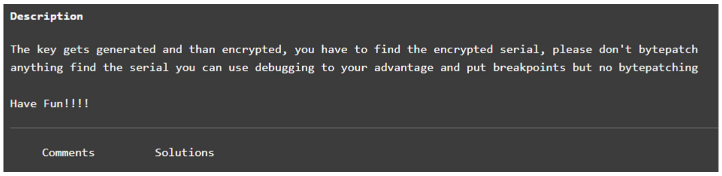
The Deathbringer

Saturday, May 28, 2022 8:27 AM

The program description is as follows…

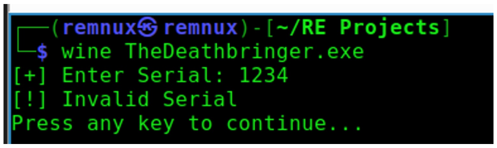


Description

The key gets generated and than encrypted, you have to find the encrypted serial, please don't bytepatch anything find the serial you can use debugging to your advantage and put breakpoints but no bytepatching

Have Fun!!!!

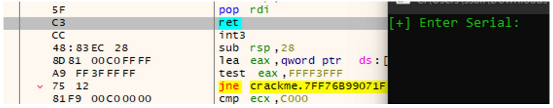
We run the program in our Linux virtual machine…



```
┌──(remnux㉿remnux)-[~/RE Projects]
└─$ wine TheDeathbringer.exe
[+] Enter Serial: 1234
[!] Invalid Serial
Press any key to continue...
```

It looks like the program runs as intended…

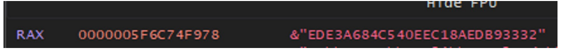
Running this program in windows proper, we find the following…



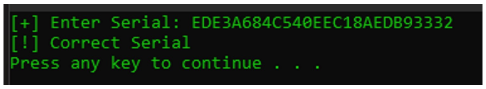
```
5F                    pop  rdi
C3                    ret
CC                    int3
48:83EC 28            sub  rsp,28
8D81 00C0FFFF         lea  eax,qword ptr  ds:[
A9 FF3FFFFF           test  eax,FFFF3FFF
75 12                 jne  crackme.7FF76B99071F
81F9 00C00000         cmp  ecx,C000
```

```
[+] Enter Serial:
```

We find the runtime portion of the program and find that the program retains the key in RAX at runtime…we copy/paste that key into the program and…

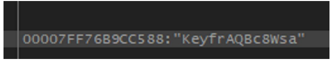


```
RAX     0000005F6C74F978     &"EDE3A684C540EEC18AEDB93332"
```



```
[+] Enter Serial: EDE3A684C540EEC18AEDB93332
[!] Correct Serial
Press any key to continue . . .
```

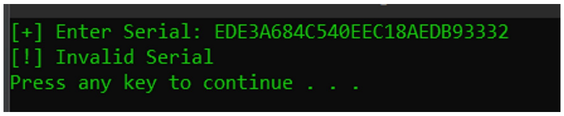
There's a couple of interesting things that help us figure out how this key is made…
Firstly, we find that the program generates this string…



```
00007FF76B9CC588:"KeyfrAQBc8Wsa"
```

KeyfrAQBc8Wsa

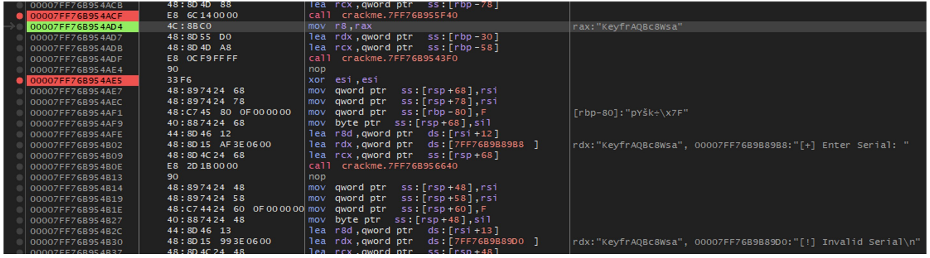
And running the program, back-to-back generates different serials that are only valid in that program instance…for example, they key that worked in the above example will NOT work again



