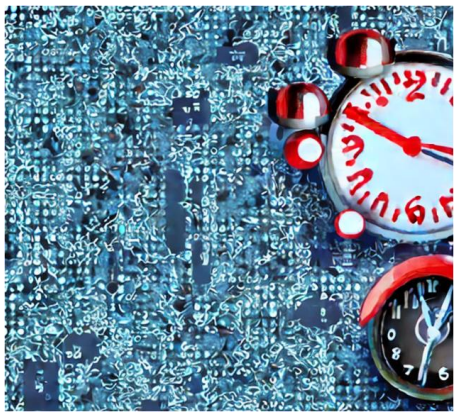
TIMESTOMPING EXPLAINED ON API LEVEL



This is my first blog so please do let me know if I made any mistakes or any concepts can be improved.

Inspiration:
While working on unpacking a malware, I encountered a challenge that I have yet to overcome as I write this. However, upon successfully executing the malware, I noticed that it had dropped a file onto the disk with a different timestamp. I knew Time Stomping is used but this made me curious to explore the concept on lower level and examining how malware utilizes it to manipulate timestamps and how we can determine the original timestamp set for a file.

What is Time Stomping?
Time Stomping refers to the act of altering the file time attributes of a file. This technique poses a challenge for investigating analysts or responders, as they might overlook it due to the manipulated timestamp, which appears older than when the system was infected. For more information on Time Stomping, you can refer to the MITRE ATT&CK® knowledge base:
https://attack.mitre.org/techniques/T1070/006/

Fortunately, there are various forensic tools available to retrieve the real timestamp of a file. One such tool is the **Get-ForensicFileRecord** cmdlet from the Power Forensics Module. This cmdlet allows investigators to extract and parse information directly from the Master File Table (MFT) record associated with a specific file or files within the NTFS file system. You can find more details about the Get-ForensicFileRecord cmdlet here:
https://powerforensics.readthedocs.io/en/latest/modulehelp/Get-ForensicFileRecord/

When malware employs the time stomping technique, it typically modifies the $STANDARD_INFORMATION attribute in the MFT, while the $FILE_NAME attribute still retains information about the real timestamp. It is worth noting that modifying the timestamps in the $FILE_NAME attribute requires a lower-level approach, which I have not encountered in the handful malware samples that I have reversed.

Figure 1 illustrates an example of how the file timestamp appears in Windows Explorer compared to the timestamp obtained from the MFT using the Get-ForensicFileRecord tool.

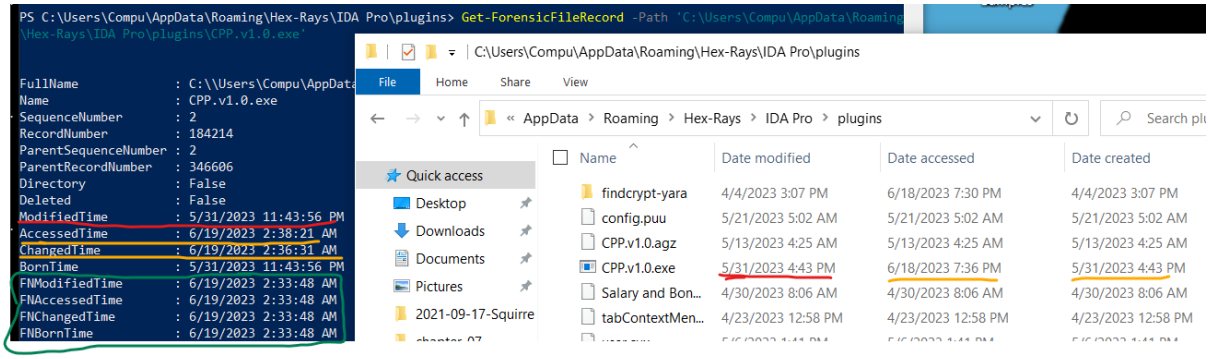


Figure 1: Displaying Time Stomping

In Figure 1, we can observe that the malware has altered the modified, accessed, and created timestamps (i.e., the $STANDARD_INFORMATION attributes). However, by utilizing the Get-ForensicFileRecord tool, we can still view the original timestamp (labeled as "FN" for $FILE_NAME attribute, highlighted in green).

