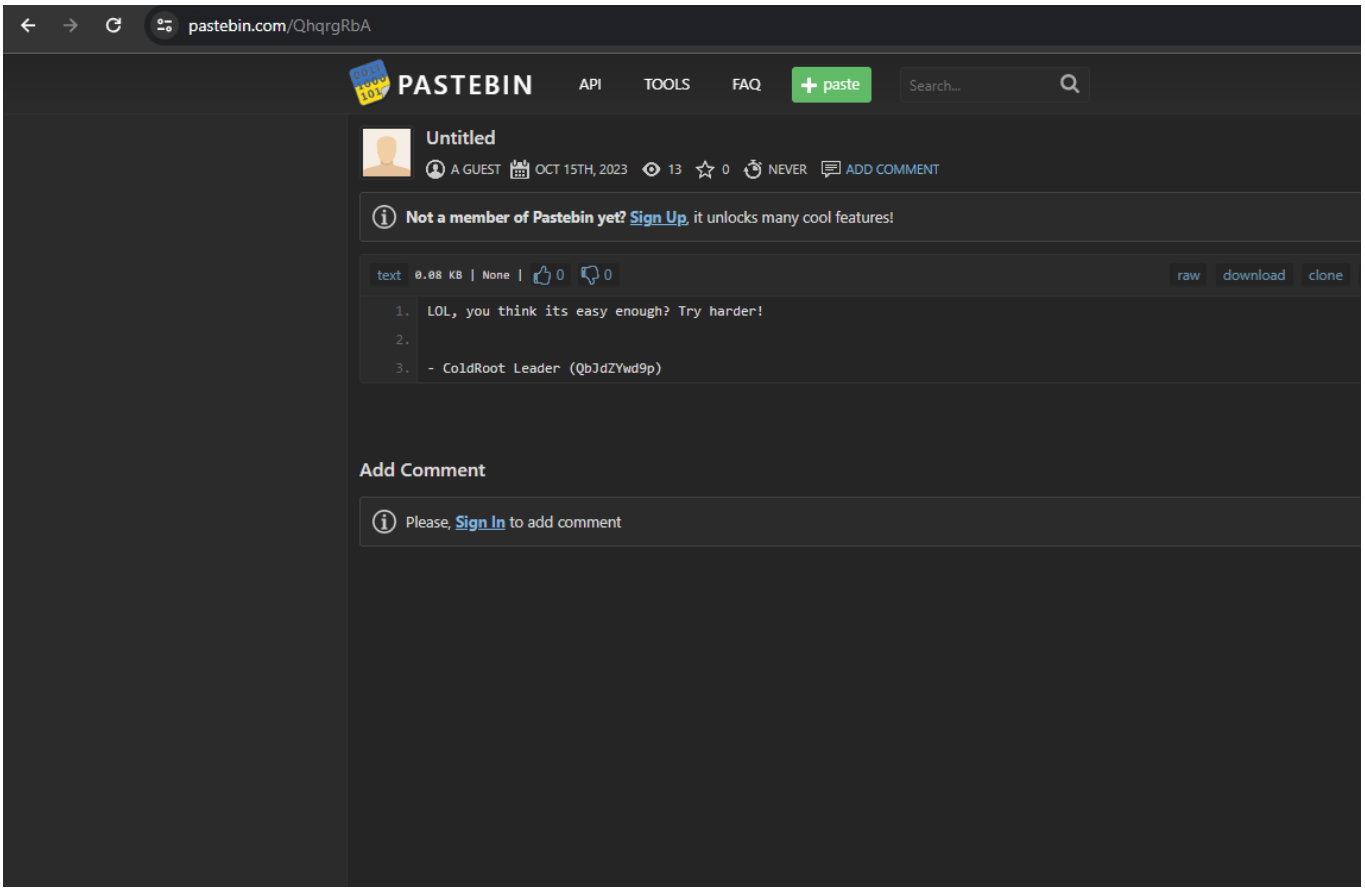
We noted that, now we will decode the base64 string and proceed with other analysis techniques.