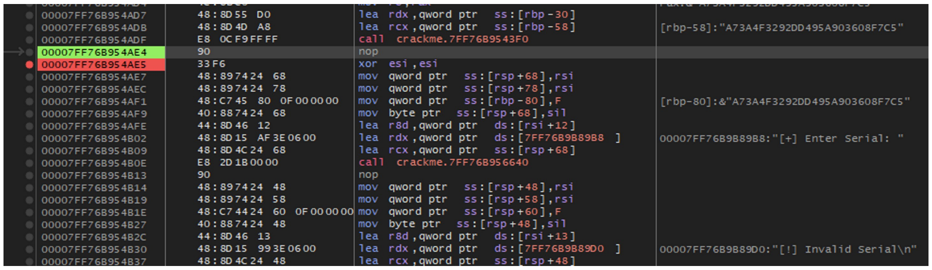
```
[+] Enter Serial: EDE3A684C540EEC18AEDB93332
[!] Invalid Serial
Press any key to continue . . .
```

So that leads us to believe that this program combines the string KeyfrAQBc8Wsa with a time value, and likely other system names/settings when generating a key

Taking a look at the runtime program variable, we notice that until crackme.7FF76B9543F0 is called, the program expects the raw key value to get passed into

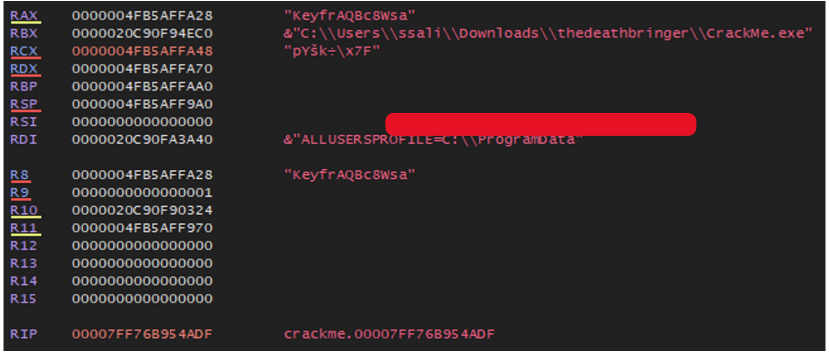


```
00007FF76B954ACB  48:8D4D 88            lea  rcx,qword ptr  ss:[rbp-78]
00007FF76B954ACF  E8 6C140000           call  crackme.7FF76B955F40
00007FF76B954AD4  4C:8BC0               mov  r8,rax                          rax:"KeyfrAQBc8Wsa"
00007FF76B954AD7  48:8D55 D0            lea  rdx,qword ptr  ss:[rbp-30]
00007FF76B954ADB  48:8D4D A8            lea  rcx,qword ptr  ss:[rbp-58]
00007FF76B954ADF  E8 0CF9FFFF           call  crackme.7FF76B9543F0
00007FF76B954AE4  90                    nop
00007FF76B954AE5  33F6                  xor  esi,esi
00007FF76B954AE7  48:897424 68          mov  qword ptr  ss:[rsp+68],rsi
00007FF76B954AEC  48:897424 78          mov  qword ptr  ss:[rsp+78],rsi
00007FF76B954AF1  48:C745 80 0F000000   mov  qword ptr  ss:[rbp-80],F        [rbp-80]:"pYšk÷\x7F"
00007FF76B954AF9  40:887424 68          mov  byte ptr  ss:[rsp+68],sil
00007FF76B954AFE  44:8D46 12            lea  r8d,qword ptr  ds:[rsi+12]
00007FF76B954B02  48:8D15 AF3E0600      lea  rdx,qword ptr  ds:[7FF76B9B89B8 ]  rdx:"KeyfrAQBc8Wsa", 00007FF76B9B89B8:"[+] Enter Serial: "
00007FF76B954B09  48:8D4C24 68          lea  rcx,qword ptr  ss:[rsp+68]
00007FF76B954B0E  E8 2D1B0000           call  crackme.7FF76B956640
00007FF76B954B13  90                    nop
00007FF76B954B14  48:897424 48          mov  qword ptr  ss:[rsp+48],rsi
00007FF76B954B19  48:897424 58          mov  qword ptr  ss:[rsp+58],rsi
00007FF76B954B1E  48:C74424 60 0F000000 mov  qword ptr  ss:[rsp+60],F
00007FF76B954B27  40:887424 48          mov  byte ptr  ss:[rsp+48],sil
00007FF76B954B2C  44:8D46 13            lea  r8d,qword ptr  ds:[rsi+13]
00007FF76B954B30  48:8D15 993E0600      lea  rdx,qword ptr  ds:[7FF76B9B89D0 ]  rdx:"KeyfrAQBc8Wsa", 00007FF76B9B89D0:"[!] Invalid Serial\n"
00007FF76B954B37  48:8D4C24 48          lea  rcx,qword ptr  ss:[rsp+48]
```