Enough boring stuff let's see it in actions:
There are multiple methods to change a file's timestamp, but ultimately, they all make use of the **NtSetInformationFile** API. This API is part of the Native API and serves as a lower-level function that directly interacts with the internal structures of Windows to modify file attributes, including timestamps. To gain a better understanding, let's start with a higher-level API and follow the path down to the lower-level API.

SetFileTime
One such higher-level API is SetFileTime. When you use the SetFileTime API or any other method to modify the timestamps of a file on an NTFS system, you are not directly altering the file's original metadata stored in the Master File Table (MFT). Instead, you are updating the Standard_Information Attribute, which is a component stored in the MFT that contains file metadata, including timestamps such as creation, modification, metadata change, and last access time. For more information on the SetFileTime API, you can refer to the Microsoft documentation: https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-setfiletime

The SetFileTime API has the following structure,

```
BOOL SetFileTime(
  [in]           HANDLE         hFile,
  [in, optional] const FILETIME *lpCreationTime,
  [in, optional] const FILETIME *lpLastAccessTime,
  [in, optional] const FILETIME *lpLastWriteTime
);
```

As can be seen in the API, it can be used to **Set/Alter the date and time of the specified file or directory was created, last accessed, or last modified**. The handle is received from CreateFileA or CreateFileW API, but we do not need that as we will see below.

PRACTICAL
Now, let's observe this process in action using Olly Debugger. I will be using an old malware sample from 2018.
Hash – EDA0A631619ABB9C38C6436D511BC67E

To begin, open the malware file in Olly Debugger and set a breakpoint (F2) on the SetFileTime API. You can locate the API using the Ctrl + G shortcut. You can use SetFileTime API from Kernel32 or KERNELBASE, basically Kernel32 one is just wrapper around KERNELBASE, so I am showing KERNELBASE only here.

Once the breakpoint is set, run the debugger to initiate the analysis.

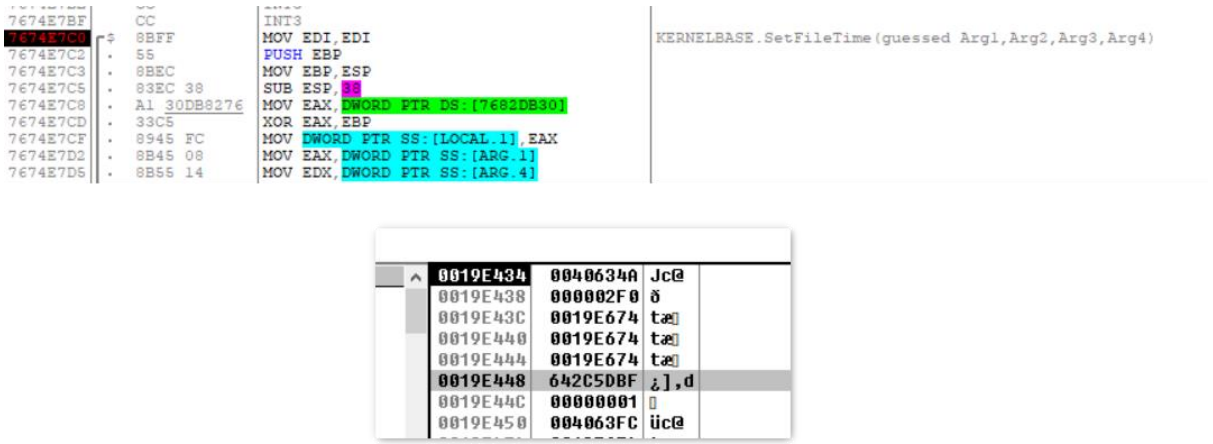


Figure 2: Displaying Breakpoint hit on SetFileTime and Stack at that time.

