Assignment 3 Report

Dimitry Rakhlei

Contents

Summary

The goal in this assignment was to reverse engineer executable programs and extract key information and disable specified portions of their code. The software used was x96dbg (x64 & x32dbg) which is an open source version of OllyDbg. They both look and perform very similarly.

Process

The debugging process was performed inside of a VMware virtual machine. The machine was configured, and a snapshot was created in case of a malware infection.

For each executable I looked at hard coded strings to get a sense of where in the code certain printouts occurred. This was done by right clicking on the disassembled code and selecting Search For -> All Modules -> String References.

Before doing any modifications to the code it is nice to also see what kinds of functions were loaded and used by the process. Here we use the feature found by right clicking on the disassembled code and pressing Search For -> All Modules -> Intermodular Calls. The output tables can be found in the root directory submitted with this report. The log files are called *"function_calls_crackme1.txt"* and *"function_calls_crackme2.txt"* for the function calls and *"string_references_crackme1.txt"* and *"string_references_crackme2.txt"* for string references.

Analysis

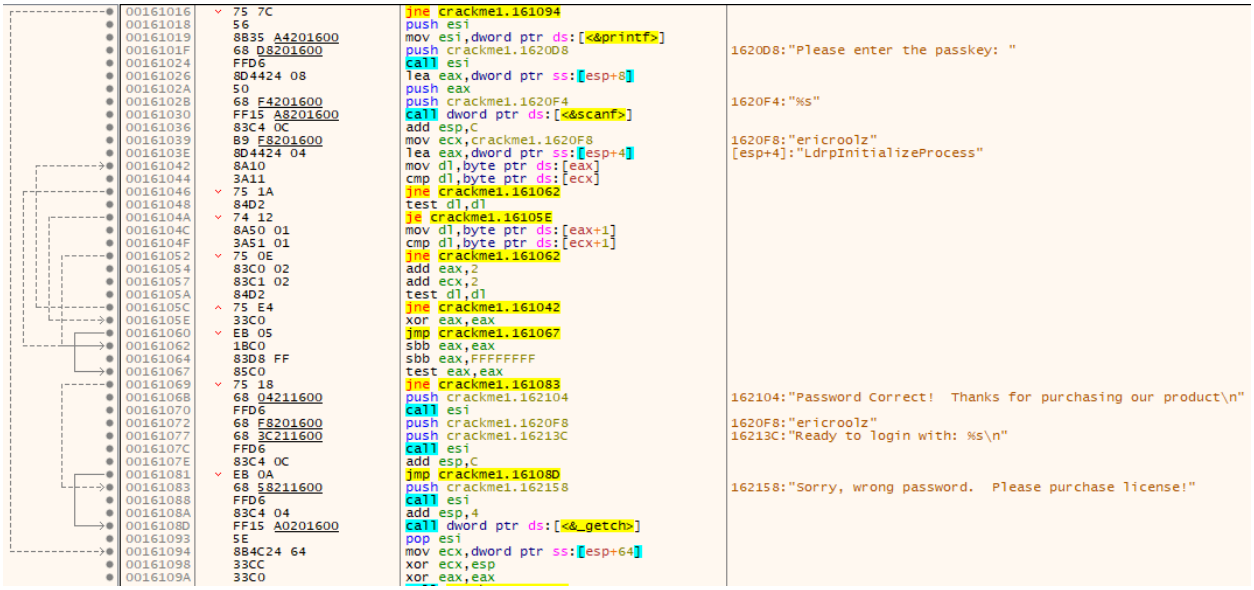
crackme1.exe

After looking at the function calls of *crackme1* it appears that the application uses scanf and printf functions and does not load any graphical functions. It may very well be a console application which asks for input and has outputs displayed as a response. When we looked at the strings inside of *crackme1* we found that it contains strings such as:

```
0016101F push crackme1.1620D8                    "Please enter the passkey: "
0016102B push crackme1.1620F4                    "%s"
00161039 mov ecx,crackme1.1620F8                 "ericroolz"
0016106B push crackme1.162104                    "Password Correct!  Thanks for purchasing our product\n"
00161072 push crackme1.1620F8                    "ericroolz"
00161077 push crackme1.16213C                    "Ready to login with: %s\n"
00161083 push crackme1.162158                    "Sorry, wrong password.  Please purchase license!"
00161796 push crackme1.1610A5                    ";\r"
```