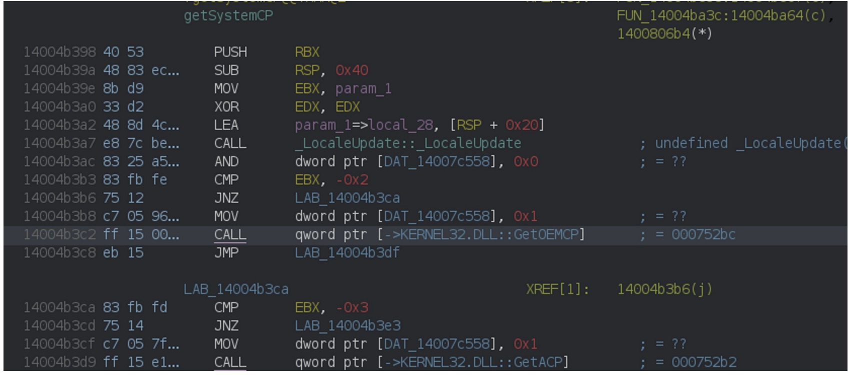
```
00007FF76B954AD7   48:8D55 D0          lea rdx,qword ptr ss:[rbp-30]
00007FF76B954ADB   48:8D4D A8          lea rcx,qword ptr ss:[rbp-58]        [rbp-58]:"A73A4F3292DD495A903608F7C5"
00007FF76B954ADF   E8 0CF9FFFF         call crackme.7FF76B9543F0
00007FF76B954AE4   90                  nop
00007FF76B954AE5   33F6                xor esi,esi
00007FF76B954AE7   48:897424 68        mov qword ptr ss:[rsp+68],rsi
00007FF76B954AEC   48:897424 78        mov qword ptr ss:[rsp+78],rsi
00007FF76B954AF1   48:C745 80 0F000000 mov qword ptr ss:[rbp-80],F          [rbp-80]:&"A73A4F3292DD495A903608F7C5"
00007FF76B954AF9   40:887424 68        mov byte ptr ss:[rsp+68],sil
00007FF76B954AFE   44:8D46 12          lea r8d,qword ptr ds:[rsi+12]
00007FF76B954B02   48:8D15 AF3E0600    lea rdx,qword ptr ds:[7FF76B9B89B8 ] 00007FF76B9B89B8:"[+] Enter Serial: "
00007FF76B954B09   48:8D4C24 68        lea rcx,qword ptr ss:[rsp+68]
00007FF76B954B0E   E8 2D1B0000         call crackme.7FF76B956640
00007FF76B954B13   90                  nop
00007FF76B954B14   48:897424 48        mov qword ptr ss:[rsp+48],rsi
00007FF76B954B19   48:897424 58        mov qword ptr ss:[rsp+58],rsi
00007FF76B954B1E   48:C74424 60 0F000000 mov qword ptr ss:[rsp+60],F
00007FF76B954B27   40:887424 48        mov byte ptr ss:[rsp+48],sil
00007FF76B954B2C   44:8D46 13          lea r8d,qword ptr ds:[rsi+13]
00007FF76B954B30   48:8D15 993E0600    lea rdx,qword ptr ds:[7FF76B9B89D0 ] 00007FF76B9B89D0:"[!] Invalid Serial\n"
00007FF76B954B37   48:8D4C24 48        lea rcx,qword ptr ss:[rsp+48]
```