In Figure 2, we can observe the following information at the time the breakpoint was hit:

hFile = 2F0
**lpCreationTime = 0019E674*
**lpLastAccessTime = 0019E674*
**lpLastWriteTime = 0019E674*

Now that we have the handle value, let's open Process Hacker and locate the file we opened in Olly Debugger. Switch to the "Handle" tab in Process Hacker to identify the file whose timestamp is being altered.

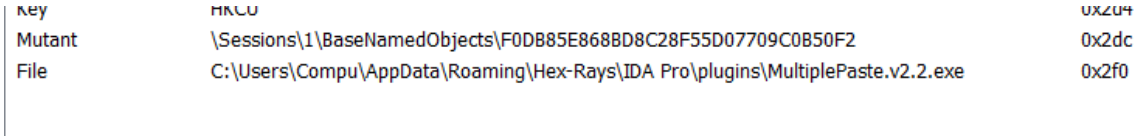


Figure 3: Handle 2F0 image from Process hacker

In Figure 3, we can see that the file being modified is "MultiplePaste.v2.2.exe."

Next, let's check the current timestamp of the file.

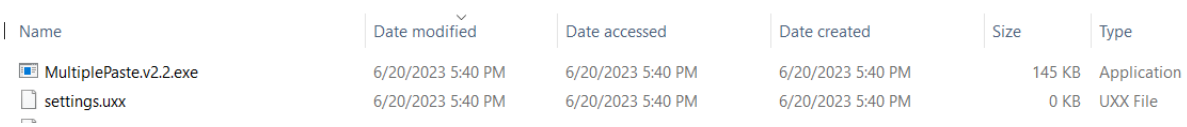


Figure 4: Displays time before it was updated.

As depicted in Figure 4, the current timestamp of the file is 6/20/2023 5:40 PM.

Now, let's return to the Olly Debugger window. We can observe that we have a pointer (indicated by *) to the FILETIME structure (as indicated by the data type) for lpCreationTime etc., pointing to the address 0019E674.

To further investigate, right-click on the stack for this address and follow it in the dump.

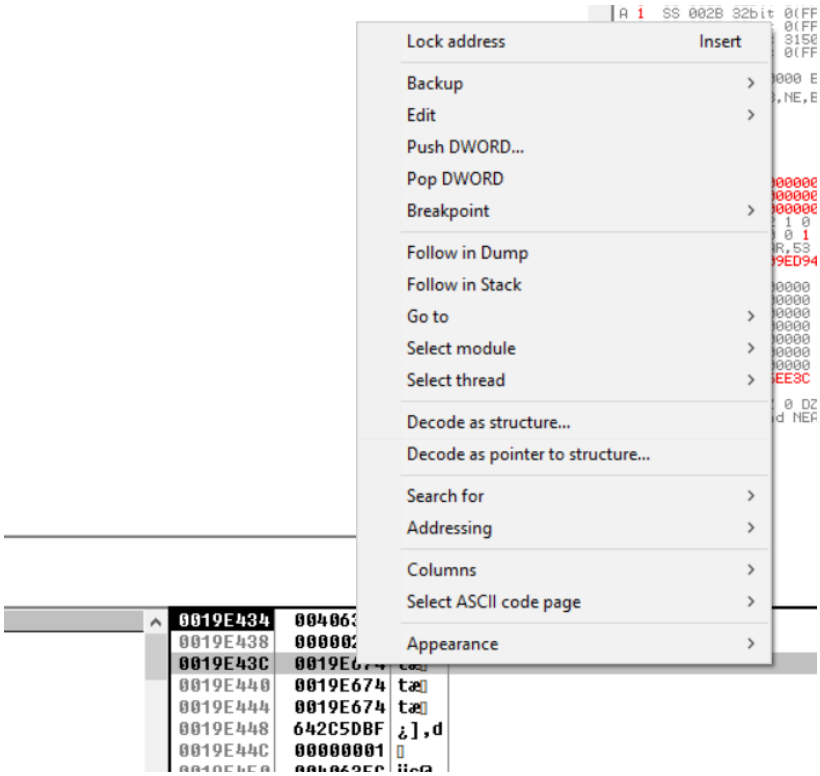


Figure 5: Following the pointer.