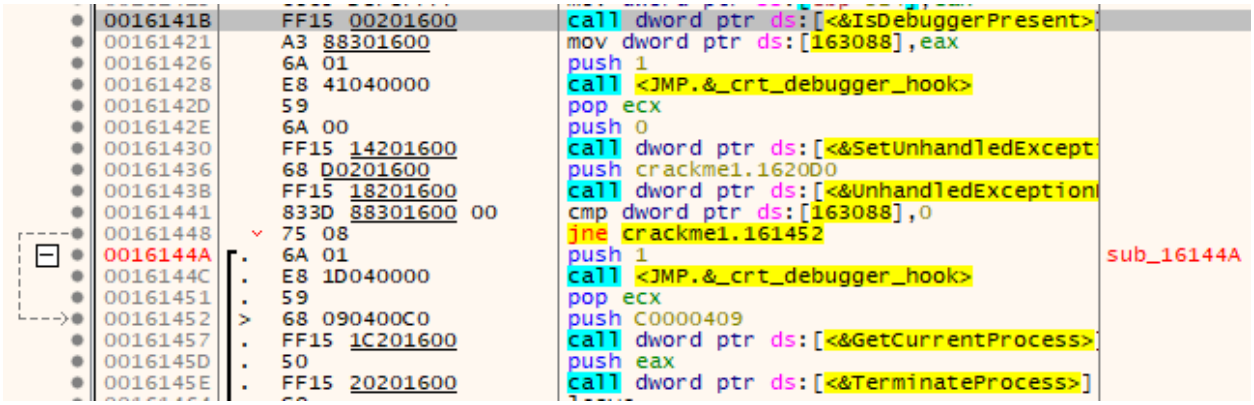
We will follow the string which says, *"Password Correct!"* and see what comparison instruction causes this code to execute.

Here is a portion of the code which will perform the authentication:



At address 00161030 scanf is called with the parameter “%s” meaning that a string is being read from stdin into eax.

Some code starting at 0016141B does not allow the debugger to step through. We could disable it but it is not required for this assignment.



Unsuccessful Authentication:

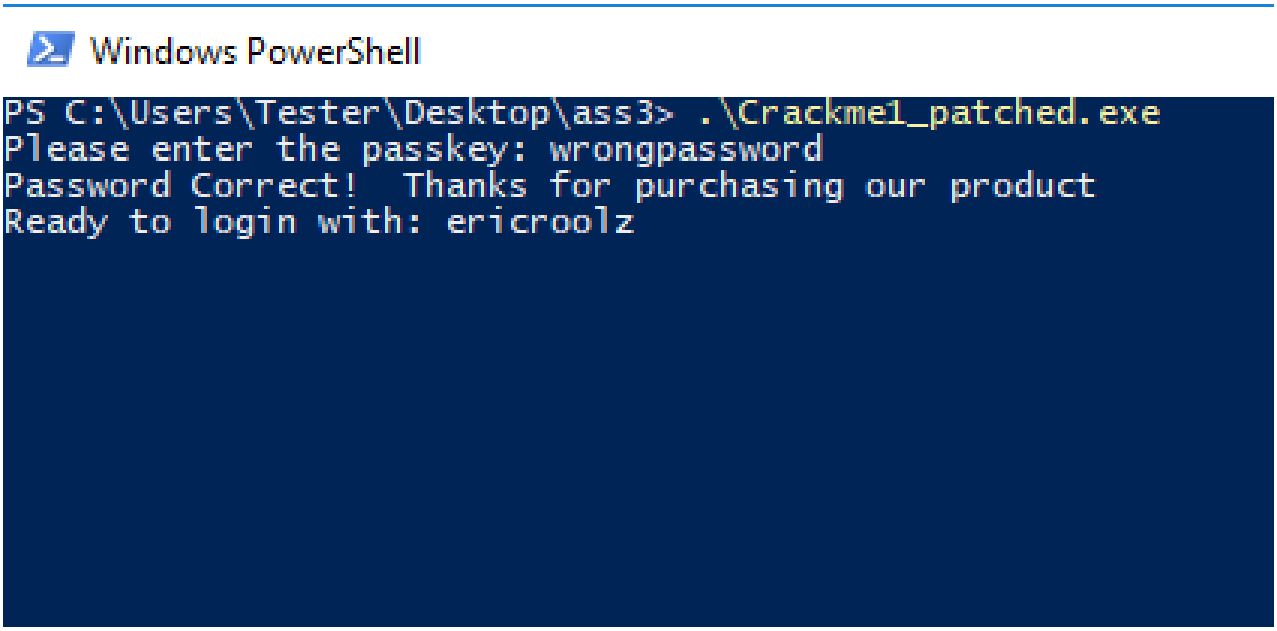
At 00161044 the input string is compared to a string literal “ericroolz”. If the two strings are not equal the code jumps to 00161062 where the code proceeds to jump to 00161083. At this point the failed code pushes the string “Sorry, wrong password.” Onto the stack and calls the address of the printf command to print it.