We will run this command to decode the base64:

```
echo "aHR0cHM6Ly9wYXN0ZWJpbi5jb20vUWhxcmdSYkE=" | base64 -d
```

```
┌──(kali㉿kali)-[~/Desktop/CTF-Training/RE/Custom]
└─$ echo "aHR0cHM6Ly9wYXN0ZWJpbi5jb20vUWhxcmdSYkE=" | base64 -d
https://pastebin.com/QhqrgRbA
```

Oh well, here is a URL, lets visit it!



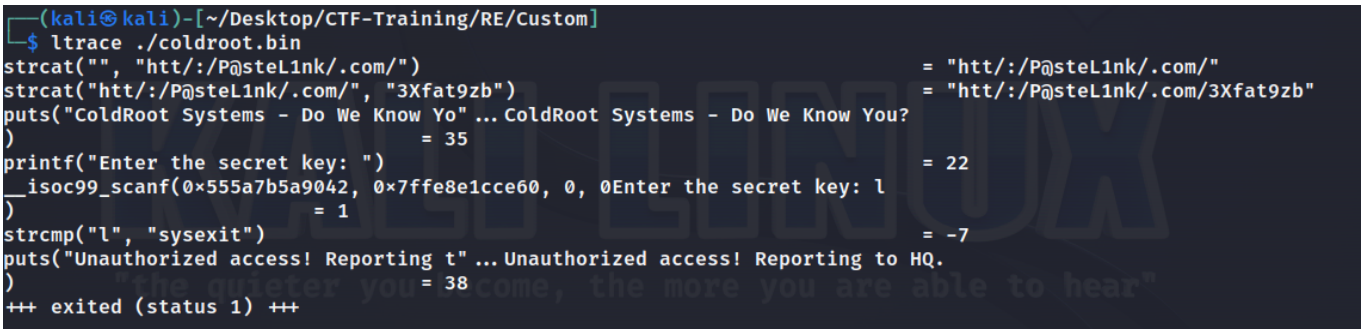
Ah its a trick, but pay attention, there is a clue which is "QbJdZYwd9p", its worth saving, will be useful later on I guess.

4. Now lets proceed with other analysis techniques

Now, we will be using ltrace, which captures library calls, and is useful to analyze what happens behind the scenes when we run the executable:

```
ltrace ./coldroot.bin
```

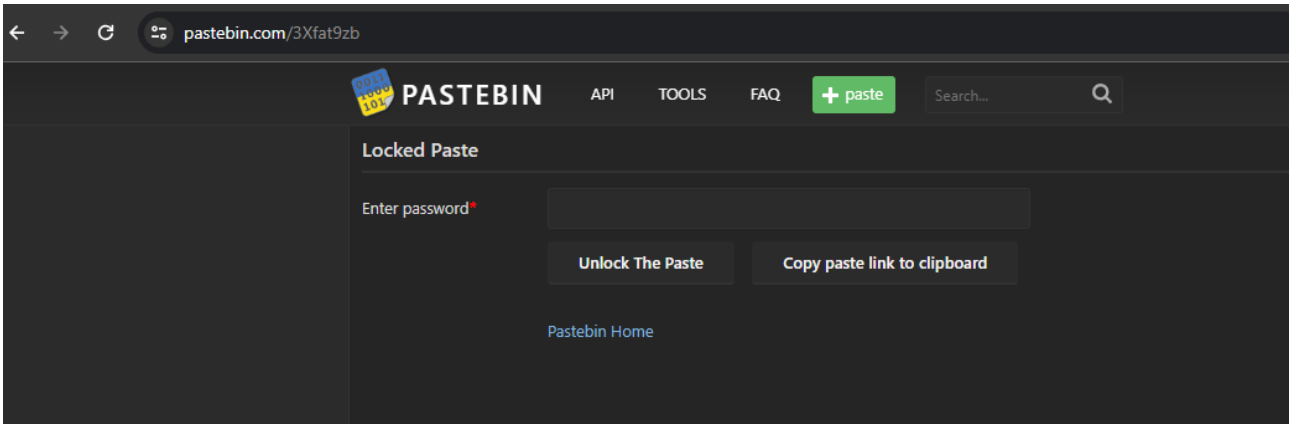
It will stop at the secret key input, you cant enter anything and press ENTER.