Taking a close look at the call to crackme.7FF76B9543F0, we know that when the generate_serial_number function is called these are the status of the registers…



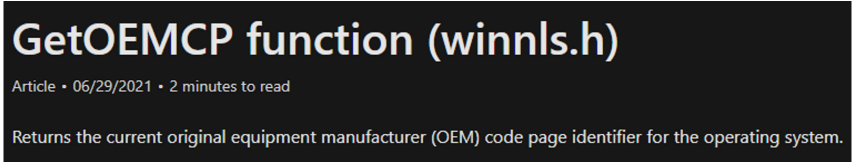
```
RAX   0000004FB5AFFA28   "KeyfrAQBc8Wsa"
RBX   0000020C90F94EC0   &"C:\\Users\\ssali\\Downloads\\thedeathbringer\\CrackMe.exe"
RCX   0000004FB5AFFA48   "pYšk÷\x7F"
RDX   0000004FB5AFFA70
RBP   0000004FB5AFFAA0
RSP   0000004FB5AFF9A0
RSI   0000000000000000   [redacted]
RDI   0000020C90FA3A40   &"ALLUSERSPROFILE=C:\\ProgramData"

R8    0000004FB5AFFA28   "KeyfrAQBc8Wsa"
R9    0000000000000001
R10   0000020C90F90324
R11   0000004FB5AFF970
R12   0000000000000000
R13   0000000000000000
R14   0000000000000000
R15   0000000000000000

RIP   00007FF76B954ADF   crackme.00007FF76B954ADF
```

Secondly, We find that prior to the program generating the serial, a couple of interesting call to getOEMCP and GetACP



```
                    getSystemCP                                          FUN_14004ba3c:14004ba64(c),
                                                                         1400806b4(*)
  14004b398 40 53        PUSH     RBX
  14004b39a 48 83 ec...  SUB      RSP, 0x40
  14004b39e 8b d9        MOV      EBX, param_1
  14004b3a0 33 d2        XOR      EDX, EDX
  14004b3a2 48 8d 4c...  LEA      param_1=>local_28, [RSP + 0x20]
  14004b3a7 e8 7c be...  CALL     _LocaleUpdate::_LocaleUpdate               ; undefined _LocaleUpdate(
  14004b3ac 83 25 a5...  AND      dword ptr [DAT_14007c558], 0x0            ; = ??
  14004b3b3 83 fb fe     CMP      EBX, -0x2
  14004b3b6 75 12        JNZ      LAB_14004b3ca
  14004b3b8 c7 05 96...  MOV      dword ptr [DAT_14007c558], 0x1            ; = ??
  14004b3c2 ff 15 00...  CALL     qword ptr [->KERNEL32.DLL::GetOEMCP]      ; = 000752bc
  14004b3c8 eb 15        JMP      LAB_14004b3df

                    LAB_14004b3ca                             XREF[1]:   14004b3b6(j)
  14004b3ca 83 fb fd     CMP      EBX, -0x3
  14004b3cd 75 14        JNZ      LAB_14004b3e3
  14004b3cf c7 05 7f...  MOV      dword ptr [DAT_14007c558], 0x1            ; = ??
  14004b3d9 ff 15 e1...  CALL     qword ptr [->KERNEL32.DLL::GetACP]        ; = 000752b2
```

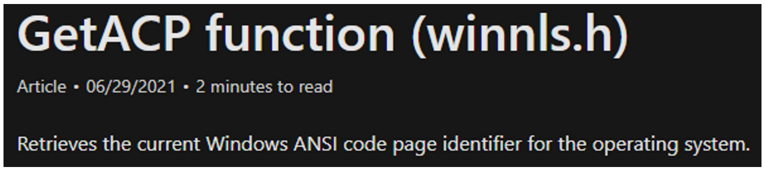
Looking at some documentation…



GetOEMCP function (winnls.h)

Article • 06/29/2021 • 2 minutes to read

Returns the current original equipment manufacturer (OEM) code page identifier for the operating system.