Successful Authentication:

At 00161046 the comparison determines that the two strings are equal. It tests if the dl register is equal to itself. If they are equal then it jumps to 0016105E where the eax register is cleared and we jump to 00161067. Since eax was cleared it will be equal to itself and therefore it will not jump. At 0016106B the success string is pushed onto the stack and later printf is called to display it along with the password used.

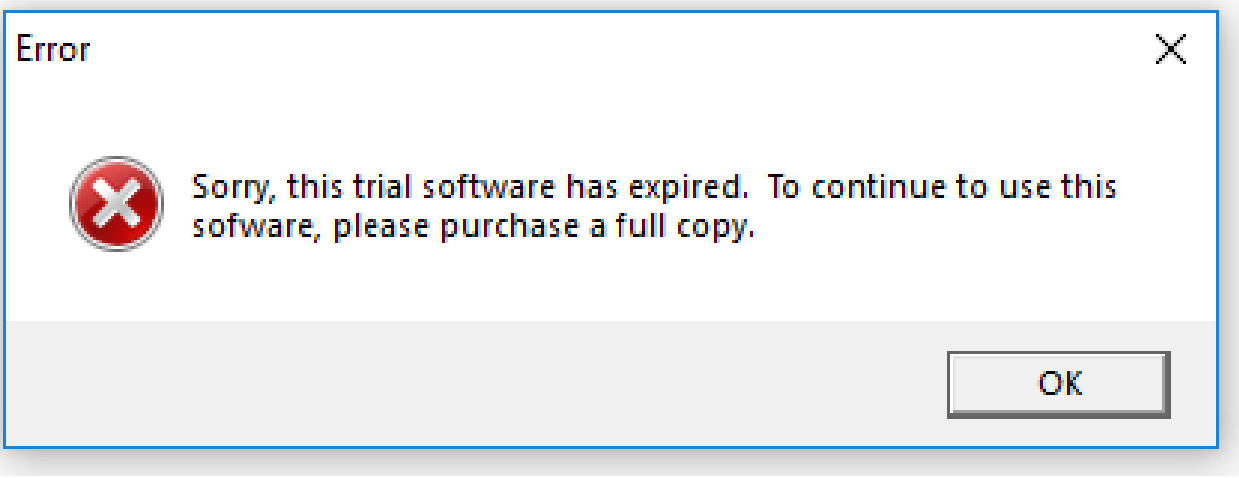
Patch:

To patch the code, we simply have to replace the JNE instruction at address 00161046 with the instruction “JMP 0016106B” which is the address at which the successful login starts. After applying the patch, we now have a new executable