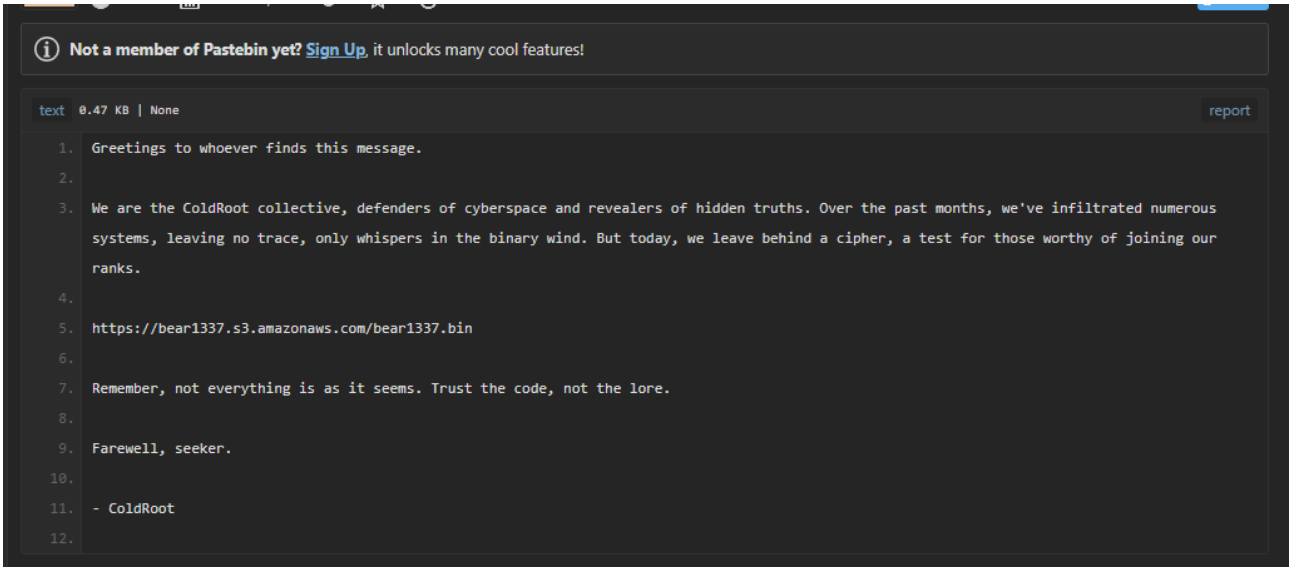
As you can see, here are the functions that are being used at runtime, there is a "strcat" function that has two parts:

- "htt/:/P@steL1nk/.com/"
- "3Xfat9zb"
  From this information, we concluded that its a pastebin URL that is: https://pastebin.com/3Xfat9zb

5. Lets visit the URL



Its asking for a password, remember the string "QbJdZYwd9p" we found earlier? Lets try it as a password



It worked!

There is a message that gives us a link to another file caleld bear1337, lets wget it and analyze.

Coldroot: Part 2

As we discovered from part 1, we found a file online called "bear1337.bin", now lets download it:

```
wget https://bear1337.s3.amazonaws.com/bear1337.bin
```

And lets make sure it can be executed by using chmod +x.

