Lab: ProcMon

Environment

- Windows Virtual Machine
- IDA
- ProcMon

Links for the required tools are available in the repository under the 'tools' directory.

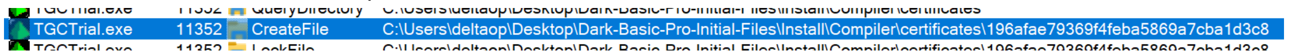
Target

- Dark Basic Pro

Provided in the 'target' directory.

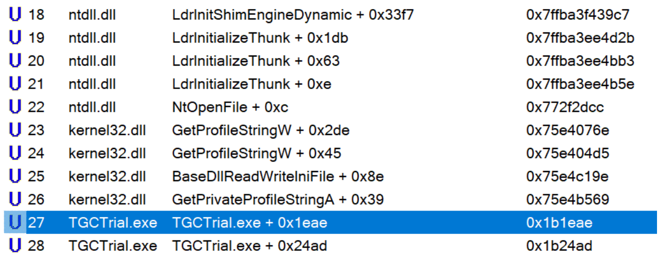
Outline

1. Dark-Basic-Pro was a popular tool for creating Windows video games. It contains several different licensing schemes. We previously explored the CD authenticity check and the Serial check in the software. In this lab, we will determine how the software checks for installed licenses.
2. First, examine the target folder with TGCTrial.exe. What else is in this folder? What licenses are currently installed for the application? What is stored in the license file?
3. Launch Process Monitor (ProcMon). Using the buttons at the top of ProcMon ( ), ensure that it is capturing process information.
4. Run TGTrial.exe. Go to ProcMon, and use the buttons at the top to stop capturing process information.
5. ProcMon will likely have thousands of system and library calls captured at this point. Like everything in reverse-engineering, we need some way to find the pieces that actually matter. Click the Filter button in ProcMon.
6. We're interested in how the target application is processing the license files, so consider what kinds of things we might filter for in ProcMon. Filter by Process Name ("is" "TGCTrial.exe") and Path ("contains" "certificates").
7. Scroll to find the first time the process accesses the installed certificate.



```
TGCTrial.exe   11352   QueryDirectory   C:\Users\deltaop\Desktop\Dark-Basic-Pro-Initial-Files\Install\Compiler\certificates
TGCTrial.exe   11352   CreateFile       C:\Users\deltaop\Desktop\Dark-Basic-Pro-Initial-Files\Install\Compiler\certificates\196afae79369f4feba5869a7cba1d3c8
TGCTrial.exe   11352   LockFile         C:\Users\deltaop\Desktop\Dark-Basic-Pro-Initial-Files\Install\Compiler\certificates\196afae79369f4feba5869a7cba1d3c8
```

8. Double click this entry to pull up additional information from this call. Navigate to the "stack" tab to find the call stack for this call. The top (most recent) several entries will all be inside various Windows libraries, but we are only interested in when the *target* (TGCTrial) made the initial call to access the file. Scroll down to the first line with TGCTrial on it. This tells us exactly where in the code the target tried to access the license file.

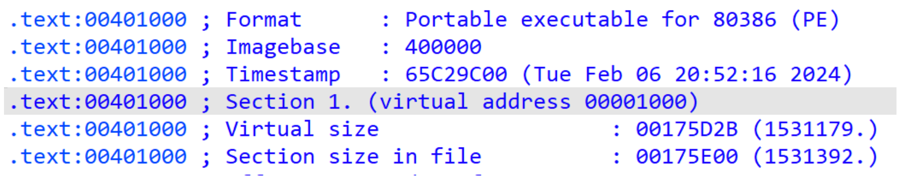


```
U 18   ntdll.dll      LdrInitShimEngineDynamic + 0x33f7   0x7ffba3f439c7
U 19   ntdll.dll      LdrInitializeThunk + 0x1db          0x7ffba3ee4d2b
U 20   ntdll.dll      LdrInitializeThunk + 0x63           0x7ffba3ee4bb3
U 21   ntdll.dll      LdrInitializeThunk + 0xe            0x7ffba3ee4b5e
U 22   ntdll.dll      NtOpenFile + 0xc                    0x772f2dcc
U 23   kernel32.dll   GetProfileStringW + 0x2de           0x75e4076e
U 24   kernel32.dll   GetProfileStringW + 0x45            0x75e404d5
U 25   kernel32.dll   BaseDllReadWriteIniFile + 0x8e      0x75e4c19e
U 26   kernel32.dll   GetPrivateProfileStringA + 0x39     0x75e4b569
U 27   TGCTrial.exe   TGCTrial.exe + 0x1eae               0x1b1eae
U 28   TGCTrial.exe   TGCTrial.exe + 0x24ad               0x1b24ad
```

