[QuickNote] Techniques for decrypting BazarLoader strings

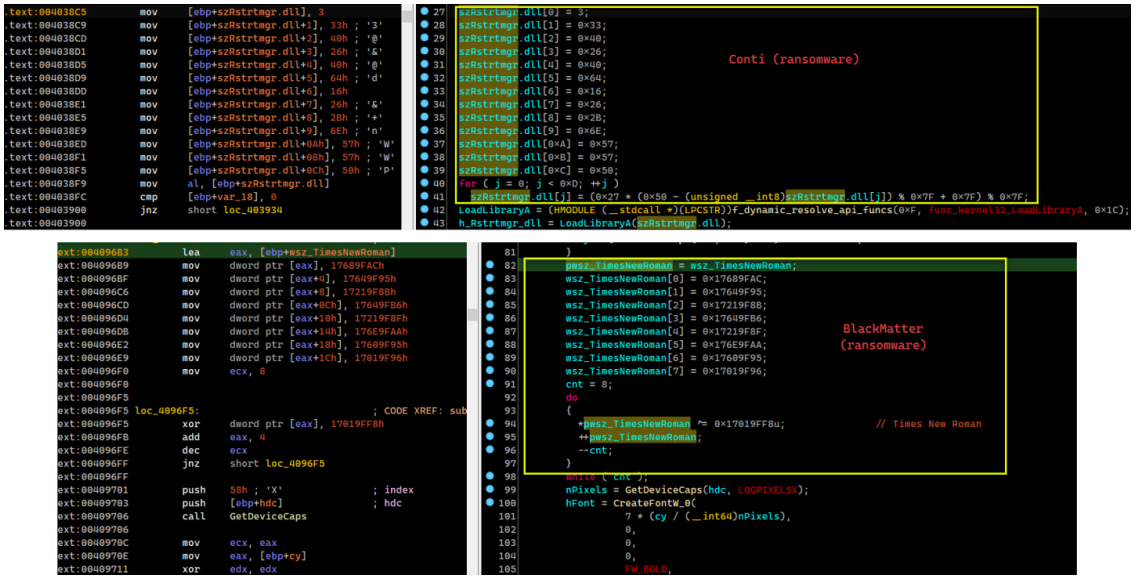
Published: 2022-02-24 · Archived: 2026-04-05 19:33:07 UTC

1. Overview

Usually, to make it more difficult for analysts, malware authors will hide important strings and only decrypt these strings during runtime. The famous malwares like Emotet, QakBot or TrickBot often use the one or some functions to perform decrypting strings when needed.

However, on researching and analyzing some other malwares such as **Conti**, **BlackMatter** and **BazarLoader**, instead of using a separate function to decrypt strings, these malwares make it more difficult by saving the encrypted strings on the stack as stack strings. Then, strings are decrypted by XOR-ing with a key value (this value may not be fixed) or through quite complex computation. This technique consumes time of the analyst.

The images below are the pseudocode of the Conti and BlackMatter malware.



This article uses the BazarLoader samples as an example to demonstrate how to decrypt strings with:

- Automate resolving with IDAPython script.
- Emulate code with IDA uEmu plugin.
- Debugging with IDA Bochs plugin.

2. BazarLoader samples

BazarLoader was first discovered in April 2020. The malware loader has been continuously evolving, allowing attackers to install additional malware, often used for ransomware attacks, dropping Cobalt Strike, and stealing

sensitive data. The common assumption is that the distribution and post-exploitation activities of the loader are akin to the Trickbot malware.

These samples are all **64-bit** Windows executable.