GetACP function (winnls.h)

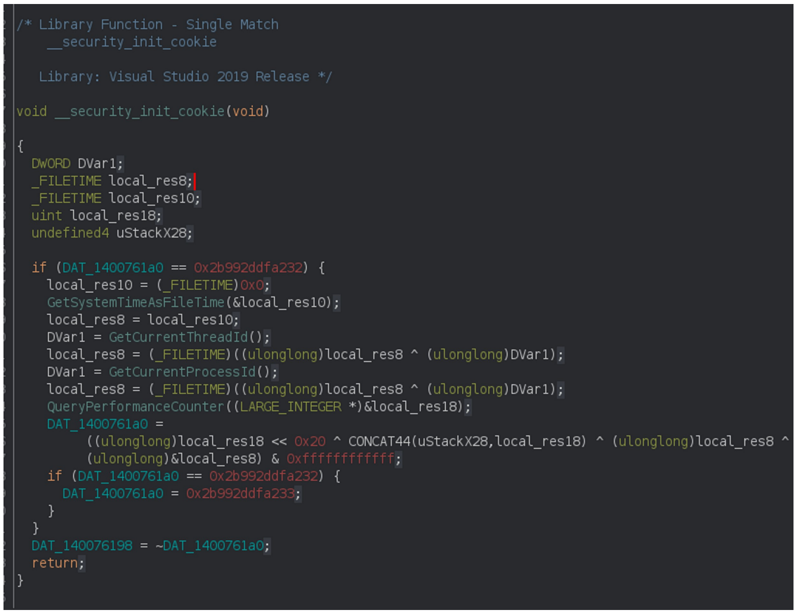
Article • 06/29/2021 • 2 minutes to read

Retrieves the current Windows ANSI code page identifier for the operating system.

But even these two functions don't account for the different serial numbers at run time. Neither of these inputs should change at runtime

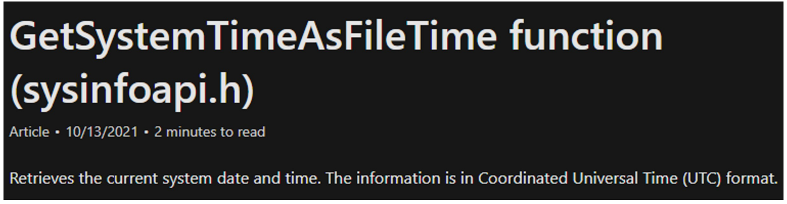
Digging deeper, we find a security init cookie that holds a lot of details about the runtime environment.

GetSystemTimeAsFileTime, GetCurrentProcessId, and GetCurrentThreadId are all variables that get determined at run time…
so if we can bypass these we should get a reusable serial number



```
/* Library Function - Single Match
    __security_init_cookie

   Library: Visual Studio 2019 Release */

void __security_init_cookie(void)

{
  DWORD DVar1;
  _FILETIME local_res8;
  _FILETIME local_res10;
  uint local_res18;
  undefined4 uStackX28;

  if (DAT_1400761a0 == 0x2b992ddfa232) {
    local_res10 = (_FILETIME)0x0;
    GetSystemTimeAsFileTime(&local_res10);
    local_res8 = local_res10;
    DVar1 = GetCurrentThreadId();
    local_res8 = (_FILETIME)((ulonglong)local_res8 ^ (ulonglong)DVar1);
    DVar1 = GetCurrentProcessId();
    local_res8 = (_FILETIME)((ulonglong)local_res8 ^ (ulonglong)DVar1);
    QueryPerformanceCounter((LARGE_INTEGER *)&local_res18);
    DAT_1400761a0 =
         ((ulonglong)local_res18 << 0x20 ^ CONCAT44(uStackX28,local_res18) ^ (ulonglong)local_res8 ^
         (ulonglong)&local_res8) & 0xffffffffffff;
    if (DAT_1400761a0 == 0x2b992ddfa232) {
      DAT_1400761a0 = 0x2b992ddfa233;
    }
  }
  DAT_140076198 = ~DAT_1400761a0;
  return;
}
```