9. Make note of the address of the call (in the above example, offset 0x1eae).

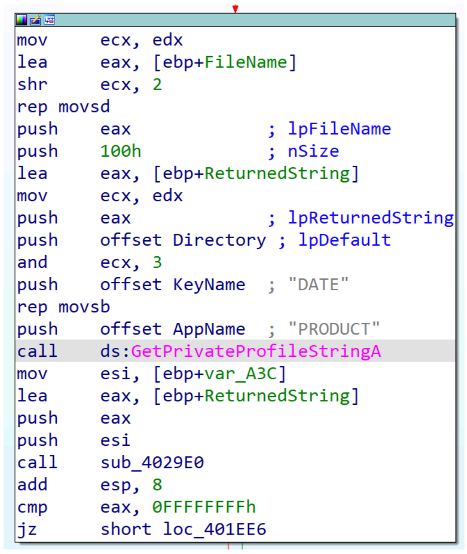
ISBN: 139419988

10. Open IDA, and load TGCTrial into IDA.
11. We want to find the code accessing the license file. We have an address from ProcMon, but we need to translate this a bit in order to translate it to IDA's view. Scroll to the top of the main IDA view, and make note of the .text section virtual address (0x1000 in the below example) and the first address of the .text section shown on the left (0x401000 in the below example, but may be different after you've launched the debugger).



```
.text:00401000 ; Format      : Portable executable for 80386 (PE)
.text:00401000 ; Imagebase   : 400000
.text:00401000 ; Timestamp   : 65C29C00 (Tue Feb 06 20:52:16 2024)
.text:00401000 ; Section 1. (virtual address 00001000)
.text:00401000 ; Virtual size                  : 00175D2B (1531179.)
.text:00401000 ; Section size in file          : 00175E00 (1531392.)
```

12. Using the call offset from ProcMon and the .text addresses, navigate to the call location in the program. In the above example, you can press "g" for "goto" in IDA, then enter, without quotes, "0x401000-0x1000+0x1eae" (IDA can perform basic arithmetic in calculating a go-to address).

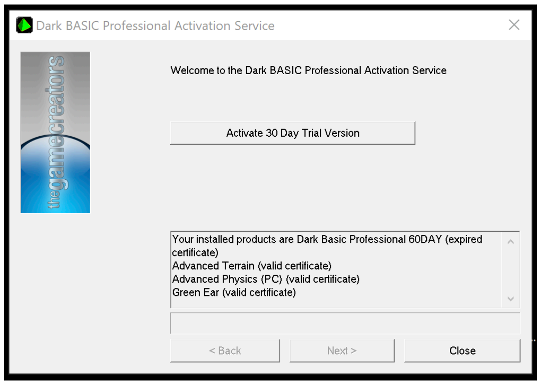


```
mov     ecx, edx
lea     eax, [ebp+FileName]
shr     ecx, 2
rep movsd
push    eax             ; lpFileName
push    100h            ; nSize
lea     eax, [ebp+ReturnedString]
mov     ecx, edx
push    eax             ; lpReturnedString
push    offset Directory ; lpDefault
and     ecx, 3
push    offset KeyName  ; "DATE"
rep movsb
push    offset AppName  ; "PRODUCT"
call    ds:GetPrivateProfileStringA
mov     esi, [ebp+var_A3C]
lea     eax, [ebp+ReturnedString]
push    eax
push    esi
call    sub_4029E0
add     esp, 8
cmp     eax, 0FFFFFFFFh
jz      short loc_401EE6
```

13. You should recognize "DATE" and "PRODUCT" as strings from the license file.
14. Without any code analysis, we've successfully found the point responsible for license checks in the product.