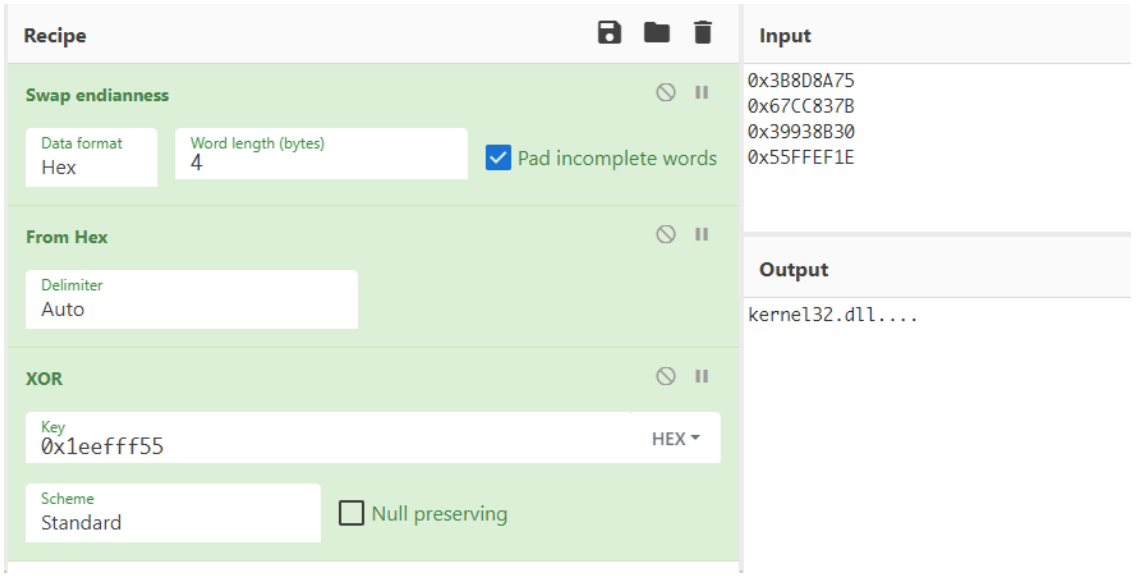
- Unpacked sample 1: [cc522400b3fed1d2c4dcca16666ddcff]
- Unpacked sample 2: [63c4bb3f1044f36632ce1759b62296dc]

3. Decrypt strings

3.1. Using IDApython script

Analyzing the first sample of BazarLoader, we will see that it uses the same stack strings decryption technique as in BlackMatter ransomware:

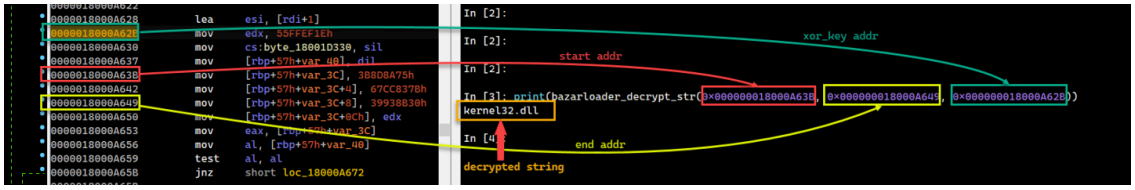
To decrypt these strings, you can use **x64dbg** to debug or extract the above values and use **CyberChef** to perform the following:



However, debugging with **x64dbg** or using **CyberChef** as above will take more time, to make static analysis easier, I will use **IDAPython** script to decrypt the strings. The code I use is as follows:

```python
import idc, idaapi, struct

def bazarloader_decrypt_str(start_ea, end_ea, xor_key_ea):
    enc_buf = []
    xor_key = []

    """get xor_key"""
    xor_key_val = struct.pack("<I", idc.get_operand_value(xor_key_ea, 1))
    for i in bytearray(xor_key_val): xor_key.append(i)

    '''get encoded bytes'''
    if start_ea is None or end_ea is None or xor_key_ea is None:
        print ('Not enough information to decrypt')
        return

    curr = start_ea
    while curr <= end_ea:
        enc_val = struct.pack("<I", idc.get_operand_value(curr, 1))

        for i in bytearray(enc_val): enc_buf.append(i)
        curr = idc.next_head(curr)

    '''decrypt'''
    for i in range(len(enc_buf)):
        enc_buf[i] ^= (xor_key[i % len(xor_key)]) & 0xFF

    return ''.join([chr(c) for c in enc_buf])
```

Load this script into IDA, providing the relevant addresses to perform the decryption:

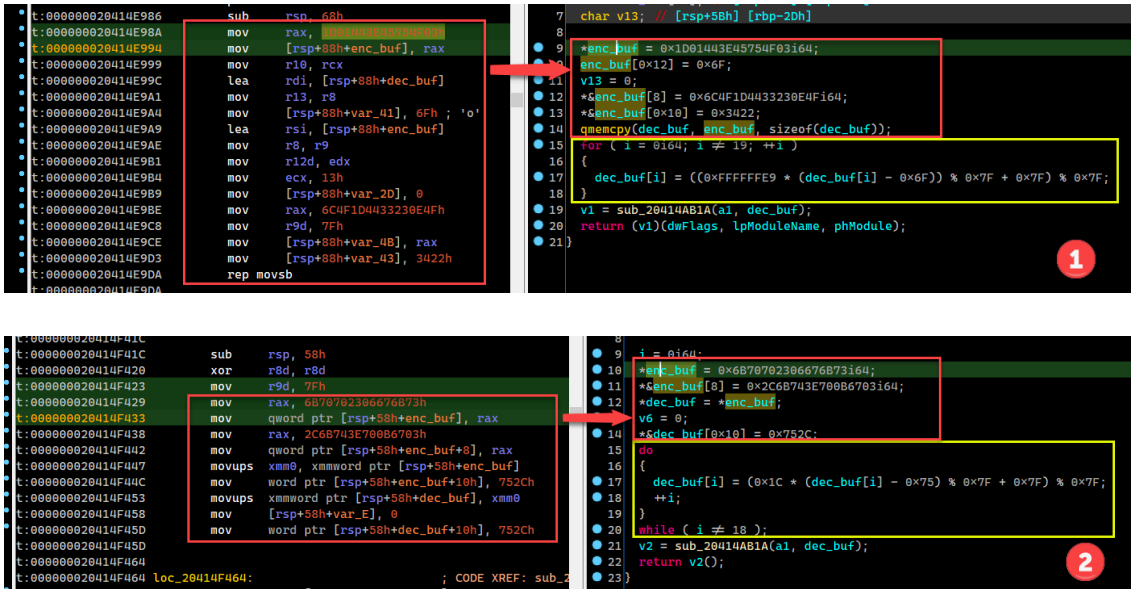


Finally, by using the above script, the analysis process will be much more convenient:

```
46  char v57; // [rsp+D4h] [rbp+2Bh]
47  int sz_advapi32.dll[4]; // [rsp+D8h] [rbp+2Fh]
48
49  if ( !byte_18001D330 )
50  {
51    byte_18001D330 = 1;
52    v55 = 0;
53    sz_kernel32.dll[0] = 0x3B8D8A75;
54    sz_kernel32.dll[1] = 0x67CC837B;
55    sz_kernel32.dll[2] = 0x39938B30;
56    sz_kernel32.dll[3] = 0x55FFEF1E;
57    for ( i = 0i64; i < 4; ++i )
58    {
59      sz_kernel32.dll[i] ^= 0x55FFEF1Eu;                    // kernel32.dll
60    }
61    lstrcpyA = (LPSTR (__stdcall *)(LPSTR, LPCSTR))f_bazar_dyn_resolve_API(1i64, func_kernel32_lstrcpyA, 0x80);
62    if ( lstrcpyA )
63    {
64      lstrcpyA(g_sz_kernel32_dll, (LPCSTR)sz_kernel32.dll);
65    }
66    v57 = 0;
67    sz_advapi32.dll[0] = 0x74DDFEF3;
68    sz_advapi32.dll[1] = 0x2798F3E2;
69    sz_advapi32.dll[2] = 0x79C7FEBC;
70    sz_advapi32.dll[3] = 0x15AB9A92;
71    for ( j = 0i64; j < 4; ++j )
72    {
73      sz_advapi32.dll[j] ^= 0x15AB9A92u;                    // advapi32.dll
74    }
75    lstrcpyA = (LPSTR (__stdcall *)(LPSTR, LPCSTR))f_bazar_dyn_resolve_API(1i64, func_kernel32_lstrcpyA, 0x80);
76    if ( lstrcpyA )
```

3.2. Using uEmu plugin

In the second sample of BazarLoader, the code that decrypt the stack strings is similar to the Conti ransomware and quite complicated:



With the code as shown in the figure, the implementation of using IDApython script will be difficult and not feasible. The most suitable solution for this case is to use an emulator to emulate the code. Here, I will use uEmu, a tiny cute emulator plugin for IDA based on unicorn engine.

