Lab: Cracking with IDA

Environment

- Windows Virtual Machine
- IDA

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

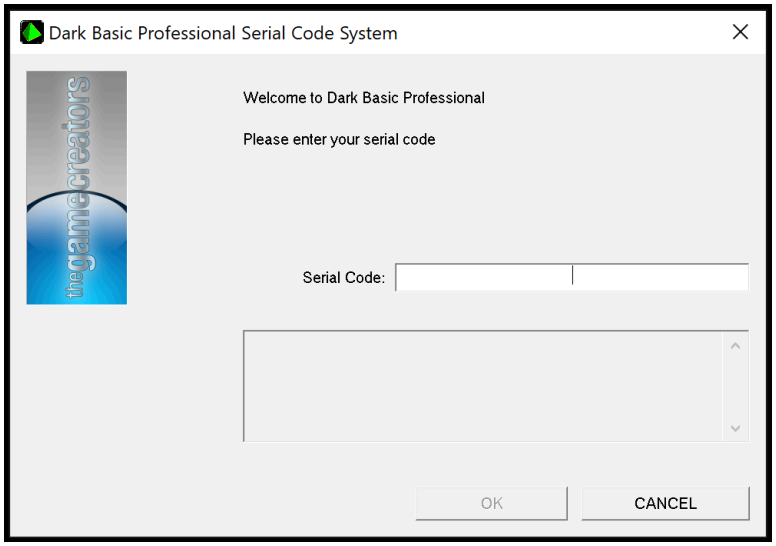
Target

- 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 the application uses. Next, we will examine the Serial check used by the software.



2. In this lab, we'll modify the program to remove the serial check, to allow continuing without a valid serial.
3. We will need to make modifications to the program, so, as always, make a backup of the original:
   - Navigate to the Dark-Basic-Pro program folder.
   - Make a backup copy of TGCSerial.exe, called TGCSerial.exe.orig. This will be the original file to restore from, in case we break the program in the process of modification.
4. Launch the IDA Free tool.
5. Select 'New – Disassemble a New File'
6. If prompted, select PE executable (this means an executable that runs on Windows).
7. Navigate to and select the TGCSerial.exe.
8. Leave all the default settings as is, and keep hitting "OK" or "Next" to proceed through the setup process. Click "OK" on any warnings, they will not impact our analysis.
9. Let IDA work while it processes the program. Watch the status bar on the bottom of the IDA window to see when analysis is finished. In general, beginning your analysis while IDA is still working may interfere with the results.

ISBN: 139419988

10. When the bottom text box shows 'The initial auto analysis has been finished, you are ready to begin.

Task 1

Attempt to use our previous "strings" strategy to find and circumvent the serial check. Remember, this is a large application, and tracing through its entire execution is infeasible. Instead, we need to find a way to isolate the code of interest.

- Enter an invalid serial and observe the output.
- Use "View > Open subviews > Strings" to view the strings used by the program. Can you find where the "invalid" string gets used?

You will commonly find strings embedded in a program, which can easily be used to statically locate code of interest. However, *this* program loads strings from a language file, which won't show up in static analysis.

Run the program in the IDA debugger:

- Use 'Debugger > Start Process'.
    - It will warn you this is about to run code on your computer (if this were a potentially malicious application, you might not want to run it).
    - In this case, you can trust the application, and accept the warning.
- Enter an invalid serial, and click "OK".
- Pause the program Be patient while pausing the process in the debugger, IDA may need to analyze newly loaded memory, which may take several minutes.
- Search for the "invalid" string in the process memory using "Search > Sequence of bytes". Generally, "Find all occurrences" is a good strategy.
- Focus on the strings that appear in the "debug" or ".data" sections, as opposed to strings in other DLLs. These are the copies in direct use by the target application.

Experiment with setting breakpoints on these strings to see if you can trace their usage in the program.

A different, sometimes better strategy is to focus on a string that we can control. Repeat the above process, this time entering a serial of your choosing (for example, "DAZZLECAT"). Use the debugger to find and follow this string in memory.

Reversing tips:

- Use IDA's default breakpoint settings.
- If IDA has an issue where the view changes when you click to set a breakpoint: remove the incorrect breakpoint, click the "Back" button, and try again.
- Use the debugger breakpoints window to manage your breakpoints. Disable breakpoints when not in use.
- When you hit a breakpoint, look at the current program address. If the address contains the phrase ".DLL", then the program is currently executing library code. While the library may be accessing the memory you set the breakpoint on, the serial check code itself is unlikely to be here. In this situation, use the debugger's call stack view to figure out what code in the target application called the library. Scroll to the top of the call stack to find the current frame, and gradually scroll down until you find an address in the target program. In the main view, navigate to this address.

While it is possible to find the serial check algorithm by tracing a fake serial through memory, it can be cumbersome due to the way the string is repeatedly copied around.

ISBN: 139419988

Good reverse-engineering involves being able to pivot and approach the challenge from many different angles. If you are struggling to locate the serial check code via the breakpoint/strings approach, proceed to Task 2 for an alternate strategy.

Task 2

Change strategies, and locate the key check with *static* analysis instead:

- Program keys and serials are commonly in specific formats, such as "ABCD-EFGH-IJKL-MNOP". In parsing these, it is common that programs use various format strings and modifiers ("%d", "%s", "%x", "%c", etc.), in conjunction with formatting functions like "sprintf" or "scanf".
- Using the IDA strings view ("View > Open subviews > Strings"), search for the "%" character to look for any format strings that appear in the program. Many of these are unlikely to be serial formats; for example, "CLSID\\%1" looks more like windows class ID information than a product serial. However, some of the results may be good serial candidates.
- Follow cross-references to any interesting format strings to locate the serial check function.
- After finding the serial check function, use the 010 tool to patch the program.
    - This lab uses IDA Free, which does not support patching (the professional version of IDA does)

Patching tips:

- The IDA disassembly view can sometimes be cluttered, or can sometimes not show *enough* information. You can show or hide the x86 machine code in IDA, by going to Options > General:
    - Selecting "Line Prefixes" will display the addresses of each line.
    - Putting an 8 in "Number of opcode bytes" will show up to 8 byes per instruction.
    - Line addresses and opcode information is valuable when you are ready to write a patch for cracking a program. But until then, they mostly just take up space, and are better left off in the meantime.

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

- Reverse engineer the serial check algorithm in the target. What information is it checking? Find a valid serial number for the program. Then find a hundred. "Watermark" your serials by generating only serials containing the text "1337".

ISBN: 139419988

Task 2 Hints

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

- Follow the string "%c%c%c%c%c%c%c%c" in the Strings window.
- Next to the string in the main window, you will see cross-references to the string, in the form "DATA XREF:". Double click a cross-reference to find where that string is used in code.
- Rather than patching this function (which is the serial check function), it is often easier to patch the *call* to the check function.
- Scroll to the top of the serial check function, and, next to the subroutine name, look for code cross-references in the form "CODE XREF:". Double click the code cross-reference to find who calls the serial check function.
- Use 010 to patch out the call to this function (remember nops), or modify the conditional "jz" that checks the return value from the function.

ISBN: 139419988