RingZer0 CTF: Kernel introduction

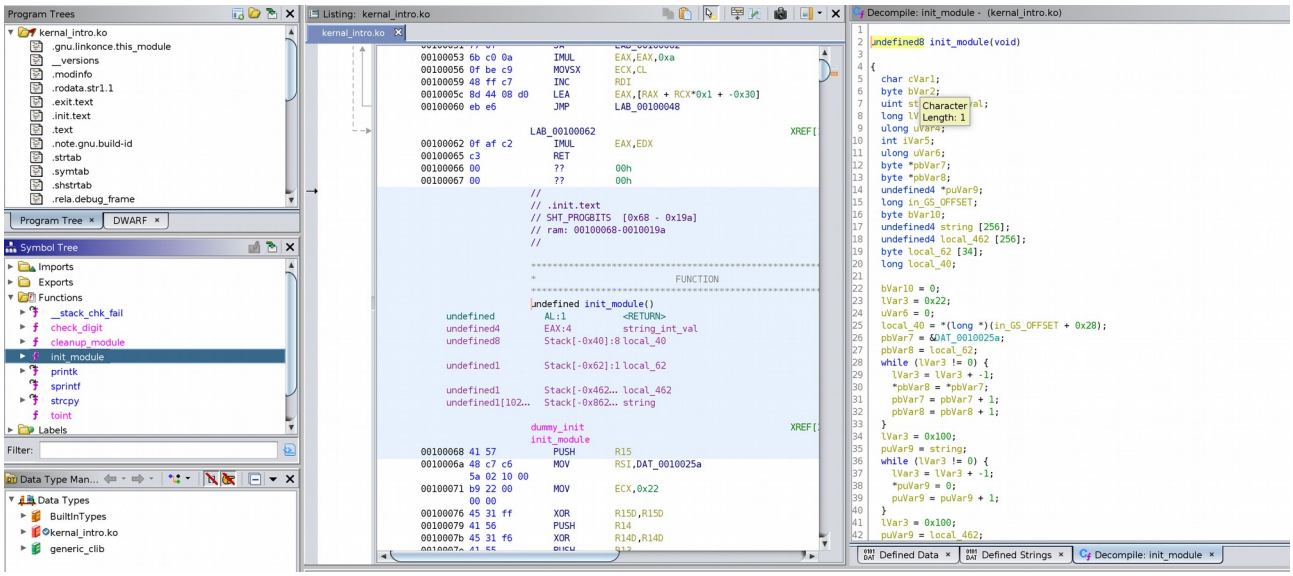
With this challege we were given a “.ko” file which is linux kernel module.

Kernel modules are “plugins” to the kernel which can be loaded and unloaded on demand without the need to reboot the system.
They are mostly used for developing device drivers.

Since im using different kernel version and distro, i decided to reverse the program staticlly without debugging it.

Lets view the code !

Loading the program with “ghidra”, we can see that is unstripped binary, and we can see clear function names.



The analogues to main function for kernel modules is init_module, so jumping right to that function we see 3 buffers declared.
One of them is 34 bytes length, which is the length of typical flag.

Then, we see some data is loaded to pbVar7 and then copied to the 34 bytes string.