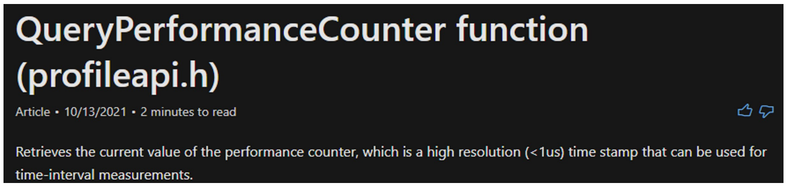
And the documentation confirms what we've suspected…



GetSystemTimeAsFileTime function (sysinfoapi.h)

Article • 10/13/2021 • 2 minutes to read

Retrieves the current system date and time. The information is in Coordinated Universal Time (UTC) format.

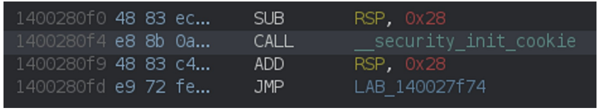


QueryPerformanceCounter function (profileapi.h)

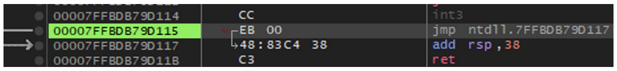
Article • 10/13/2021 • 2 minutes to read

Retrieves the current value of the performance counter, which is a high resolution (<1us) time stamp that can be used for time-interval measurements.

Establishing this cookie, is actually one of the first things the program does after entry



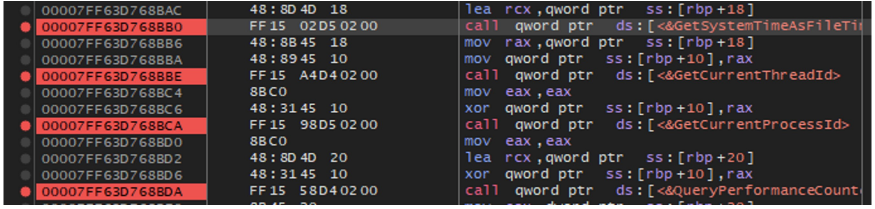
```
1400280f0 48 83 ec...    SUB     RSP, 0x28
1400280f4 e8 8b 0a...    CALL    __security_init_cookie
1400280f9 48 83 c4...    ADD     RSP, 0x28
1400280fd e9 72 fe...    JMP     LAB_140027f74
```



```
00007FFBDB79D114    CC             int3
00007FFBDB79D115    EB 00          jmp  ntdll.7FFBDB79D117
00007FFBDB79D117    48:83C4 38     add  rsp,38
00007FFBDB79D11B    C3             ret
```

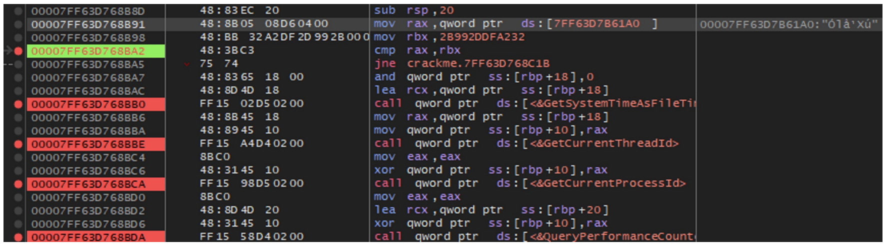
If our theory is correct, we should be able to force the program to require the same key twice by changing the data stored at the security cookie address

