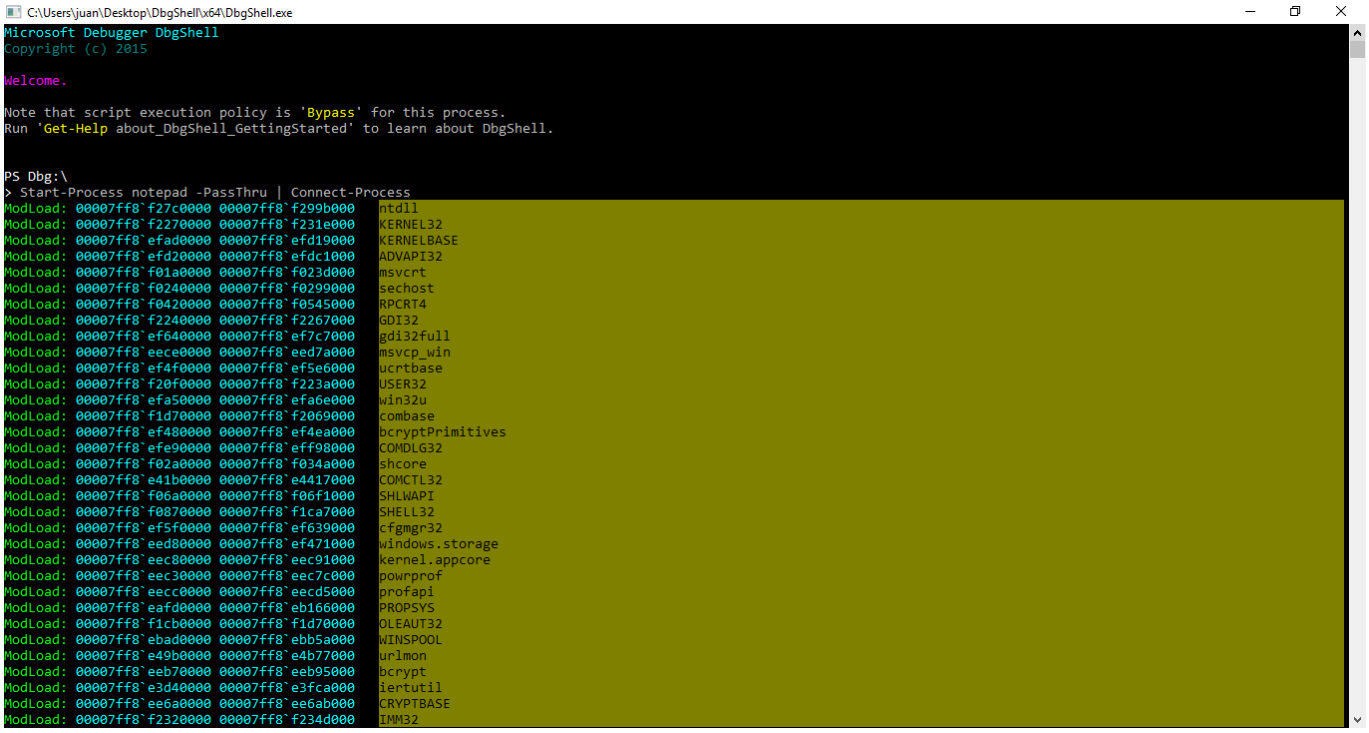
DbgShell

A PowerShell front-end for the Windows debugger engine.



Ready to tab your way to glory? For a quicker intro, take a look at Getting Started.

Binaries

https://aka.ms/dbgshell-latest

Motivation

Have you ever tried automating anything in the debugger? (cdb/ntsd/kd/windbg) How did that go for you?

The main impetus for DbgShell is that it's just waaaay too hard to automate anything in the debugger. There *are* facilities today to assist in automating the debugger, of course. But in my opinion they are not meeting people's needs.

- Using the built-in scripting language is arcane, limited, difficult to get right, and difficult to get help with.
- DScript is kind of neat, but virtually unknown, and it lacks a REPL, and it's too low-level.
- Writing a full-blown debugger extension DLL is very powerful, but it's a significant investment—way too expensive for solving quick, «one-off» problems as you debug random, real-world problems. Despite the cost, there are a large number of debugger extensions in existence. I think there should not be nearly so many; I think the only reason there are so many is because there aren't viable alternatives.
- Existing attempts at providing a better interface (such as PowerDbg) are based on «scraping» and text parsing, which is hugely limiting (not to mention idealogically annoying) and thus are not able to fulfill the promise of a truly better interface (they are only marginally better, at best).
- Existing attempts to provide an easier way to write a debugger extension are merely a stop-gap addressing the pain of developing a debugger extension; they don't really solve the larger problem. (for instance, two major shortcomings are: they are still too low-level (you have to deal with the dbgeng COM API), and there's no REPL)
- The debugger team has recently introduce Javascript scripting. Javascript is a much better (and more well-defined) language than the old windbg scripting language, but I think that PowerShell has some advantages, the largest of which is that nobody really uses a Javascript shell—PowerShell is much better as a combined shell *and* scripting language.

The goal of the DbgShell project is to bring the goodness of the object-based PowerShell world to the debugging world. When you do 'dt' to dump an 'object', you should get

an *actual* object. Scripting should be as easy as writing a PowerShell script.

The DbgShell project provides a PowerShell front-end for dbgeng.dll, including:

- a managed «object model» (usable from C# if you wished), which is higher-level than the dbgeng COM API,
- a PowerShell «navigation provider», which exposes aspects of a debugging target as a hierarchical namespace (so you can «cd» to a particular thread, type «dir» to see the stack, «cd» into a frame, do another «dir» to see locals/registers/etc.),
- cmdlets for manipulating the target,
- a custom PowerShell host which allows better control of the debugger CLI experience, as well as providing features not available in the standard powershell.exe host (namely, support for text colorization using ANSI escape codes (a la ISO/IEC 6429))

The custom host is still a command-line (conhost.exe-based) program (analogous to ntsd/cdb/kd), but it can be invoked from windbg (!DbgShell).

In addition to making automation much easier and more powerful, it will address other concerns as well, such as ease of use for people who don't have to use the debuggers so often. (one complaint I've heard is that «when I end up needing to use windbg, I spend all my time in the .CHM»)

For seasoned windbg users, on the other hand, another goal is to make the transition as seamless as possible. So, for instance, the namespace provider is not the only way to access data; you can still use traditional commands like «~3 s«, «k«, etc.

Mor information

https://github.com/Microsoft/DbgShell