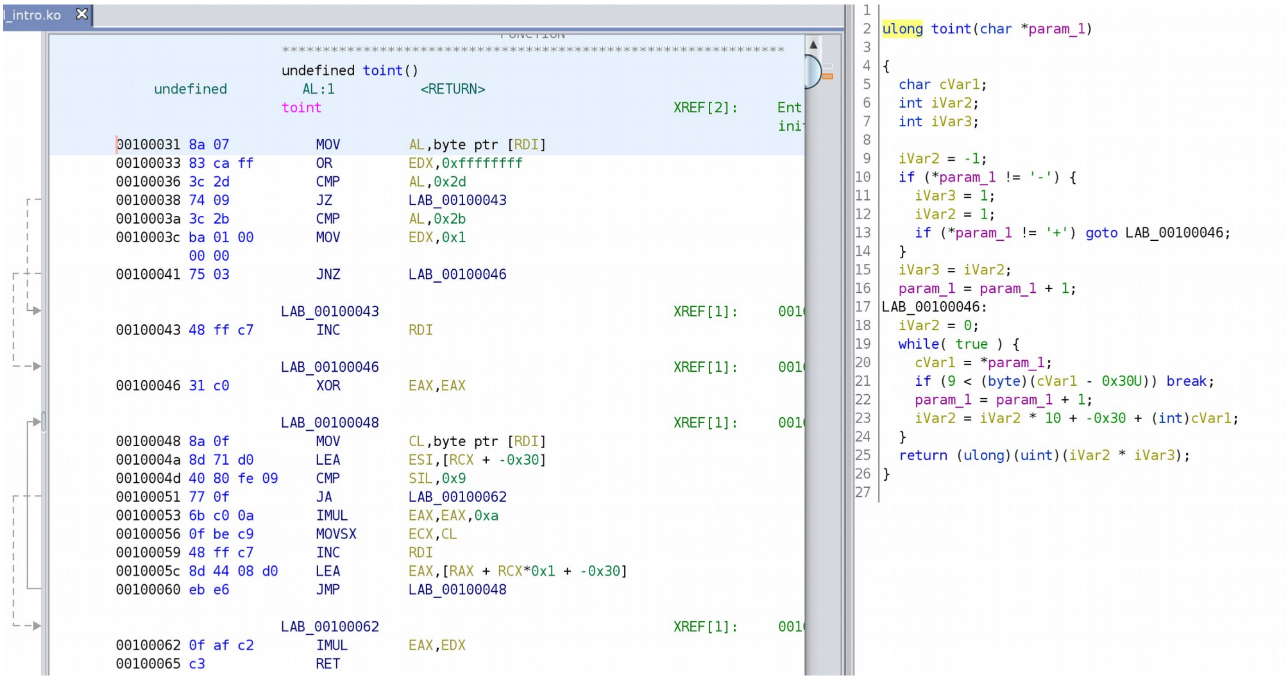
Looking at the data we see some garbage bytes, probably encrypted flag.

```
0010025a 76              undefined1  76h
0010025b 34              ??          34h    4
0010025c 71              ??          71h    q
0010025d 76              ??          76h    v
0010025e 4f              ??          4Fh    O
0010025f 4c              ??          4Ch    L
00100260 7b              ??          7Bh    {
00100261 42              ??          42h    B
00100262 0e              ??          0Eh
00100263 5e              ??          5Eh    ^
00100264 06              ??          06h
00100265 4e              ??          4Eh    N
00100266 02              ??          02h
00100267 72              ??          72h    r
00100268 50              ??          50h    P
00100269 01              ??          01h
0010026a 53              ??          53h    S
0010026b 07              ??          07h
0010026c 0c              ??          0Ch
0010026d 34              ??          34h    4
0010026e 06              ??          06h
0010026f 4b              ??          4Bh    K
00100270 07              ??          07h
00100271 04              ??          04h
00100272 2a              ??          2Ah    *
00100273 61              ??          61h    a
00100274 0a              ??          0Ah
00100275 66              ??          66h    f
00100276 73              ??          73h    s
00100277 53              ??          53h    S
00100278 03              ??          03h
00100279 4e              ??          4Eh    N
0010027a 04              ??          04h
0010027b 00              ??          00h
```

Next, we can notice hard coded string being copied to a buffer.
Then a function “toint” is called and the string is passed as a parameter.
Next, the value of the hard coded string is updated to the hex representation of the returned value from the previous “toint” call.

```
48    strcpy((char *)string,
49
50            "28714143kkl23jlsdkj34hji53jhk345khj543jhk354h354jh354jhkl354jhkl354hjk345hjk3h4i5h3l4h5iul
              34u6h4e5uh7ui5h7uilyhhiuyhuileyhlui6yhuilyuhil55hhuilhiw543uhiw34uhihuiuh6iwl354h"
51           );
52    string_int_val = toint(string);
53    iVar5 = 0;
54    sprintf((char *)string,"0x%08x",(ulong)string_int_val);
55    do {
56      uVar4 = 0xffffffffffffffff;
57      pbVar7 = local_62;
58      do {
59        if (uVar4 == 0) break;
60        uVar4 = uVar4 - 1;
61        bVar2 = *pbVar7;
62        pbVar7 = pbVar7 + (ulong)bVar10 * -2 + 1;
63      } while (bVar2 != 0);
64      if (~uVar4 - 1 <= uVar6) {
65        printk("<1>%s",local_462);
66        if (local_40 != *(long *)(in_GS_OFFSET + 0x28)) {
67                        /* WARNING: Subroutine does not return */
68          __stack_chk_fail();
69        }
70        return 0;
71      }
72      lVar3 = (long)iVar5;
73      iVar5 = iVar5 + 1;
74      sprintf((char *)local_462,"%s%c",local_462,
75              (ulong)((uint)local_62[uVar6] ^ (int)*(char *)((long)string + lVar3)));
76      uVar4 = 0xffffffffffffffff;
77      puVar9 = string;
78      do {
79        if (uVar4 == 0) break;
80        uVar4 = uVar4 - 1;
81        cVar1 = *(char *)puVar9;
82        puVar9 = (undefined4 *)((long)puVar9 + (ulong)bVar10 * -2 + 1);
83      } while (cVar1 != '\0');
84      if ((long)iVar5 == ~uVar4 - 1) {
85        iVar5 = 0;
86      }
87      uVar6 = uVar6 + 1;
88    } while( true );
89  }
90
```

