Review: BoundsChecker 4.0

Reviewed by Dave Jewell

BoundsChecker is now a well established tool, from NuMega Technologies, makers of the renowned SoftIce/W system-level debugger (a new version of which has recently been released for Windows NT). Although it's been around for a while, BoundsChecker will be relatively unknown to the Delphi programming community because, up until now, it has been specific to C/C++ application development. BoundsChecker 4.0 changes all that and is now fully compatible with Delphi 2.0. This review looks at a late beta (Release Candidate) of the Windows 95 version. A separate version is available for use under NT.

So What Exactly Is It?

NuMega refer to BoundsChecker as an automated error detection and analysis tool. You can think of it as an error detector which lurks in the background while your application is running. As soon as something goes wrong, BoundsChecker makes a note of the error and optionally gives an indication of the problem. At the end of the debugging session, you can review all the errors and make any required changes to your source files.

This is quite a different philosophy to normal debugging practice. As a rule, programmers only bother to debug a program when they know that there's something wrong with it! Conventional debugging involves narrowing down an already-known problem to a specific line of source code and determining exactly what's going wrong so that the source can be fixed.

BoundsChecker really comes after this point in the development cycle. When you think you've got all the bugs out of your program, that's the time to use BoundsChecker! It will often show you that there are still problems with your code. These problems can be quite obvious, or they can be very subtle. For instance, suppose that each time your program runs it consumes a few more bytes of precious system resources. From your perspective, the program runs flawlessly, but after being executed a few hundred times on a server, Windows will eventually die a horrible death.

This is just the sort of problem that BoundsChecker is particularly good at finding. Indeed, when BoundsChecker was first released, many C/C++ programmers believed that it wasn't working properly because it was reporting many problems with the OWL and MFC class libraries. It turned out that those problems were real: the most popular C++ class libraries had both resource and memory leaks! The real problem, of course, was that neither Borland nor Microsoft had had the benefit of BoundsChecker during their development – but you have no such excuse!

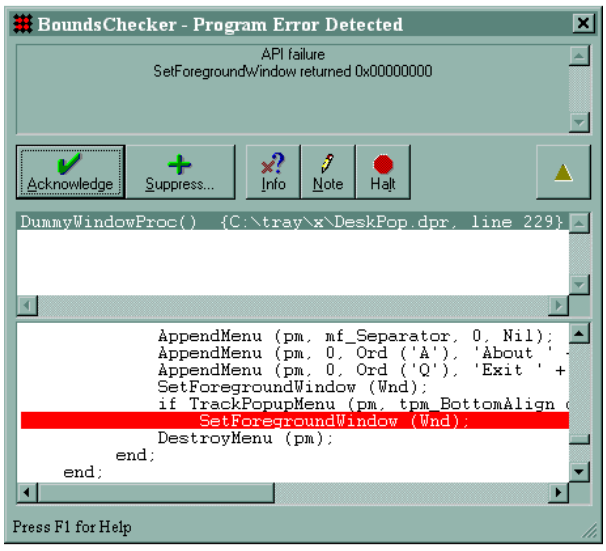
Installation And Usage

The Release Candidate I had for review ships on five diskettes and occupies around 12.5Mb of disk space. Installation is very straightforward and you can then immediately fire up Delphi 2.0 to find that three new options have been added to the `Tools` menu: `BoundsChecker`, `BoundsChecker Settings` and `BoundsChecker Help`.

To check a program, you simply invoke the `BoundsChecker` menu option and your program will start running automatically. Under normal circumstances, you will be checking for problems in your own code rather than in the Delphi runtime library or VCL classes. However, it is possible to rebuild the these libraries so that BoundsChecker has access to source-code information even within the RTL and VCL. Be warned, though, that if you want to do this, you'll need a copy of Borland's 32-bit assembler, TASM32.

Every time BoundsChecker encounters a problem in the running program a dialog similar to Figure 1 is displayed and execution is suspended until you've dealt with the dialog. If you wish, you can tell BoundsChecker not to display this dialog, but simply to record all errors for later examination. Pressing the `Acknowledge` button simply restarts execution of the program. If you encounter a large number of errors of the same type, you can use the `Suppress` button to tell BoundsChecker how you want to deal with future occurrences of the same error. The options here are:

- Suppress the error if it occurs in the same function,
- Suppress the error if it occurs in the same source file,