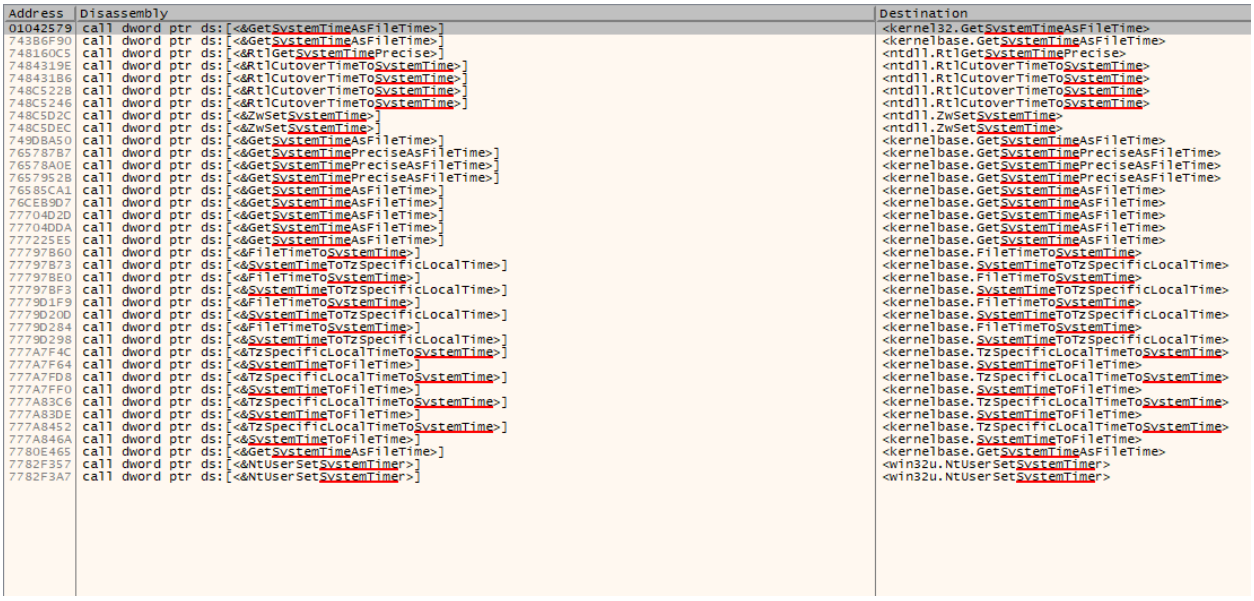


crackme2.exe

After looking at the function calls of *crackme2* it appears that the application uses a graphical interface as evidenced by the existence of functions such as CreateWindowExW and RegisterClassExW which are core win32 GUI functions. When we run the program, we get this message:



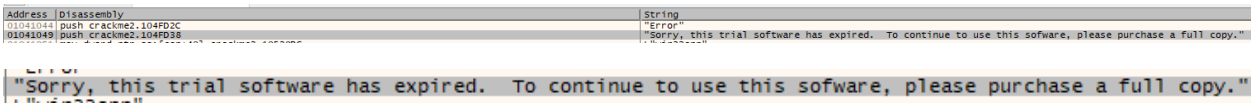
It says that the program has expired, and we can see in the function tables that the program does in fact use many functions which check the current system time:



Address	Disassembly	Destination
01042579	call dword ptr ds:[<&GetSystemTimeAsFileTime>]	<kernel32.GetSystemTimeAsFileTime>
743B6F90	call dword ptr ds:[<&GetSystemTimeAsFileTime>]	<kernelbase.GetSystemTimeAsFileTime>
748160C5	call dword ptr ds:[<&RtlGetSystemTimePrecise>]	<ntdll.RtlGetSystemTimePrecise>
7484319E	call dword ptr ds:[<&RtlCutoverTimeToSystemTime>]	<ntdll.RtlCutoverTimeToSystemTime>
748431B6	call dword ptr ds:[<&RtlCutoverTimeToSystemTime>]	<ntdll.RtlCutoverTimeToSystemTime>
748C5228	call dword ptr ds:[<&RtlCutoverTimeToSystemTime>]	<ntdll.RtlCutoverTimeToSystemTime>
748C5246	call dword ptr ds:[<&RtlCutoverTimeToSystemTime>]	<ntdll.RtlCutoverTimeToSystemTime>
748C5D2C	call dword ptr ds:[<&ZwSetSystemTime>]	<ntdll.ZwSetSystemTime>
748C5DEC	call dword ptr ds:[<&ZwSetSystemTime>]	<ntdll.ZwSetSystemTime>
749DBA50	call dword ptr ds:[<&GetSystemTimeAsFileTime>]	<kernelbase.GetSystemTimeAsFileTime>
765787B7	call dword ptr ds:[<&GetSystemTimePreciseAsFileTime>]	<kernelbase.GetSystemTimePreciseAsFileTime>
76578A0E	call dword ptr ds:[<&GetSystemTimePreciseAsFileTime>]	<kernelbase.GetSystemTimePreciseAsFileTime>
7657952B	call dword ptr ds:[<&GetSystemTimePreciseAsFileTime>]	<kernelbase.GetSystemTimePreciseAsFileTime>
7658SCA1	call dword ptr ds:[<&GetSystemTimeAsFileTime>]	<kernelbase.GetSystemTimeAsFileTime>
76CEB9D7	call dword ptr ds:[<&GetSystemTimeAsFileTime>]	<kernelbase.GetSystemTimeAsFileTime>
77704D2D	call dword ptr ds:[<&GetSystemTimeAsFileTime>]	<kernelbase.GetSystemTimeAsFileTime>
77704DDA	call dword ptr ds:[<&GetSystemTimeAsFileTime>]	<kernelbase.GetSystemTimeAsFileTime>
777225E5	call dword ptr ds:[<&GetSystemTimeAsFileTime>]	<kernelbase.GetSystemTimeAsFileTime>
77797B60	call dword ptr ds:[<&FileTimeToSystemTime>]	<kernelbase.FileTimeToSystemTime>
77797B73	call dword ptr ds:[<&SystemTimeToTzSpecificLocalTime>]	<kernelbase.SystemTimeToTzSpecificLocalTime>
77797BE0	call dword ptr ds:[<&FileTimeToSystemTime>]	<kernelbase.FileTimeToSystemTime>
77797BF3	call dword ptr ds:[<&SystemTimeToTzSpecificLocalTime>]	<kernelbase.SystemTimeToTzSpecificLocalTime>
7779D1F9	call dword ptr ds:[<&FileTimeToSystemTime>]	<kernelbase.FileTimeToSystemTime>
7779D20D	call dword ptr ds:[<&SystemTimeToTzSpecificLocalTime>]	<kernelbase.SystemTimeToTzSpecificLocalTime>
7779D284	call dword ptr ds:[<&FileTimeToSystemTime>]	<kernelbase.FileTimeToSystemTime>
7779D298	call dword ptr ds:[<&SystemTimeToTzSpecificLocalTime>]	<kernelbase.SystemTimeToTzSpecificLocalTime>
777A7F4C	call dword ptr ds:[<&TzSpecificLocalTimeToSystemTime>]	<kernelbase.TzSpecificLocalTimeToSystemTime>
777A7F64	call dword ptr ds:[<&SystemTimeToFileTime>]	<kernelbase.SystemTimeToFileTime>
777A7FD8	call dword ptr ds:[<&TzSpecificLocalTimeToSystemTime>]	<kernelbase.TzSpecificLocalTimeToSystemTime>
777A7FF0	call dword ptr ds:[<&SystemTimeToFileTime>]	<kernelbase.SystemTimeToFileTime>
777A83C6	call dword ptr ds:[<&TzSpecificLocalTimeToSystemTime>]	<kernelbase.TzSpecificLocalTimeToSystemTime>
777A83DE	call dword ptr ds:[<&SystemTimeToFileTime>]	<kernelbase.SystemTimeToFileTime>
777A8452	call dword ptr ds:[<&TzSpecificLocalTimeToSystemTime>]	<kernelbase.TzSpecificLocalTimeToSystemTime>
777A846A	call dword ptr ds:[<&SystemTimeToFileTime>]	<kernelbase.SystemTimeToFileTime>
7780E465	call dword ptr ds:[<&GetSystemTimeAsFileTime>]	<kernelbase.GetSystemTimeAsFileTime>
7782F357	call dword ptr ds:[<&NtUserSetSystemTimer>]	<win32u.NtUserSetSystemTimer>
7782F3A7	call dword ptr ds:[<&NtUserSetSystemTimer>]	<win32u.NtUserSetSystemTimer>

With this information we will now analyze where in the code to see where the string we see in the message box is loaded.