We set breakpoints at the appropriate positions with the goal of skipping all of this generation and get a repeatable serial number…



```
00007FF63D768BAC    48:8D4D 18       lea rcx,qword ptr ss:[rbp+18]
00007FF63D768BB0    FF15 02D50200    call qword ptr ds:[<&GetSystemTimeAsFileTi
00007FF63D768BB6    48:8B45 18       mov rax,qword ptr ss:[rbp+18]
00007FF63D768BBA    48:8945 10       mov qword ptr ss:[rbp+10],rax
00007FF63D768BBE    FF15 A4D40200    call qword ptr ds:[<&GetCurrentThreadId>
00007FF63D768BC4    8BC0             mov eax,eax
00007FF63D768BC6    48:3145 10       xor qword ptr ss:[rbp+10],rax
00007FF63D768BCA    FF15 98D50200    call qword ptr ds:[<&GetCurrentProcessId>
00007FF63D768BD0    8BC0             mov eax,eax
00007FF63D768BD2    48:8D4D 20       lea rcx,qword ptr ss:[rbp+20]
00007FF63D768BD6    48:3145 10       xor qword ptr ss:[rbp+10],rax
00007FF63D768BDA    FF15 58D40200    call qword ptr ds:[<&QueryPerformanceCount
```

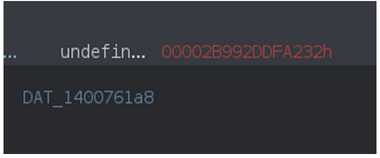
BUT WAIT, we notice one more check just before running into these checks…



```
00007FF63D768B8D   48:83EC 20                 sub rsp,20
00007FF63D768B91   48:8B05 08D60400           mov rax,qword ptr ds:[7FF63D7B61A0 ]     00007FF63D7B61A0:"Ólà'Xú"
00007FF63D768B98   48:BB 32A2DF2D992B0000     mov rbx,2B992DDFA232
00007FF63D768BA2   48:3BC3                    cmp rax,rbx
00007FF63D768BA5   75 74                      jne crackme.7FF63D768C1B
00007FF63D768BA7   48:8365 18 00              and qword ptr ss:[rbp+18],0
00007FF63D768BAC   48:8D4D 18                 lea rcx,qword ptr ss:[rbp+18]
00007FF63D768BB0   FF15 02D50200              call qword ptr ds:[<&GetSystemTimeAsFileTi
00007FF63D768BB6   48:8B45 18                 mov rax,qword ptr ss:[rbp+18]
00007FF63D768BBA   48:8945 10                 mov qword ptr ss:[rbp+10],rax
00007FF63D768BBE   FF15 A4D40200              call qword ptr ds:[<&GetCurrentThreadId>
00007FF63D768BC4   8BC0                       mov eax,eax
00007FF63D768BC6   48:3145 10                 xor qword ptr ss:[rbp+10],rax
00007FF63D768BCA   FF15 98D50200              call qword ptr ds:[<&GetCurrentProcessId>
00007FF63D768BD0   8BC0                       mov eax,eax
00007FF63D768BD2   48:8D4D 20                 lea rcx,qword ptr ss:[rbp+20]
00007FF63D768BD6   48:3145 10                 xor qword ptr ss:[rbp+10],rax
00007FF63D768BDA   FF15 58D40200              call qword ptr ds:[<&QueryPerformanceCount
```

Running the program a couple more times, we notice that the value that gets stored in rax is NOT the same per instance

Doing a little more static analysis we notice that the correct value is INITIALLY in the data location, but it get overwritten at some point…



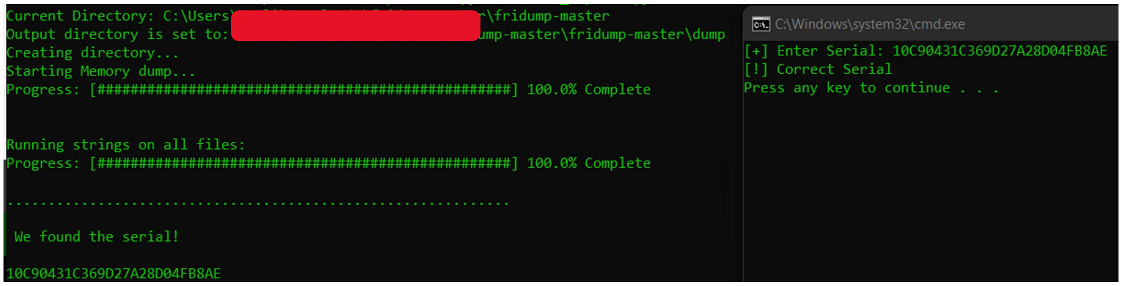
```
...   undefin...  00002B992DDFA232h

DAT_1400761a8
```