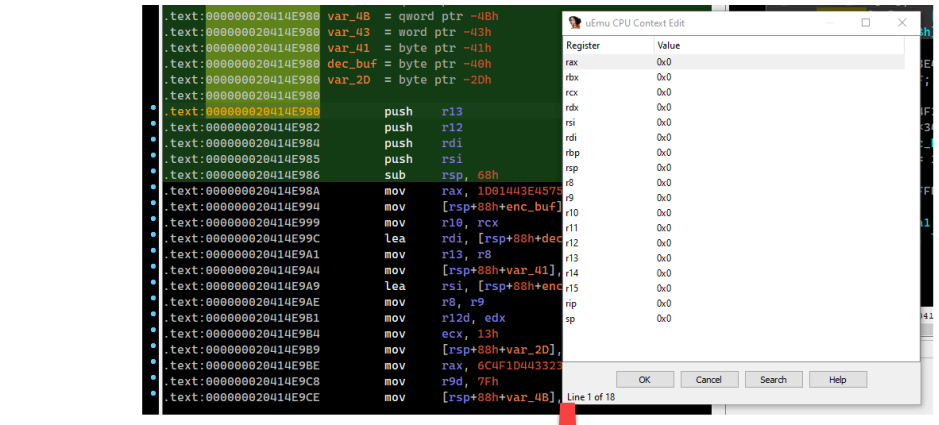
Very easy to emulate the decoding code with uEmu:

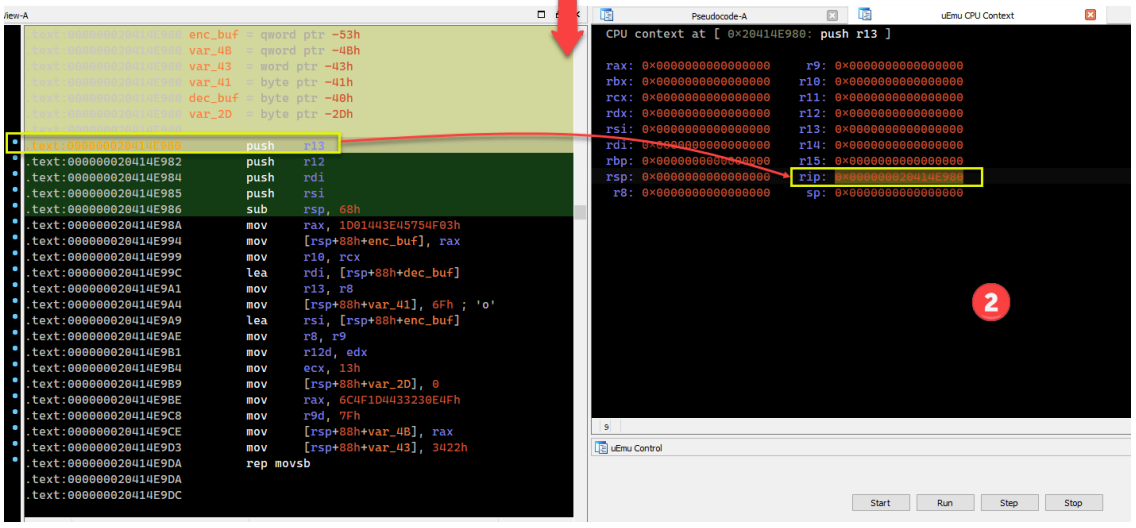
- First, set a breakpoint at the address after the string has been decrypted.

```
.text:000000020414E9DC          movsx   eax, [rsp+rcx+88h+dec_buf]
.text:000000020414E9E1          sub     eax, 6Fh ; 'o'
.text:000000020414E9E4          imul    eax, -23
.text:000000020414E9E7          cdq
.text:000000020414E9E8          idiv    r9d
.text:000000020414E9EB          lea     eax, [rdx+7Fh]
.text:000000020414E9EE          cdq
.text:000000020414E9EF          idiv    r9d
.text:000000020414E9F2          mov     [rsp+rcx+88h+dec_buf], dl
.text:000000020414E9F6          inc     rcx
.text:000000020414E9F9          cmp     rcx, 13h
.text:000000020414E9FD          jnz     short loc_20414E9DC
.text:000000020414E9FF          lea     rdx, [rsp+88h+dec_buf]
.text:000000020414EA04          mov     rcx, r10
.text:000000020414EA07          mov     [rsp+88h+var_60], r8
```