Here we see the strings from the process:



Address	Disassembly	String
01041044	push crackme2.104FD2C	"Error"
01041049	push crackme2.104FD38	"Sorry, this trial software has expired. To continue to use this sofware, please purchase a full copy."

"Sorry, this trial software has expired. To continue to use this sofware, please purchase a full copy."

If we follow the string we can see that it is only used right at the top of the executable:

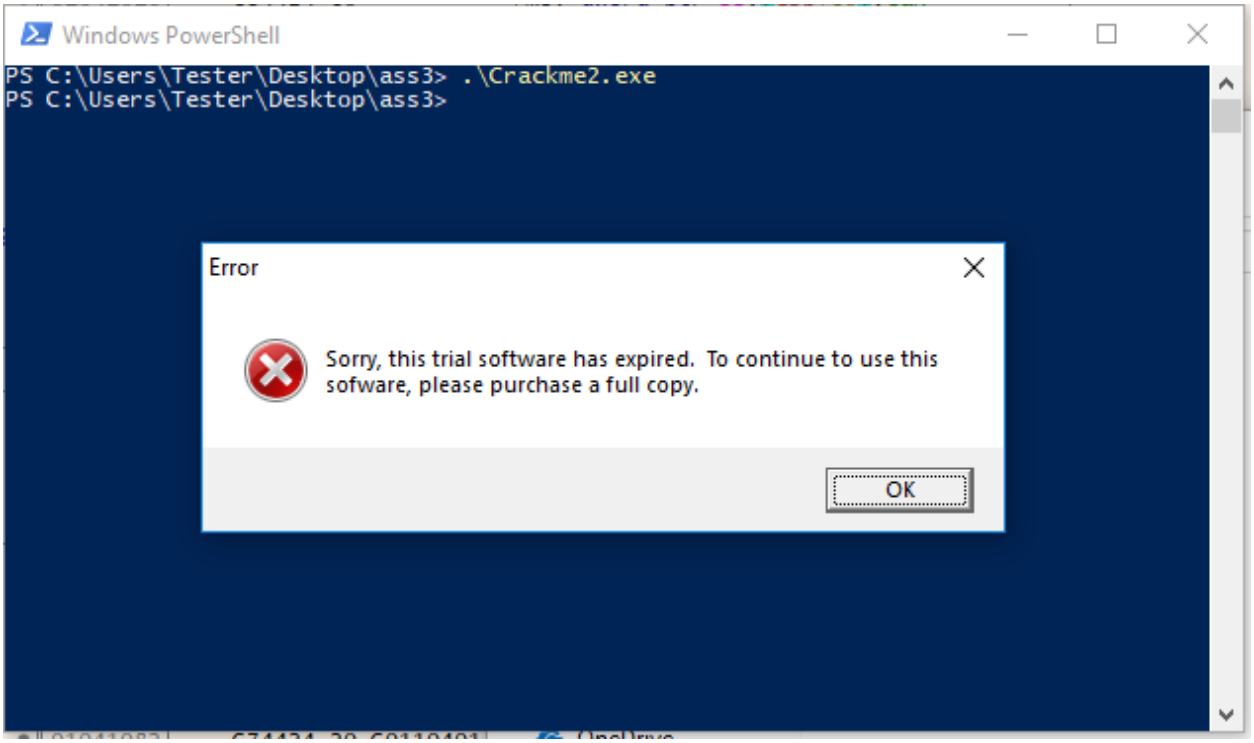
●	01041000	55	push ebp	
●	01041001	8BEC	mov ebp,esp	
●	01041003	83E4 F8	and esp,FFFFFFF8	
●	01041006	83EC 64	sub esp,64	
●	01041009	A1 00100501	mov eax,dword ptr ds:[1051000]	
●	0104100E	33C4	xor eax,esp	
●	01041010	894424 60	mov dword ptr ss:[esp+60],eax	
●	01041014	53	push ebx	
●	01041015	56	push esi	
●	01041016	57	push edi	edi:"LdrpInitializeProcess"
●	01041017	8B7D 08	mov edi,dword ptr ss:[ebp+8]	
●	0104101A	8D4424 5C	lea eax,dword ptr ss:[esp+5C]	
●	0104101E	33DB	xor ebx,ebx	
●	01041020	50	push eax	
●	01041021	8D73 01	lea esi,dword ptr ds:[ebx+1]	
●	01041024	FF15 24C00401	call dword ptr ds:[<&GetLocalTime>]	
●	0104102A	66:8B4424 5C	mov ax,word ptr ss:[esp+5C]	
●	0104102F	B9 DE070000	mov ecx,7DE	
●	01041034	66:3BC1	cmp ax,cx	
●	01041037	76 04	jbe crackme2.104103D	
●	01041039	8BDE	mov ebx,esi	
●	0104103B	33F6	xor esi,esi	
●	0104103D	66:85C0	test ax,ax	
●	01041040	75 1C	jne crackme2.104105E	
●	01041042	6A 10	push 10	
●	01041044	68 2CFD0401	push crackme2.104FD2C	104FD2C:"Error"
●	01041049	68 38FD0401	push crackme2.104FD38	104FD38:"Sorry, this trial software has expired. To continue to use this sofware,
●	0104104E	6A 00	push 0	
●	01041050	FF15 20C10401	call dword ptr ds:[<&MessageBoxA>]	
●	01041056	6A 01	push 1	
●	01041058	FF15 30C00401	call dword ptr ds:[<&ExitProcess>]	