Lets analyze “toint” function.



We can see that the return value is constructed as the multiplaction of two integers.
First entering the function we check for the string sign.
If it not “-” then ivar2 is becoming 1 , o.w ivar is -1.
Then , before entering a loop ivar3=ivar2, and ivar2 is zero’d,
Hence, ivar3 is the sign of the returned value, and ivar2 is the magnitude.
Entering the loop we can see that iterates over the characters of the string.
If (int)char > 0x39 then we exit.
The ascii representation of 0x39 is 9. which means only ascii numbers are taken into account.
Next the int representation is calculated in a standared way, the loop starts from the msb and adding to it the decimal value of the ascii char and accumaltes values.
So, for our hard coded string the toint() return value should be 28714143.
The hex representation is ‘0x01b6249f’- which is the decryption key.

Back to our init_module function, we can see a while loop, iterating over the encrypted flag buffer and xor’ing it with the key that calculated above.
Finally at line 64 we can see the loop exit condition, if the enc_flag index is equal to len(enc_flag) – 1 (which is 33) the flag is simply printed out using printk.

If we were loading the module then the flag was simply printed out to us.

Reversed c code to get the flag:

```c
#include <stdio.h>

int toint(char []);

char dec_str [] = "28714143kkl23jlsdkj34hji53jhk345khj543jhk354h354jh354jhkl354jhkl354hjk345hjk3h4i5h3l4h5iul34u6h4e5uh7ui5h7uilyhhiuyhuileyhlui6yhuilyuhil55hhuilhiw543uhiw34uh
char enc_flag [34] = {0x76, 0x34, 0x71, 0x76, 0x4F, 0x4C, 0x7B, 0x42, 0x0E, 0x5E, 0x06, 0x4E,
        0x02, 0x72, 0x50, 0x01, 0x53, 0x07, 0x0C, 0x34, 0x06, 0x4B, 0x07, 0x04, 0x2A, 0x61, 0x0A, 0x66, 0x73, 0x53, 0x03, 0x4E ,0x04};

int main(int argv, char ** argc)
{
    int str_index = 0;
    int enc_flag_index = 0;
    char dec_flag[256];
    char key [11];
    sprintf(key,"0x%08x", toint(dec_str));
    while( 1 )
    {
        if(enc_flag_index == 33)
        {
            printf("%s", dec_flag);
            printf("\n");
            break;
        }

        sprintf(dec_flag, "%s%c", dec_flag, (enc_flag[enc_flag_index] ^ (char) key[str_index]) );
        str_index += 1;

        if(str_index > 9)
            str_index = 0;

        enc_flag_index += 1;

    }
    return 0;
}
```

```c
int toint(char str [])
{
    int var1,var2, flag;
    int index = 0;
    char dig;
    var1 = -1;
    flag = 0;
    if(str[0] != 0x2d)
    {
        var1= 1;
        var2= 1;
        if(str[0] != 0x2b )
        {
            flag=1;
        }
    }
    if (flag == 0)
    {
        var2 = var1;
        index +=1;
    }
    var1 = 0;
    while( 1 )
    {
        dig = str[index];
        if((int)dig > 0x39)
            break;
        var1 = var1 * 0xa + (int)dig - 0x30;
        index += 1;
    }
    return (long)(var1 * var2);
}
```