Likely because the .data section is where globals are tracked, at some point getting to entry the program assigned the .data section and the data allocated at our address is arbitrary

Exploit

Windows exploits are strange because (free) tools are largely underdeveloped. After a lot of searching I found this free tool that dumps program memory. We know our program retains the valid serial when it asks us for the valid serial. So I'm going to extend this python script and get us a one-stop-shop solution to this crackme

https://github.com/Nightbringer21/fridump



```
Current Directory: C:\Users\[redacted]r\fridump-master
Output directory is set to: ump-master\fridump-master\dump
Creating directory...
Starting Memory dump...
Progress: [##################################################] 100.0% Complete


Running strings on all files:
Progress: [##################################################] 100.0% Complete

.............................................................

 We found the serial!

10C90431C369D27A28D04FB8AE
```

C:\Windows\system32\cmd.exe

```
[+] Enter Serial: 10C90431C369D27A28D04FB8AE
[!] Correct Serial
Press any key to continue . . .
```

After a lot of tweaking, we make a script that that consistently cracks the serial to this crackme

```
import textwrap
import frida
import os
import sys
import frida.core
import dumper
import utils
import argparse
import logging
import subprocess
from time import sleep
from itertools import islice

## Define Configurations

APP_NAME = "CrackMe.exe"
DIRECTORY = ""
DEBUG_LEVEL = logging.INFO
MAX_SIZE = 20971520
PERMS = 'rw-'

## Start Process
try:
    os.system("rmdir /s /q dump")
except:
    pass
output = subprocess.Popen('start cmd /C CrackMe.exe', shell=True)

sleep(2)
## Creating Directory to Dump Program Memory
print("Current Directory: " + str(os.getcwd()))
DIRECTORY = os.path.join(os.getcwd(), "dump")
print("Output directory is set to: " + DIRECTORY)
if not os.path.exists(DIRECTORY):
    print("Creating directory...")
    os.makedirs(DIRECTORY)

mem_access_viol = ""

print("Starting Memory dump...")

session = frida.attach(APP_NAME)
script = session.create_script(
    """'use strict';

    rpc.exports = {
      enumerateRanges: function (prot) {
        return Process.enumerateRangesSync(prot);
      },
      readMemory: function (address, size) {
        return Memory.readByteArray(ptr(address), size);
      }
    };

    """)

script.on("message", utils.on_message)
script.load()

agent = script.exports
ranges = agent.enumerate_ranges(PERMS)

i = 0
l = len(ranges)

# Performing the memory dump
for range in ranges:
    base = range["base"]
    size = range["size"]

    logging.debug("Base Address: " + str(base))
    logging.debug("")
    logging.debug("Size: " + str(size))


    if size > MAX_SIZE:
        logging.debug("Too big, splitting the dump into chunks")
        mem_access_viol = dumper.splitter(
            agent, base, size, MAX_SIZE, mem_access_viol, DIRECTORY)
        continue
    mem_access_viol = dumper.dump_to_file(
        agent, base, size, mem_access_viol, DIRECTORY)
    i += 1
    utils.printProgress(i, l, prefix='Progress:', suffix='Complete', bar=50)
print("")

files = os.listdir(DIRECTORY)
i = 0
l = len(files)
print("Running strings on all files:")
for f1 in files:
    utils.strings(f1, DIRECTORY)
    i += 1
    utils.printProgress(i, l, prefix='Progress:', suffix='Complete', bar=50)

f = open(".\dump\strings.txt", 'r')

for line in f:
    if len(line) >= 27 and "=" not in line and "?" not in line and "@" not in line and " " not in line and "abcdefghijklmnopqrstuvwxyz" not in line and "ABCDEFGHIJKLMNOPQRSTUVWXYZ" not in line:
        print("\n\n We found the serial! \n")
        print(line, end="\n\n")
        quit()
    else:
        print(".", end="")

print("\nWe couldn't find the serial...exiting now")
quit()
```