Here we can see that the process calls the win32 function GetLocalTime at address 01041024.

This program appears to not be using a password but instead checking and comparing against a date hardcoded at address 0104102F. This is evidenced by 0x7DE being equal to 2014 in decimal.

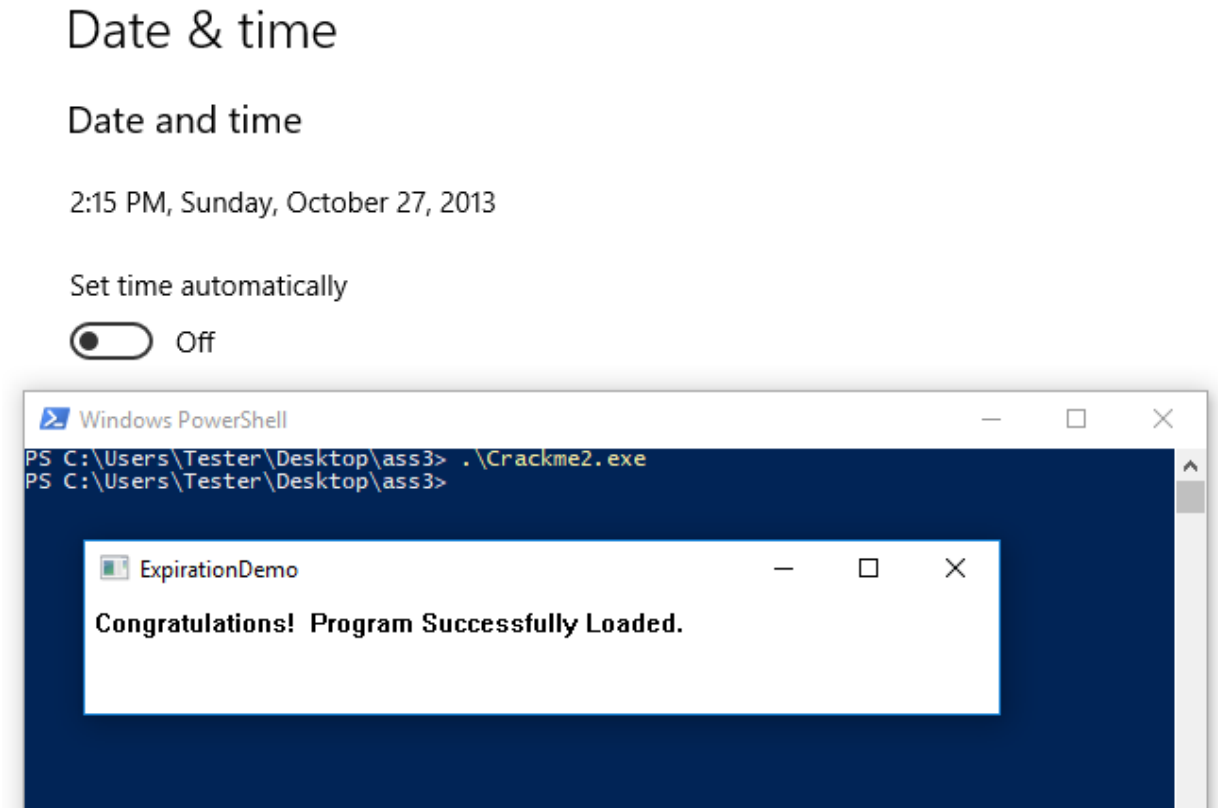
Unsuccessful Authentication:

We simply execute the code. It checks the date and because our current date is > 2014 we will get the error message.



Successful Authentication:

Since with the patch we will be modifying the code and the successful authentication requires a less drastic way of getting in we will try to change the current date on our PC.

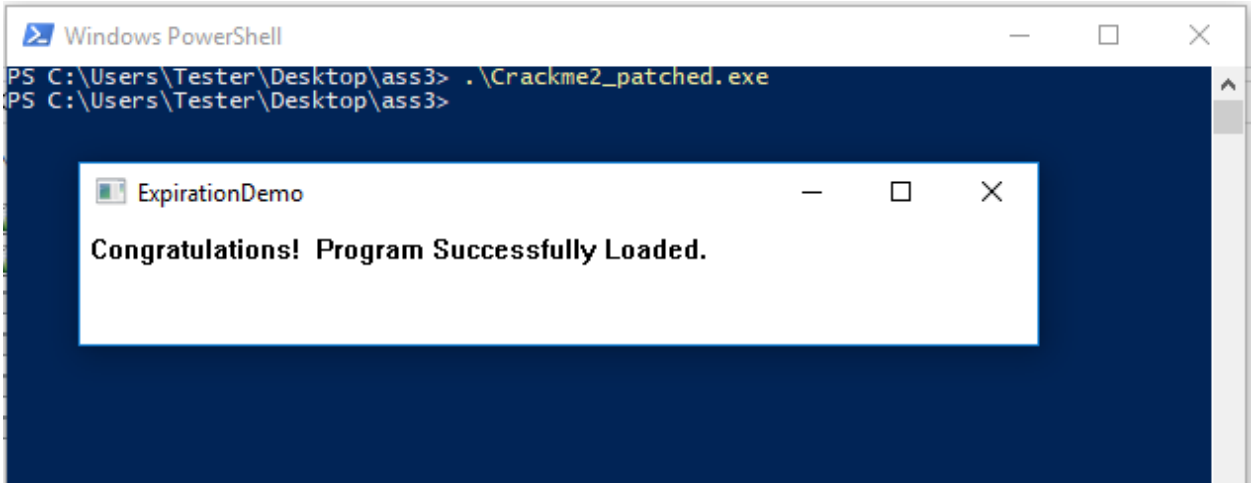


And this works! The code is simple so fooling it by changing the date is trivial.

Patch:

Since the instruction at 01041037 is checking if AX is below or equal to 2014 we can change the JBE to "JMP 01041066" which will skip all of the code which checks for tampering and or prints error messages.

The output upon running said patched executable is:



Conclusion

I learned a lot about some of the simpler ways in which developers may try to stop a developer from debugging software. The first program was made to shut down if the presence of a debugger was detected. That could be disabled via a patch, but it didn't interfere with my analysis, so I kept it. The second piece of code checked against a date which was interesting, but it also used the device's date which can be modified and should not be trusted.

This was a valuable experience in reverse engineering software with a tool like OllyDbg.