1. Lets run the executable:

```
┌──(kali㉿kali)-[~/Desktop/CTF-Training/RE/Custom]
└─$ ./bear1337.bin
Enter the colors to decrypt the flag:
Enter the first color: e
Enter the second color: e
Enter the third color: d
Encrypted flag:
Try again!
Enter the first color:
```

It's asking for three colors, lets analyze it further by running strings on it

```
Encrypted flag: %s
White
Blue
Decrypted flag: %s
Try again!
;*3$"
GCC: (Debian 12.2.0-14) 12.2.0
size_t
color2
decrypted_flag
__isoc99_scanf
strlen
dest
unsigned char
input
strcpy
dynamic_key
short unsigned int
strcat
GNU C17 12.2.0 -mtune=generic -march=x86-64 -g -fasynchronous-unwind-tables
memset
color1
color3
main
constructFlag
key_base
long long unsigned int
strcmp
```

```
xorOperation
```

Here are some useful findings:

- strcmp which probably compares our input with the expected input
- White and Blue Colors
- constructFlag function that constructs the flag
- xorOperation which probably does xoring

2. We ran ltrace too:

```
┌──(kali㉿kali)-[~/Desktop/CTF-Training/RE/Custom]
└─$ ltrace ./bear1337.bin
strlen("")                                                          = 0
strlen("")                                                          = 0
strlen("ColdRoot")                                                  = 8
memset(0x7ffeae694860, '\0', 38)                                    = 0x7ffeae694860
puts("Enter the colors to decrypt the " ... Enter the colors to decrypt the flag:
)                                   = 38
printf("Enter the first color: ")                                   = 23
__isoc99_scanf(0x5581c8bd6046, 0x7ffeae69490d, 0, 0Enter the first color: s
)                                   = 1
printf("Enter the second color: ")                                  = 24
__isoc99_scanf(0x5581c8bd6046, 0x7ffeae694903, 0, 0Enter the second color: d
)                                   = 1
printf("Enter the third color: ")                                   = 23
__isoc99_scanf(0x5581c8bd6046, 0x7ffeae6948f9, 0, 0Enter the third color: w
)                                   = 1
strcpy(0x7ffeae694860, "ColdRoot")                                  = 0x7ffeae694860
strcat("ColdRoot", "s")                                             = "ColdRoots"
strcat("ColdRoots", "d")                                            = "ColdRootsd"
strcat("ColdRootsd", "w")                                           = "ColdRootsdw"
strlen("")                                                          = 0
printf("Encrypted flag: %s\n", "\002"Encrypted flag:
)                                       = 18
strcmp("s", "White")                                                = 28
puts("Try again!"Try again!
)                                                       = 11
printf("Enter the first color: ")                                   = 23
__isoc99_scanf(0x5581c8bd6046, 0x7ffeae69490d, 0, 0Enter the first color:
```

3. Nothing much found, lets try to open ghidra and statically analyze the main function:

```
                    **************************************************************
                    *                          FUNCTION
                    **************************************************************
                    undefined main()
     undefined          AL:1           <RETURN>
     undefined1         Stack[-0x30]:1 local_30
     undefined8         Stack[-0x40]:8 local_40
     undefined8         Stack[-0x48]:8 local_48

     undefined8         Stack[-0x50]:8 local_50
     undefined8         Stack[-0x58]:8 local_58

     undefined8         Stack[-0x60]:8 local_60
     undefined8         Stack[-0x68]:8 local_68
```

```
103   puVar2 = (undefined *)local_48;
104   alStack_120[uVar5 * -2 + uVar6 * -2 + uVar8 * -2] = 0x101659;
105   printf("Encrypted flag: %s\n",puVar2);
106   alStack_120[uVar5 * -2 + uVar6 * -2 + uVar8 * -2] = 0x101672;
107   iVar4 = strcmp(local_bb,"White");
108   if (iVar4 == 0) {
109     alStack_120[uVar5 * -2 + uVar6 * -2 + uVar8 * -2] = 0x10168f;
110     iVar4 = strcmp(local_c5,"Blue");
111     if (iVar4 == 0) {
112       alStack_120[uVar5 * -2 + uVar6 * -2 + uVar8 * -2] = 0x1016ac;
113       iVar4 = strcmp(local_cf,"Red");
114       puVar3 = (undefined *)local_48;
115       puVar2 = (undefined *)local_58;
116       pcVar1 = (char *)local_68;
117       if (iVar4 == 0) {
118         alStack_120[uVar5 * -2 + uVar6 * -2 + uVar8 * -2] = 0x1016c7;
119         xorOperation(puVar3,puVar2,pcVar1);
120         puVar2 = (undefined *)local_58;
121         alStack_120[uVar5 * -2 + uVar6 * -2 + uVar8 * -2] = 0x1016e2;
122         printf("Decrypted flag: %s\n",puVar2);
```

