CMSC 491/691 Malware Analysis HW 3

Name: Daniel Roh
Assigned: 2/19/2020
Due: 3/2/2020 by 5:00pm

How to turn this in for grading: You can edit your answers right into this file. Submit the homework a .docx or .pdf to RJ via email.
Hint: Chapters 4 and 6 of your Practical Malware Analysis textbook are very useful references!

Part 1 (32 pts)

```
start:
      PUSH   EBP                 //Store base pointer
      MOV    EBP, ESP            //Move the base pointer to new location
      MOV    EDI, [EBP+arg_0]    //Move a variable into EDI
      XOR    EAX, EAX            //Zero out EAX
      MOV    ECX, 0xFFFFFFFF     //Put counter value of FFFFFFF in to ECX
      REPNE SCASB                //Look through EDI (arg_0) for 0
      NEG    ECX                 //Two complement Negation making ECX the number of bytes
      MOV    EAX, ECX            //Place the size found in to EAX
      MOV    ESI, [EBP+arg_0]    //Place pointer arg_0 into ESI
      MOV    EDI, [EBP+arg_4]    //Place pointer for arg_4 into EDI
      REP MOVSB                  //Copy the bytes from ESI to EDI
      MOV    ESP, EBP            //Return stack pointer to the original location
      POP    EBP                 //Return base pointer to original
      RETN
```

1.1) In a few sentences, explain what this function does. (10 pts)

This function starts by storing the caller's stack locations via the CEDECL format. Then takes arg_0 and finds the length of that variable. Once the variable length is found. The function copies arg_0 to arg_4. Then returns the original stack variables to the original locations

1.2) Write a function in C that is equivalent to the assembly above. (10 pts)

```
int func1(char* arg_0, char* arg_4) {
        arg_4 = arg_0;
        return (sizeof(arg_0)/sizeof(char*));
}
```

1.3) Let arg_0 be a pointer to the null-terminated string "C:\Windows\System32\" and let arg_4 be a pointer to an empty buffer. What is the value of the buffer pointed to by arg_4 when the function completes? What value does the function return? (12 pts)

arg_4 will be C:\Windows\System32\
The return will be: 20

Part 2 (34 pts)

```
start:
      PUSH   EBP                              //Store base pointer
      MOV    EBP, ESP                         //Move base pointer to new base
      MOV    ECX, [EBP+arg_0]                 //Place arg_0 into ECX
      MOV    ESI, [EBP+arg_4]                 //Place arg_4 into ESI
      MOV    [EBP+var_1], 0                   //Have a pointer that is 0
      JMP    loc_2                            //move to loc_2
loc_1:
      MOV    EAX, [EBP+var_1]                 //Place pointed variable to EAX
      ADD    EAX, ECX                         //Add
      MOV    EDX, byte ptr [EAX]              //Grab a one byte of EAX and place in EDX
      XOR    EDX, ESI                         //XOR the byte with arg_4
      MOV    [EAX], DL                        //Store back only the lower 8 byte
      ADD    [EBP+var_1], 0x1                 //Add 1 to var_1
loc_2:
      MOV    EAX, [EBP+var_1]                 //Move iterator
      CMP    byte ptr [ECX + EAX], 0          //See if ECX + EAX is pointing to 0
      JNZ    loc_1                            //Check if ZF is cleared. If not go to
loc_1
      MOV    ESP, EBP                         //Return stack pointer back to original
      POP    EBP                              //Return base pointer to original
      RETN
```

2.1) In a few sentences, explain what this function does. (10 pts)

This function starts by storing the caller's stack locations via the CEDECL format. Then takes uses a variable var_1 as a counter. A check is made to see if the zero flag is set, if not then the function does a for loop and takes the var_1 and adds it with arg_0. Then the character at arg_0[x] is taken and xor'ed with arg_4 and the lower 8 bytes of the result is taken and stored as the new arg_0. Basically, this function is decrypting a byte string.

2.2) Write a function in C that is equivalent to the assembly above. (12 pts)

```
int func2(char* arg_0, int arg_4) {
        unsigned int x = 0;

        do{
            char temp = arg_0[x] ^ arg_4; //XOR
            temp = temp & 0xFF; //Get lower 8 bytes
            printf("%c", temp); //print out characters
            x++;
          }while(arg_0[x] != NULL);

          return x;
}
```

2.3) Let arg_0 be a pointer to the null-terminated string "\xa7\xa4\xe2\xaf\xcf\xd2\xc6\xf1\xe1\xe3\xf3\xcc\xaf\xd8\xef\xdb\xf1\xe1\xa3\xa7\xaf\xe6\xa1\xd7\xd7\xae\xff\xe5\xfc\xe4\xa0\xc5\xdb\xe1" and let arg_4 be the integer 0x96. What is the value of the string pointed to by arg_0 when the function completes? (*Hint: It will decode to a bitcoin wallet*) What value does the function return? (12 pts)

Value of string: 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
Function returns: 34

Part 3 (34 pts)

```
start:
            push   ebp                      //Place base pointer to stack
            mov    ebp,esp                  //Move stack pointer to base
            sub    esp,0x10                 //Subtract 16 from esp
            mov    eax,DWORD PTR [esp]      //Place esp in eax (ignore this, compiler)
            add    eax,0x2e48               //Add 11848 to eax (ignore this, compiler)
mov    DWORD PTR [ebp-0x8],0x30             //Place 48 as a local variable2
mov    DWORD PTR [ebp-0x4],0x0              //Place 0 as a local variable1
jmp    loc_2                                //Go to loc_2

loc_1:
            mov    edx,DWORD PTR [ebp-0x4] //Move var1 in edx
            mov    eax,DWORD PTR [ebp+0x8] //Move arg0 in eax
            add    eax,edx                 //Get the next char address
            movzx  eax,BYTE PTR [eax]      //Add 0 to the value of eax (0 ext)
            movsx  eax,al                  //Take the lower byte and add 0s (sign ext)
                                           //The mov's grabs the 1 char from string
            xor    DWORD PTR [ebp-0x8],eax //XOR var2 with the char at var1
            add    DWORD PTR [ebp-0x4],0x1 //Adds 1 to the value at var1

loc_2:
            mov    eax,DWORD PTR [ebp-0x4] //Put var1 (starts at 0) in eax
            cmp    eax,DWORD PTR [ebp+0xc] //Compare var1 with arg_4 (0 < x)
            jl     loc_1                   //Move to loc_1 if eax is less

loc_3:
            mov    eax,DWORD PTR [ebp-0x8] //Place var2 to eax
            leave                          //Return the stack to original
            ret
```

3.1) In a few sentences, explain what this function does. (10 pts)

This function takes a char string and an int value and goes through the string xor'ing the characters with the numbers to get a resulting int. Should result in a unique int value given a string and the size of the string.

3.2) Write a function in C that is equivalent to the assembly above. (12 pts)

```
int func3(char* arg_0, int arg_4) {
        int var2 = 30;
        int var1 = 0;

        do{
                var2  = arg_0[var1] ^ var2;
                var1++;
        }while(var1 < arg_4);
```

```
        return var2;
}
```

3.3) Let arg_0 be a pointer to the string "x64 is better than x86" and let arg_4 be 22. What does the function return? (12 pts)

The function returns: 11