Now, Olly Debugger can assist us in decoding the FILETIME structure. Right-click on the structure and select the option to decode it.

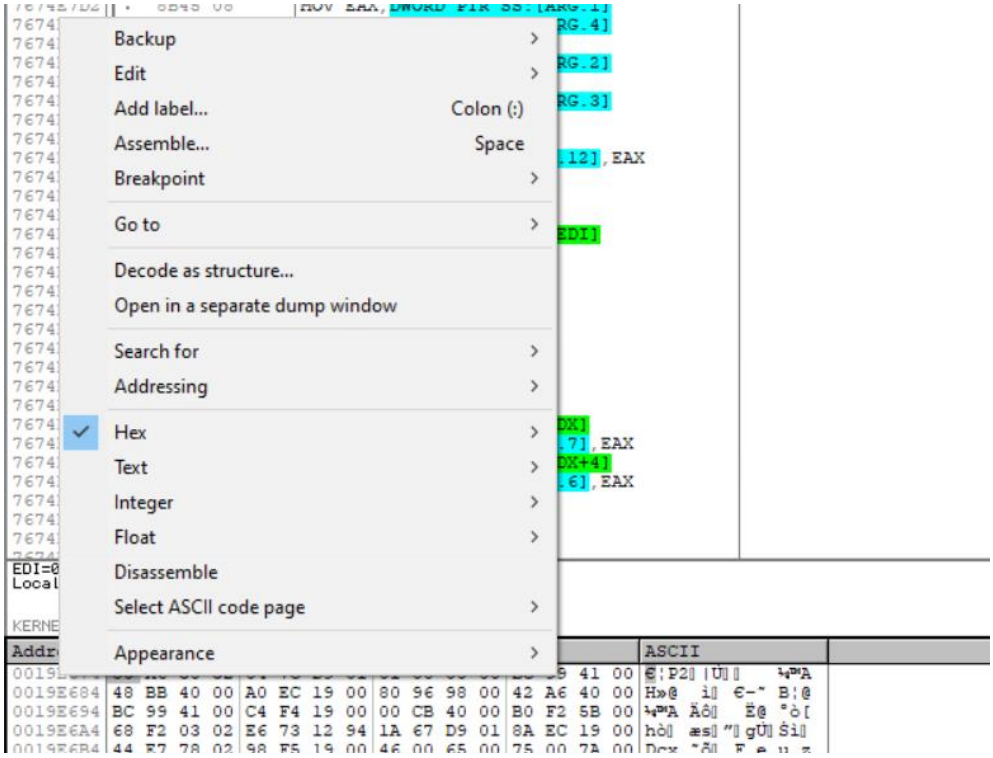


Figure 6: Accessing structure using Olly Debugger.