4. We found the 3 colors, lets try running them in this order, White -> Blue -> Red

```
┌──(kali㉿kali)-[~/Desktop/CTF-Training/RE/Custom]
└─$ ./bear1337.bin
Enter the colors to decrypt the flag:
Enter the first color: White
Enter the second color: Blue
Enter the third color: Red
Encrypted flag:
Decrypted flag: A⌖w4⌖
```

5. Great, But what is that? There is something happening dynamically and the constructFlag is probably doing it.

So our next step probably is dynamic analysis using GDB and trying to break on main.

```
gdb ./bear1337.bin
```

```
┌──(kali㉿kali)-[~/Desktop/CTF-Training/RE/Custom]
└─$ gdb ./bear1337.bin
GNU gdb (Debian 13.2-1) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bear1337.bin...
(gdb)
```

Now, lets break main and move step by step using "next"

```
(gdb) break main
Breakpoint 1 at 0x136f: file bear1337.c, line 36.
(gdb) run
Starting program: /home/kali/Desktop/CTF-Training/RE/Custom/bear1337.bin
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at bear1337.c:36
36      int main() {
(gdb) next
37          char flag[50] = {0};
(gdb) next
39          char encrypted_flag[strlen(flag) + 1];
(gdb) next
40          char decrypted_flag[strlen(flag) + 1];
(gdb) next
42          char key_base[] = "ColdRoot";
(gdb) next
43          char dynamic_key[strlen(key_base) + 30];
(gdb) next
44          memset(dynamic_key, 0, sizeof(dynamic_key));
(gdb) next
47          printf("Enter the colors to decrypt the flag:\n");
(gdb) next
Enter the colors to decrypt the flag:
50              printf("Enter the first color: ");
(gdb) next
```

Nothing much, now lets try to break on constructFlag:

```
(gdb) break constructFlag
Breakpoint 1 at 0x11b1: file bear1337.c, line 6.
(gdb) run
Starting program: /home/kali/Desktop/CTF-Training/RE/Custom/bear1337.bin
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter the colors to decrypt the flag:
Enter the first color: White
Enter the second color: Blue
Enter the third color: Red

Breakpoint 1, constructFlag (dest=0x7fffffffdd60 "") at bear1337.c:6
6           dest[0] = 'c' ^ 'c';
(gdb) next
7           dest[1] = 's' ^ 's';
(gdb) next
8           dest[2] = 'c' ^ 'c';
(gdb) next
9           dest[3] = '{' ^ '{';
(gdb) next
10          dest[4] = 't' ^ 't';
```

Lets go! We break on constructFlag and provided the correct colors in order as input,

using "next" command the program is giving us the flag which appears to be "csc{tHe_rUsSian_BeAr}".

We knew the last character of the flag when this appeared to us which is the NULL terminator char

```
(gdb) next
27          dest[21] = '\0';
(gdb) next
28      }
```

- Note: There are several ways to solve these challenges, the one provided is just one of the ways.

That's it, I hope you had fun with this challenge, and if you have any feedback or want to get in touch:

https://github.com/@smadi0x86
https://www.linkedin.com/in/saud-smadi/