Caricamento ed esecuzione di uno shellcode in memoria RAM

Per caricare ed eseguire del codice in memoria RAM abbiamo bisogno principalmente di 4 semplici istruzioni:

- -VirtualAlloc
- -RtlMoveMemory
- -CreateThread
- -WaitForSingleObject

Andiamo dunque a vedere tutte le istruzioni nel dettaglio.
La funzione VirtualAlloc serve ad allocare un area di memoria RAM che andremo successivamente ad utilizzare per caricare il nostro codice malevolo.
VirtualAlloc richiede alcuni parametri che andremo a spiegare brevemente (per maggiori informazioni vi invito a leggere la documentazione ufficiale delle singole funzioni).

-lpAddres di tipo LPVOID: questo parametro è opzionale e indica l'indirizzo di inizio dell'area di memoria che vogliamo allocare, se settato a NULL sarà il sistema a scegliere dove allocare l'area.

-dwSize di tipo SIZE_T: questo è un parametro obbligatorio ed indica la grandezza in bytes dell'area che vogliamo allocare.

-flAllocationType di tipo DWORD: questo è un parametro particolare che non spiegherò, basti sapere che nel nostro caso dobbiamo settarlo a MEM_COMMIT|MEM_RESERVE che per necessità tradurremo in 0x3000.

-flProtect di tipo DWORD: con questo parametro settiamo i permessi della nostra regione di memoria; nel nostro caso lo settiamo a 0x40 ovvero PAGE_EXECUTE_READWRITE.

La nostra funzione ci restituirà l'indirizzo di inizio della nostra area di memoria in caso di successo o NULL in caso di insuccesso; possiamo sfruttare questi valori di ritorno per implementare dei controlli.

Andiamo a vedere adesso la seconda funzione fondamentale: RtlMoveMemory. Questa funzione ci serve per inserire il nostro codice "malevolo" all'interno dell'area di memoria che allocheremo con la funzione precedentemente spiegata.
Vediamo i parametri della funzione:

-destination: con questo parametro indichiamo un puntatore all'inizio dell'area di memoria che allocheremo.

-source: un puntatore (non necessariamente) al blocco di memoria contenente i nostri bytes.

-Lenght di tipo SIZE_T: una variabile che rappresenta la grandezza in bytes della nostra area di memoria.

Adesso vediamo la funzione CreateThread, che come ci fa intuire il nome andrà a creare un Thread ovvero un sottoprocesso per l'esecuzione del codice che avremo caricato in RAM.

-lpThreadAttributes: questo parametrò è opzionale e non andrò a spiegarlo, nel nostro caso lo andremo a settare a NULL.

-dwStackSize di tipo SIZE_T: questo parametro serve a dichiarare la dimensione iniziale del nostro stack, noi lo setteremo a 0, in questo modo la dimensione sarò quella di default per gli eseguibili.

-lpStartAddress: questo sarà un puntatore all'inizio dell'area di memoria precedentemente istanziata.

-lpParameter: questo parametro non ci interessa, è opzionale e lo setteremo su NULL.

-dwCreationFlag di tipo DWORD: questa flag serve a dare istruzioni alla creazione del thread, nel nostro caso settandolo su 0 andremo ad indicare che il thread dovrà partire immediatamente dopo la sua creazione.

-lpThreadId di tipo LPWORD: questo è un parametro opzionale, qui passeremo l'indirizzo di una variabile di tipo DWORD che conterrà l'id del thread creato, se fosse stato NULL l'id del thread non sarebbe stato restituito.

Se la funzione viene richimata ed eseguita con successo restituirà un istanza di tipo HANDLE che ci servirà per l'invocazione della prossima istruzione.

L'ultima funzione che andiamo a vedere è WaitForSingleObject, tramite questa funzione andremo ad indicare al processo possessore del thread creato di non uscire fino a quando il thread non sarà completamente eseguito. Vediamo ora i 2 parametri della funzione.

-hHandle di tipo HANDLE: qui passeremo l'handle del thread in causa. Per chi non sapesse cos'è un handle si tratta semplicemente dell'istanza di un oggetto tramite la quale gestire gli attributi e i metodi dell'oggetto in questione.

-dwMilliseconds di tipo DWORD: questo è un timeout dopo il quale la funzione restituirà un codice che solitamente è utilizzato capire il motivo della terminazione del thread passato precedentemente come parametro. Nel nostro caso lo setteremo a INFINITE.

PERFETTO!
Abbiamo finito di spiegare le funzioni che andremo ad utilizzare. Adesso passiamo all'esempio concreto. Incollerò qui sotto il codice del programma.

```
#include <string.h>
#include <windows.h>
#include <stdio.h>

int main(){

	LPVOID lpvAddress;
	HANDLE handle;
	DWORD dwWaitResult;
```

```
        DWORD threadId;

unsigned char buffer[]="il nostro shellcode qui";
lpvAddress = VirtualAlloc(NULL,strlen(buffer),0x3000,0x40);
if(lpvAddress != NULL){
        RtlMoveMemory(lpvAddress,buffer,strlen(buffer));
        handle = CreateThread(NULL,0,lpvAddress,NULL,0,&threadId);
        if(handle != NULL){
                dwWaitResult(handle,INFINITE);
        }else{
                DWORD errCode = GetLastError();
                printf("%lu",errCode);
        }
}

return 0;

}
```

FINE.