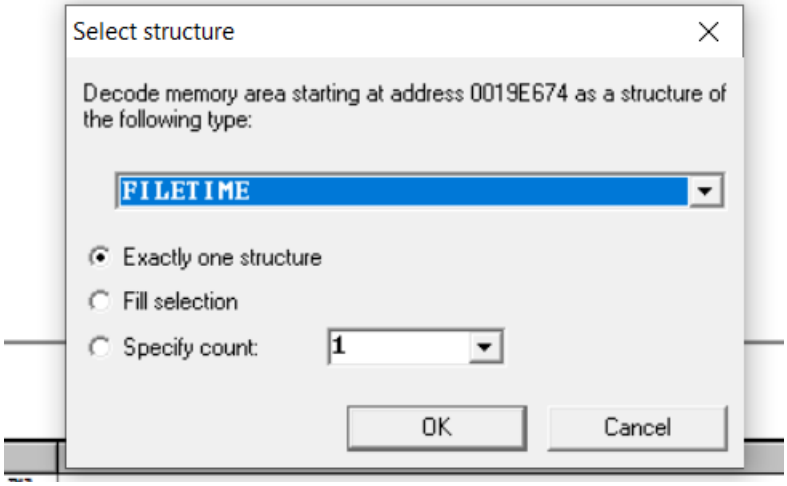


Figure 7: Selecting FILETIME structure.

For a reference on the FILETIME structure, you can visit the Microsoft documentation: https://learn.microsoft.com/en-us/windows/win32/api/minwinbase/ns-minwinbase-filetime

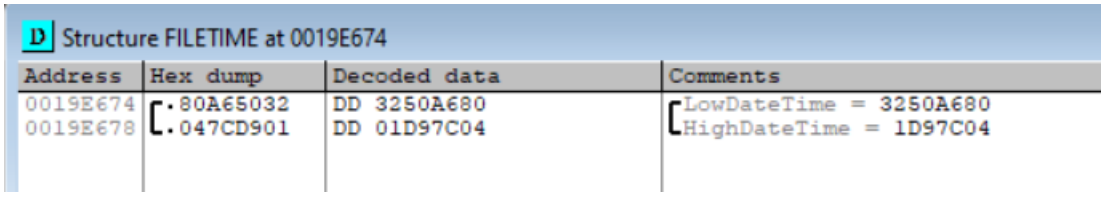


Figure 8: Displaying values in FILETIME Structure

In Figure 8, we can see the values within the FILETIME structure:
LowDateTime – 3250A680
HighDateTime – 1D97C04

Take note of these values. Now, let's continue stepping through the debugger until we reach the NtSetInformationFile API. You will notice that the time has not yet changed. As I said earlier, the SetFileTime function acts as a higher-level function designed to modify file timestamps in a user-friendly manner. However, it relies on the lower-level NtSetInformationFile function to carry out its task.

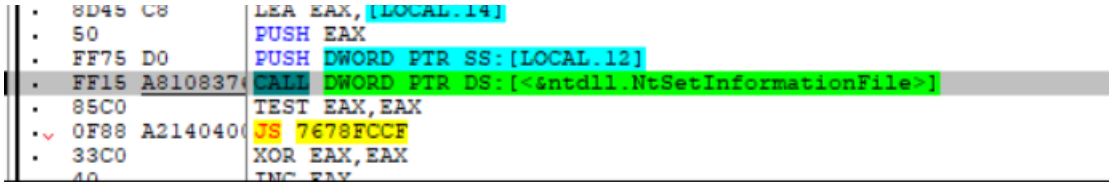


Figure 9: Displaying EIP reached at NtSetInformationFile.

In Figure 9, we have successfully reached the NtSetInformationFile function. The stack can be observed in the image below.

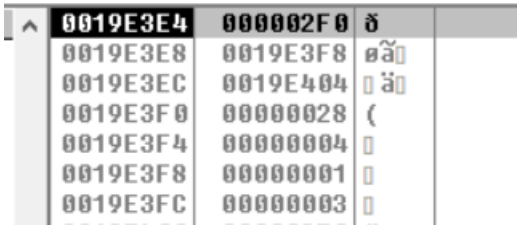


Figure 10: Displaying stack when EIP is at NTSetInformationFile.