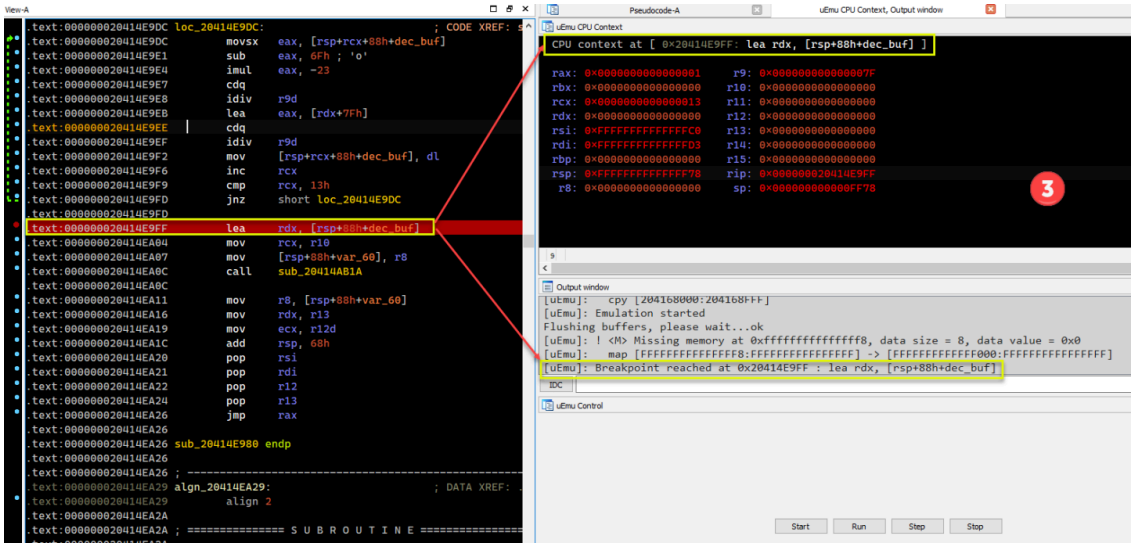
1

- Go to the beginning of the function and select the starting address of the function, then start uEmu. The **CPU Context Edit** window will appear, click **OK** to continue. uEmu will now initialize the emulator. Check the CPU context to see if the address of the `EIP/RIP` register is pointed at the beginning of the function:

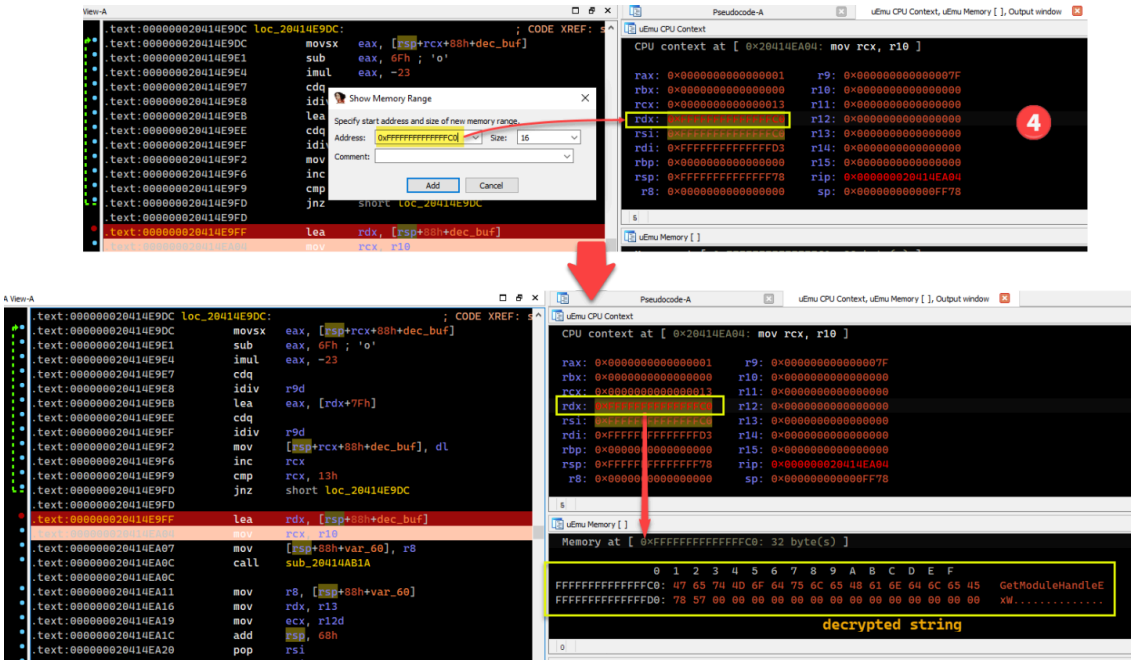




- Then, through **uEmu Control**, you can trace the code by **Step** or **Run** to emulates instructions until breakpoint is reached. During execution, uEmu will ask about unmapped memory, select **No** to continue.