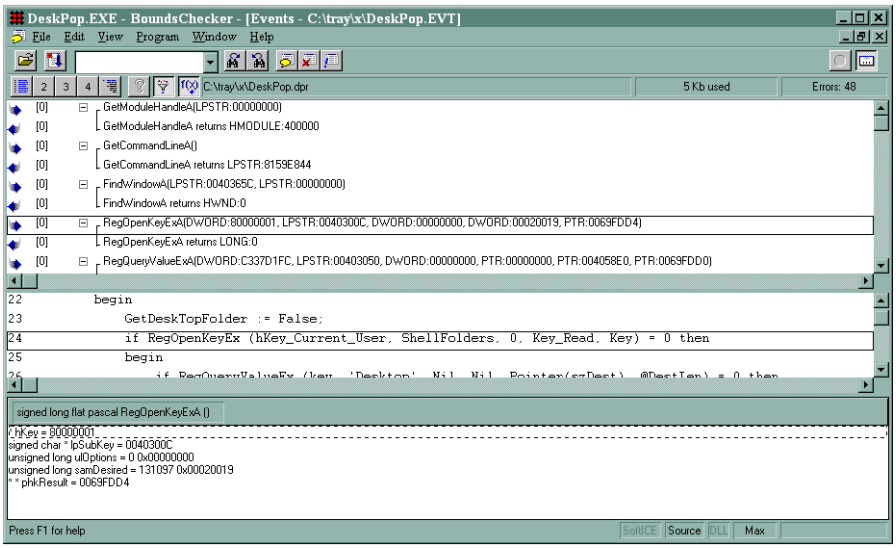
➤ *Figure 1*
Each time an error occurs in your running program, this dialog box appears. You can suppress this behaviour and elect to view all errors after the program has executed, or you can suppress certain categories of error only.

- Suppress the error if it occurs in the same executable,
- Suppress the error no matter what.

This prevents you being inundated with repeated error notifications when your program is sat in some enormous loop! Using the `BoundsChecker` menu item, you can get access to another set-up dialog which will allow you to modify your list of suppressed errors. This introduces the concept of *suppression libraries*, where you can select routines in the Visual C++ runtime library, OWL, MFC, Delphi run-time and even the Visual Basic 4.0 runtime interpreter. Another nice touch is the inclusion of a `Note` button which can be used to insert a note relating to the error that's just occurred. This will show up later when you examine the post-mortem log.

Once you've finished running your application, there are several different logs that you can open up, each of which appears as an MDI window within the main BoundsChecker application. The simplest of these is the list of Notifications window. At various points within the Windows kernel, low-level notification messages are generated when certain things happen. I'm not using the word 'message' in the normal Windows sense here. To hook these notifications, you have to call routines in TOOLHELP.DLL and this is presumably what BoundsChecker is doing. These messages are generated when a DLL is loaded or unloaded, when a new thread is created or destroyed and when processes start and terminate. Exceptions and calls to `OutputDebugString` (a low-level Windows API) are also included.

Unfortunately, BoundsChecker is a bit more invasive than it ought to be here: the notification file includes not only the DLLs, OCXs, etc used by your program, but also the modules loaded by BoundsChecker itself. This is a little confusing but you soon realise which are the BoundsChecker DLLs. This behaviour isn't mentioned in the documentation and it's possible that it only occurs when debugging Delphi applications.



➤ *Figure 2 The Event window*

The next window is the Event window (Figure 2). Events have a fairly loose definition and include calls to and returns from API routines, processing dialog and window messages and processing Windows hooks. BoundsChecker distinguishes between ordinary API events and OLE interface method calls, the latter having a separate icon. When you examine the various events that have taken place, you can see the parameters that were used to call a particular API routine and you can see the function result that was returned from the call. This is a big improvement on laboriously stepping through a program in a debugger to find what went wrong. As you select different events in the log, BoundsChecker will automatically bring into view and highlight the corresponding source code line. You can hide detail by showing different levels of event and you can use a Noise Reduction option to hide the operation of the normal message processing loop within a typical Windows application.

Error Detection

The Error window (Figure 3) gives a list of all the suspect things which BoundsChecker found. As an example, I discovered a small utility I'd written wasn't properly freeing up system resources. It's dead easy to spot this because BoundsChecker places a little tear drop symbol (a leak, get it?) next to every such problem in the Error window. Similarly, you get told when you pass invalid arguments to API routines or when an API call returns a function result indicating failure.

Of course, BoundsChecker can't always tell whether a failed API call is intentional. For instance, a program might try to open a file to check if it exists. A failed file open will be flagged as an error. You can get round this by altering the extensive array of filtering options BoundsChecker provides to ignore errors with certain function calls.