Now, let's explore this API in more detail. For reference, you can visit the Microsoft documentation: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/nf-ntifs-ntsetinformationfile

The NtSetInformationFile API has the following structure:

```
_kernel_entry NTSYSCALLAPI NTSTATUS NtSetInformationFile(
  [in]  HANDLE                 FileHandle,
  [out] PIO_STATUS_BLOCK       IoStatusBlock,
  [in]  PVOID                  FileInformation,
  [in]  ULONG                  Length,
  [in]  FILE_INFORMATION_CLASS FileInformationClass
);
```

Based on the information above and the values on the stack, we have the following:

FileHandle – 2F0 (as we saw above)
IoStatusBlock – 0019E3F8

FileInformation – 0019E404
length - 28
FileInformationClass – 4

Now we are interested in File Handle ("**MultiplePaste.v2.2.exe"**), FileInformation ("**0019E404**") and FileInformationClass (**"4"**),

If we refer to the documentation, *FileInformation is a pointer to a buffer containing the information to be set for the file. The specific structure within this buffer is determined by the* ***FileInformationClass*** *parameter.*

Alright so we have to look at ***FileInformationClass*** which is **"4",** which indicates that we need to change the information provided in a FILE_BASIC_INFORMATION structure. You can find more details about the **FILE_BASIC_INFORMATION** structure in the Microsoft documentation: https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/ns-wdm-_file_basic_information

```
typedef struct _FILE_BASIC_INFORMATION {
  LARGE_INTEGER CreationTime;
  LARGE_INTEGER LastAccessTime;
  LARGE_INTEGER LastWriteTime;
  LARGE_INTEGER ChangeTime;
  ULONG         FileAttributes;
} FILE_BASIC_INFORMATION, *PFILE_BASIC_INFORMATION;
```

Figure 11: Displaying _FILE_BASIC_INFORMATION Structure

We can see the data type as **Large_Integer i.e. 64 bit value.** Now remember I said we will decode the structure by ourselves, now this information will be helpful.
Let's follow the FileInformation ("**0019E404**") in dump, and you will notice a repeating pattern, i.e. 80 A6 50 32 04 7C D9 01 -> repeating itself 3 times which basically represents CreationTime, LastAccessTime, and LastWriteTime values.

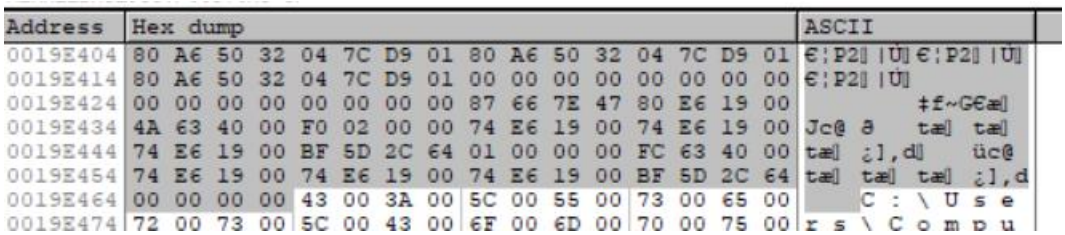


Figure 12: Displaying the FileInformation dump in memory.

However, this pattern appears different from what we observed in **SetFileTime**. In reality, it is not different; it's just in a **different byte order**. If you convert it to **little endian** format, you will see that it is actually 01 D9 7C 04 32 50 A6 80.

Nice, so now we have time value. Yayyy…

Conversion to human readable format

Now that we have the time values, let's convert them to a human-readable format. We will explore how to do this shortly. But before that, let's proceed by pressing F8 to call the function and verify the value in Explorer.