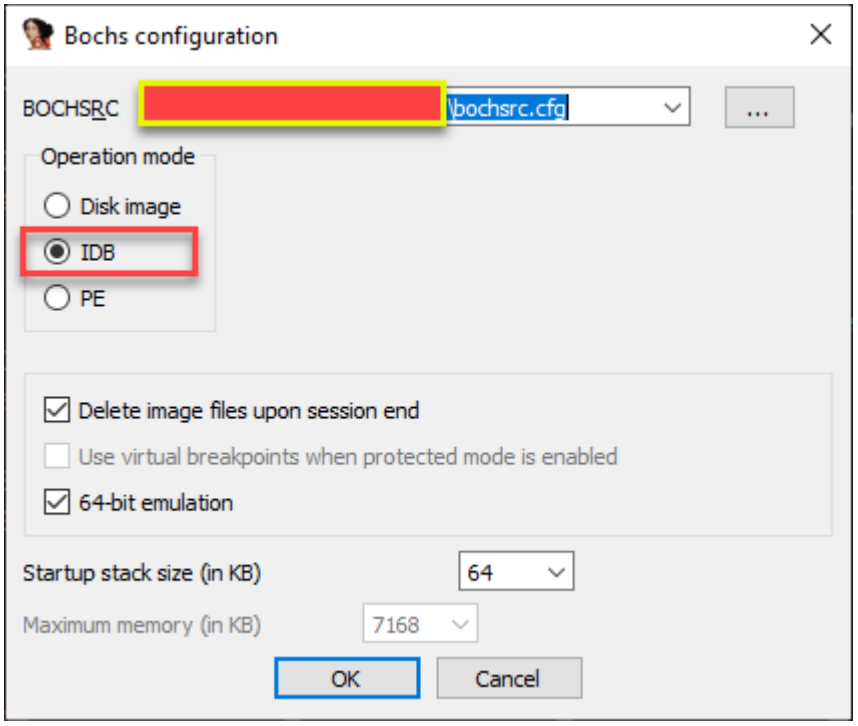
- Press **Step**, to trace over the `lea` command. Then using uEmu's **Show Memory Range** feature, enter the address of the `rdx` register and select **Add**. The result will be similar to the following:



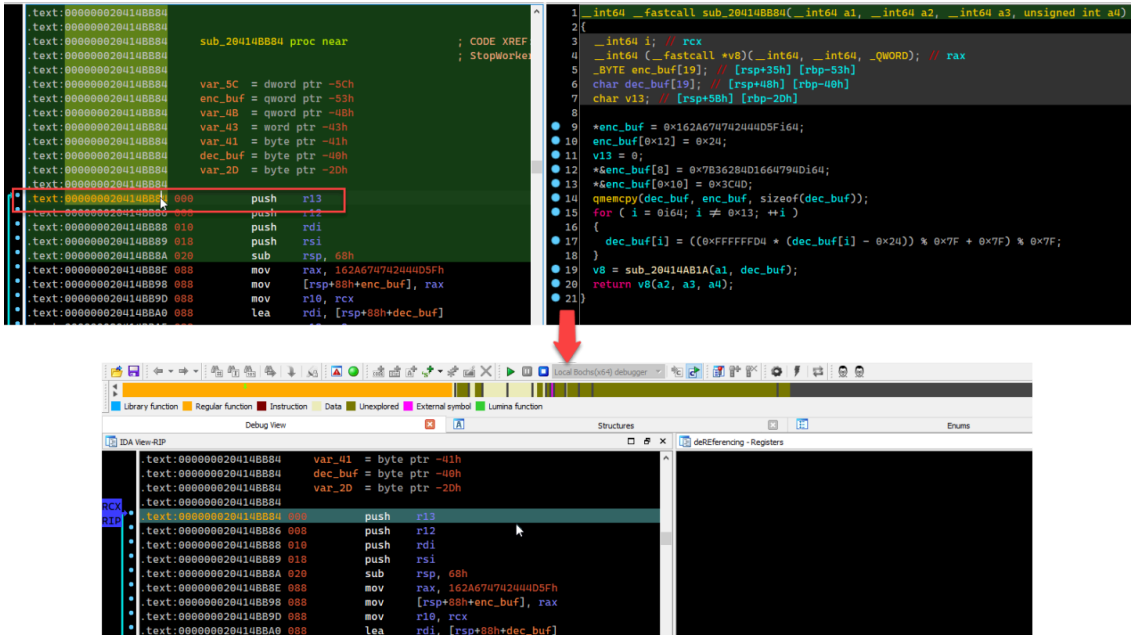
3.3. Debugging with IDA Bochs Plugin

IDA Bochs debugger plugin allows malware researchers to debug malicious code in a safe/emulated environment. For more information please visit [1], [2].

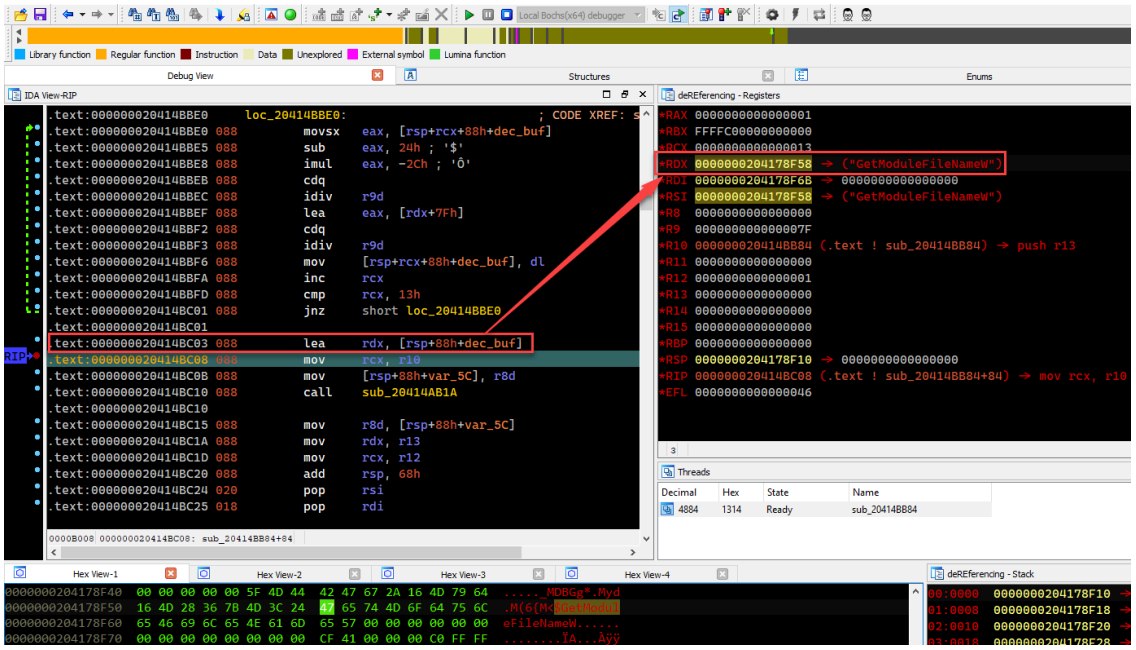
In order to debug the code that decrypt the string, we configure the Bochs plugin to work in **IDB mode**. This mode is used to debug code snippets by simply selecting the code from the database.



Next, select the position or code snippets to debug, then press **F9** to start debugging:



From here you can trace the code as usual or simply set a breakpoint at the address after finished decryting the string and press **F9**. The resulting at `rdx` register will point to the decrypted string as follows:



4. References

- BlackMatter Ransomware v2.0
- Highway to Conti: Analysis of Bazarloader
- BazarLoader infection with Cobalt Strike

End.

m4n0w4r

Source: https://kienmanowar.wordpress.com/2022/02/24/quicknote-techniques-for-decrypting-bazarloader-strings/