Task 1

Dark Basic Pro supports many different licenses for a wide variety of plugins and addons. The included license gives access to a few features, which are listed when we launch TGCTrial.exe:


Dark BASIC Professional Activation Service
thegamecreators
Activate 30 Day Trial Version
< Back
Next >
Close

Welcome to the Dark BASIC Professional Activation Service

Your installed products are Dark Basic Professional 60DAY (expired certificate)
Advanced Terrain (valid certificate)
Advanced Physics (PC) (valid certificate)
Green Ear (valid certificate)

Starting from the code we identified in the introduction, use IDA and its debugger to find a way to generate a license for an additional product.

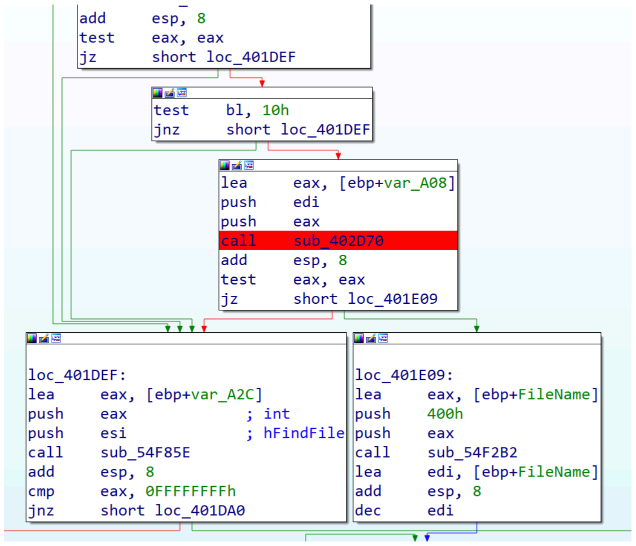
Hints:

- Hints are available at the end of this document. Try the challenge without using hints first, you will encounter and overcome many more of the inherent challenges of the tooling and reversing process this way.

Optional Bonus Tasks

Automatically find all possible license names.

- IDA script is a scripting language built into IDA.
- After you set a breakpoint, you can right click it to edit it.
- One of the fields in the breakpoint window is a breakpoint *condition*.
- The breakpoint condition determines if the debugger will pause on the breakpoint. But the condition is actually just a snippet of IDA script, and can be used for more than just conditions.
- Set a breakpoint on the "strcmp" call that precedes the GetPrivateProfileStringA call. It looks like the following, though your specific offsets may vary. IDA may not recognize the subfunction as a strcmp, giving it instead a generic name like "sub_123456".



```
add     esp, 8
test    eax, eax
jz      short loc_401DEF
```

```
test    bl, 10h
jnz     short loc_401DEF
```

```
lea     eax, [ebp+var_A08]
push    edi
push    eax
call    sub_402D70
add     esp, 8
test    eax, eax
jz      short loc_401E09
```

```
loc_401DEF:
lea     eax, [ebp+var_A2C]
push    eax             ; int
push    esi             ; hFindFile
call    sub_54F85E
add     esp, 8
cmp     eax, 0FFFFFFFFh
jnz     short loc_401DA0
```

```
loc_401E09:
lea     eax, [ebp+FileName]
push    400h
push    eax
call    sub_54F2B2
lea     edi, [ebp+FileName]
add     esp, 8
dec     edi
```

- Set the breakpoint condition to "print("X") && 0" without quotes). This will always evaluate to false, so IDA will not pause on the breakpoint. However, in the process of evaluating the condition, IDA will print "X" each time the breakpoint is hit.
- Explore the IDA scripting functions to find a way to fetch strings from memory. Using the current program registers at the breakpoint, can you get IDA to automatically print the names of the license files as they are checked?