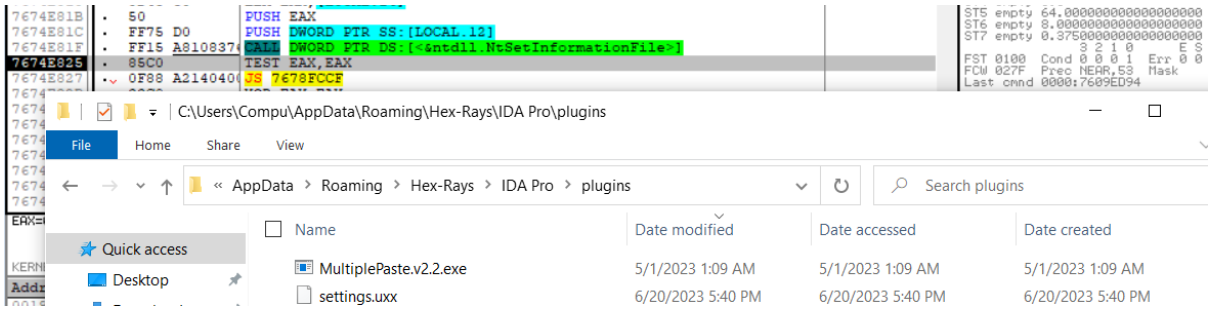


Figure 13: Shows New timestamp of the file.

We can see the new time is changed **from 6/20/2023 5:40 PM to 5/1/2023 1:09 AM**.
AWESOMEEEEEE

Now, let's learn how to convert the received value (01 D9 7C 04 32 50 A6 80) into a human-readable format.

For reference, you can visit the Microsoft documentation: https://learn.microsoft.com/en-us/windows/win32/api/minwinbase/ns-minwinbase-filetime

According to the documentation, this hexadecimal value represents a 64-bit value that denotes the number of 100-nanosecond intervals since January 1, 1601 (UTC).

So let’s first convert it into decimal format,

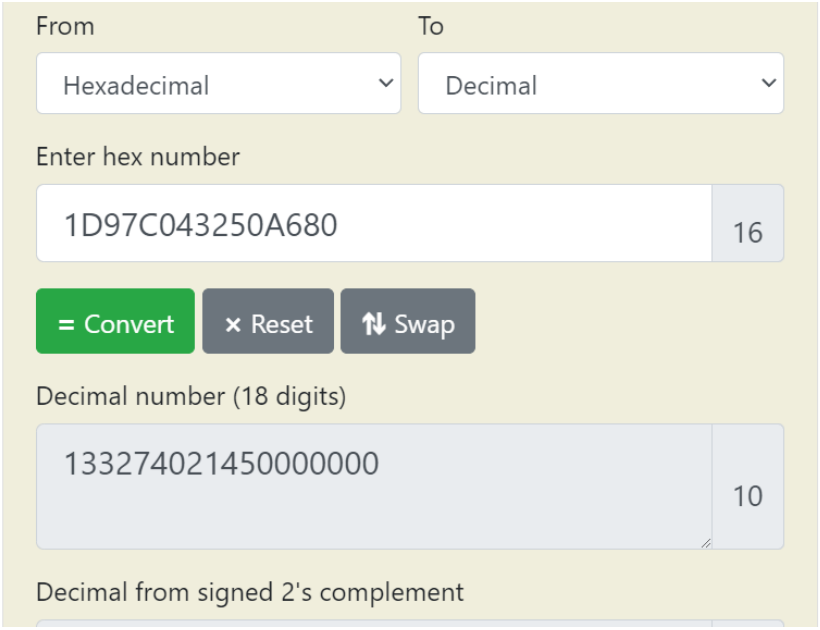


Figure 14: Displaying decimal representation for above hexadecimal value.

To convert it to a decimal format, divide the number by 10^7 since it represents 100-nanosecond intervals.

So now we have **13327402145** seconds since *January 1, 1601 (UTC).*

I have created a small python script to add this second to *January 1, 1601 (UTC)*

```
PS C:\> python
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import datetime
>>> time = 13327402145
>>> human_readable_time = datetime.datetime(1601, 1, 1) + datetime.timedelta(seconds=time)
>>> print(human_readable_time)
2023-05-01 08:09:05
>>>
```

Figure 15: Displaying time output in human readable format.

As we can see this is UTC time and my machine is set at UTC–8 time converting it will display the same time as shown in your Explorer. This is how we can determine the time altered by the malware for the specific file.

I hope you found this process enjoyable! Thank you for reading!

==========