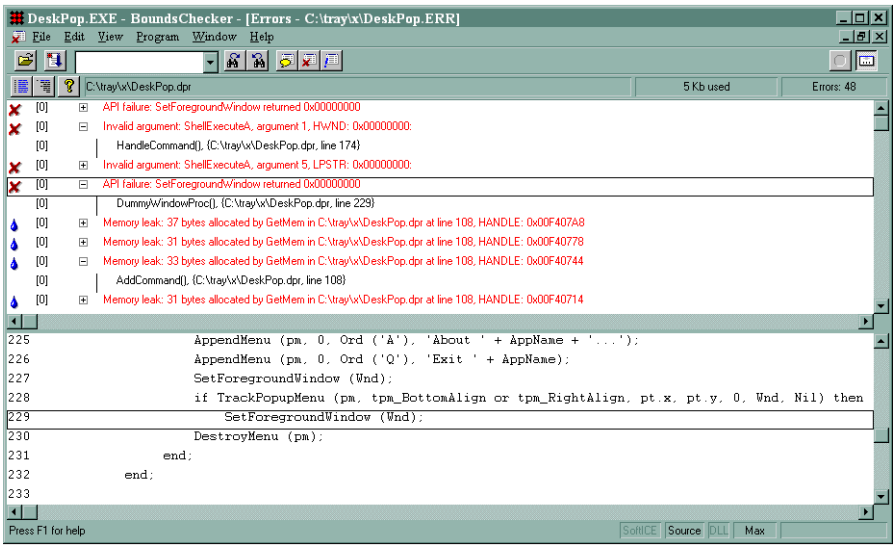
BoundsChecker isn't foolproof and will sometimes report problems that aren't there. As a real-world example, I've written a little program which (amongst other things) adds sub-menus to an existing menu structure. The program works by calling the `CreateMenu` API to create a new, empty sub-menu, adding items to the sub-menu and then using `InsertMenu` to attach the sub-menu to the existing parent menu. BoundsChecker notices the call to `CreateMenu` and apparently looks for a corresponding `DestroyMenu` call with the same menu handle value. It doesn't see it because I make just one call to `DestroyMenu` using the parent menu handle to automatically de-allocate all the attached sub-menus, something that BoundsChecker isn't aware of.

However, none of the above is intended as serious criticism. In fact, the Error window is the best part of BoundsChecker. Here's a

brief run-down of the vast number of different error types that the program is intended to catch:

- ***API Routine Checks*** Validates errors in over 3000 API functions including the complete Win32 API, DirectX, the new Internet APIs and the C/Delphi run-time libraries. For each API call, BoundsChecker tests for the correct number of arguments, bad or invalid arguments, invalid pointers, conflicting flags and more.
- ***Pointer Checking*** Detects improper use of null pointers, operations on uninitialised pointers, attempting to free global memory handles without performing a `GlobalUnlock` call and calling routines through a function pointer that hasn't been set up to point to valid code.
- ***Leak Checking*** There are two sorts of leaks you need to worry about: memory leaks and resource leaks. In addition to detecting memory leaks caused by Windows API routines, BoundsChecker will also find memory leaks resulting from improper use of the Delphi RTL library routines. Resource leaks result from a failure to de-allocate bitmaps, device contexts, menus or the various GDI-based drawing tools.
- ***OLECheck*** BoundsChecker has the ability to keep track of OLE-based interface method calls. Invalid return code and bad parameters are checked for and BoundsChecker also tracks of the number of times an interface is instantiated and released, generating an error if each interface doesn't have a corresponding `Release` call.
- ***Memory Access*** Validates memory access, checking that memory reads do not access uninitialised memory and checking that memory writes do not (for example) exceed the bounds of an array or dynamically allocated memory block.

In addition, BoundsChecker will also perform 'compliance' testing on a program. This is a excellent way of determining if your application will work under Windows 95 and NT. The built-in compliance monitor will auto-detect the use of API routines specific to either environment and this will be flagged in the generated compliance report.



➤ *Figure 3 The Error Log*

Custom Checks & Validation

In earlier BoundsChecker versions the underlying API validation code was fixed. With the new version, NuMega have introduced the concept of *validation modules*, making it possible to design custom validators for your own DLLs. Unfortunately, it presumes you have a C++ compiler: NuMega provide utilities which generate a C++ source file, which you then compile and link to create a validation module.

In the same way, BoundsChecker's on-line help information is C/C++ oriented. It is, however, of outstanding quality. A great deal of context-sensitive information is provided so that when you're looking at the Error log (for example), you can select an error item and press `F1` key to get information relating to the specific error that you've highlighted. Not only that, but the on-line help also provides sample code illustrating the error and guidance on how to fix it.

Conclusions

It's undeniable that BoundsChecker was originally developed for C/C++ developers and this emphasis continues to an extent. There are actually two different editions: Professional and Standard. However, as far as I can determine, the extra facilities of the Professional Edition are only applicable to C/C++ developers.

It's equally undeniable that BoundsChecker isn't cheap! However, if you're developing anything but the most basic Delphi application, it will more than pay for itself in allowing you to quickly locate program bugs and problems that will happily pass undetected through a conventional beta testing cycle. The end result is more robust software, fewer angry customers and less support calls. I'd go so far as to say that if you're developing commercial applications, you really shouldn't let it out the door until you've run it through BoundsChecker first.

BoundsChecker 4.0 is available in the UK from Programmer's Paradise on 0500 284 177 (Free Phone). The Standard Edition costs £395, the Professional costs £715. Because of the low level at which BoundsChecker hooks into an application, you must also specify whether you want the Windows 95 or Windows NT version.

Dave Jewell is a freelance consultant, programmer and technical journalist specialising in system-level Windows and DOS work. Contact Dave as DaveJewell@msn.com, DSJewell@aol.com or 102354,1572 on CompuServe