Find a way to *unexpire* your licenses, without modifying the program.

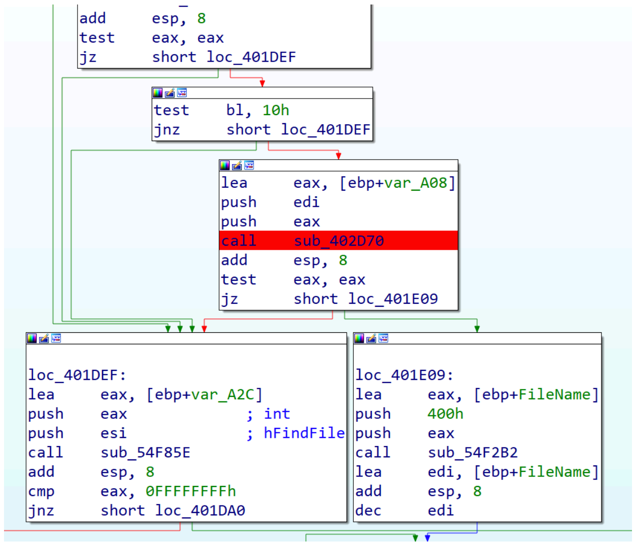
- Some of the licenses found in Task 1 may be expired (if you have no expired license, try setting the clock in your VM forward by a year and re-running the program).
- The target is looking for an obfuscated timestamp in the registry.
- Use ProcMon to find the registry logic. This will lead you to the timestamp obfuscation logic.
- Decipher the timestamp obfuscation method with IDA and reverse engineering.
- Add a valid obfuscated timestamp to your registry and license file to undo the license expiration.

ISBN: 139419988

Task 1 Hints

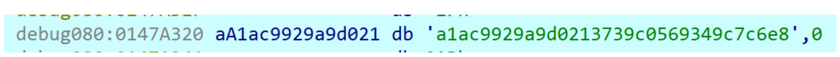
Tip: Use hints sequentially; once you've unblocked yourself, try to continue without using the subsequent hints.

- Don't get lost in the weeds, the problem can be largely solved without a detailed analysis of specific assembly instructions. Focus on the high-level flow of the code around the "GetPrivateProfileStringA" call.
- Use the debugger to try single stepping through the function that calls "GetPrivateProfileStringA". Use the "step-over" functionality in the debugger to avoid entering into subfunctions, so that we can stay focused on only this function and not get lost in the details of libraries and helper functions. As you step through, keep an eye on register values as they're being used, and hover over them to see if they point to anything interesting in memory.
- Tracing backwards from GetPrivateProfileStringA, is there a code path that can skip this block? What condition will skip GetPrivateProfileStringA? What determines this condition (recall that functions return their result in the eax register).
- Locate and place a breakpoint this call prior to the GetPrivateProfileStringA function. Run until this breakpoint is hit (restart your program if the breakpoint is missed).



```
add     esp, 8
test    eax, eax
jz      short loc_401DEF

test    bl, 10h
jnz     short loc_401DEF

lea     eax, [ebp+var_A08]
push    edi
push    eax
call    sub_402D70
add     esp, 8
test    eax, eax
jz      short loc_401E09

loc_401DEF:
lea     eax, [ebp+var_A2C]
push    eax              ; int
push    esi              ; hFindFile
call    sub_54F85E
add     esp, 8
cmp     eax, 0FFFFFFFFh
jnz     short loc_401DA0

loc_401E09:
lea     eax, [ebp+FileName]
push    400h
push    eax
call    sub_54F2B2
lea     edi, [ebp+FileName]
add     esp, 8
dec     edi
```

- Hover over EDI and EAX once the breakpoint is triggered. What do they point to? Do you recognize one of them? (Think back to the lab outline, and the contents of the license file).
- Follow EDI to memory, and right click to create a string at this location.



```
debug080:0147A320 aA1ac9929a9d021 db 'a1ac9929a9d0213739c0569349c7c6e8',0
```

- This could be the name of a license file the target is searching for. Without modifying the program, how might we create a new, mostly correct license with this name?