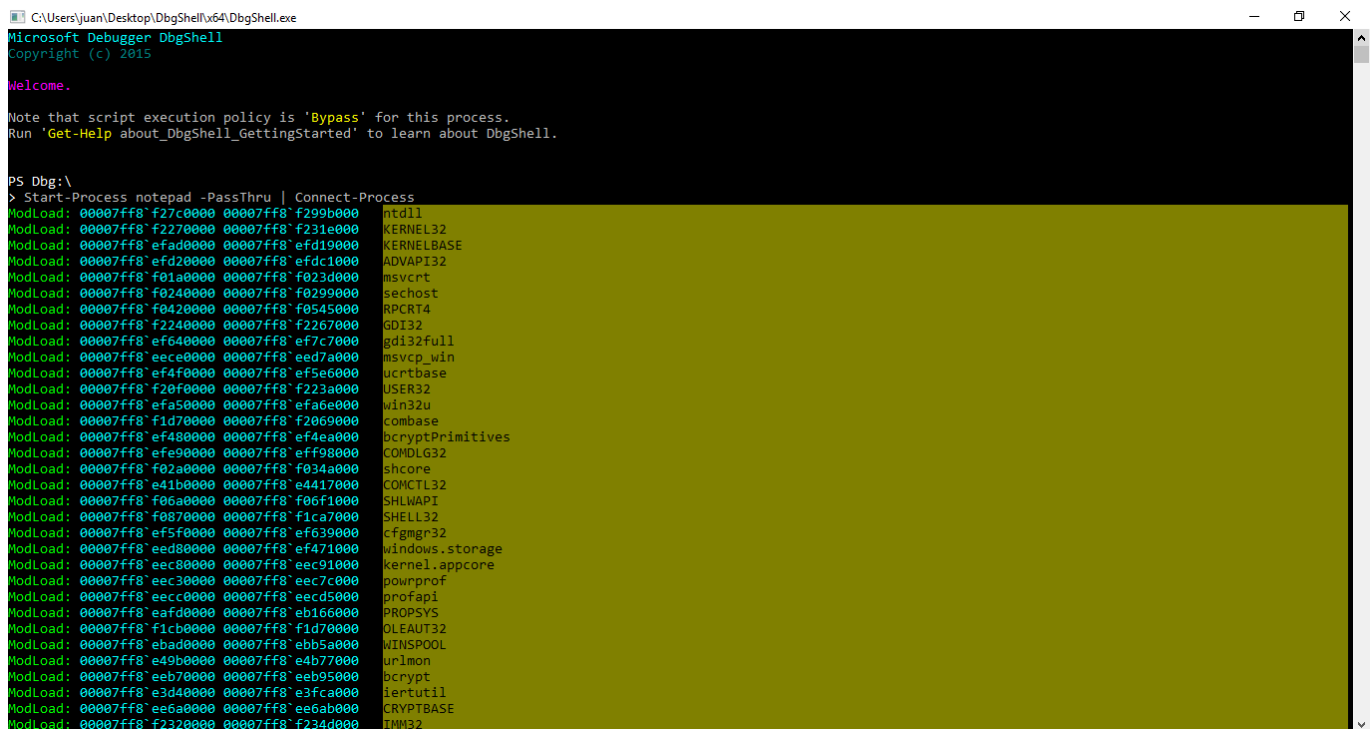


DbgShell

A PowerShell front-end for the Windows debugger engine.

A screenshot of a Windows PowerShell console window titled "Microsoft Debugger DbgShell". The window shows the following text: "Welcome.", "Note that script execution policy is 'Bypass' for this process.", "Run 'Get-Help about_DbgShell_GettingStarted' to learn about DbgShell.", and a PowerShell prompt "PS Dbg:\". Below the prompt, a command is entered: "> Start-Process notepad -PassThru | Connect-Process". This command triggers a list of loaded modules, each with two memory addresses and the module name. The modules listed include: ntdll, KERNEL32, KERNELBASE, ADVAPI32, msvcrt, sechost, RPCRT4, GDI32, gdi32full, msvc_p_win, ucrtbase, USER32, win32u, combase, bcryptPrimitives, COMDLG32, shcore, COMCTL32, SHLWAPI, SHELL32, cfmgr32, windows.storage, kernel.appcore, powrprof, profapi, PROPSYS, OLEAUT32, WINSPOOL, urlmon, bcrypt, iertutil, CRYPTBASE, and IMM32.

```
C:\Users\juan\Desktop\DbgShell\64\DbgShell.exe
Microsoft Debugger DbgShell
Copyright (c) 2015

Welcome.

Note that script execution policy is 'Bypass' for this process.
Run 'Get-Help about_DbgShell_GettingStarted' to learn about DbgShell.

PS Dbg:\
> Start-Process notepad -PassThru | Connect-Process
ModLoad: 00007ff8`f27c0000 00007ff8`f299b000 ntdll
ModLoad: 00007ff8`f2270000 00007ff8`f231e000 KERNEL32
ModLoad: 00007ff8`efad0000 00007ff8`efd19000 KERNELBASE
ModLoad: 00007ff8`efd20000 00007ff8`efdc1000 ADVAPI32
ModLoad: 00007ff8`f01a0000 00007ff8`f023d000 msvcrt
ModLoad: 00007ff8`f0240000 00007ff8`f0299000 sechost
ModLoad: 00007ff8`f0420000 00007ff8`f0545000 RPCRT4
ModLoad: 00007ff8`f2240000 00007ff8`f2267000 GDI32
ModLoad: 00007ff8`ef640000 00007ff8`ef7c7000 gdi32full
ModLoad: 00007ff8`eece0000 00007ff8`eed7a000 msvc_p_win
ModLoad: 00007ff8`ef4f0000 00007ff8`ef5e6000 ucrtbase
ModLoad: 00007ff8`f20f0000 00007ff8`f223a000 USER32
ModLoad: 00007ff8`efa50000 00007ff8`efa6e000 win32u
ModLoad: 00007ff8`f1d70000 00007ff8`f2069000 combase
ModLoad: 00007ff8`ef480000 00007ff8`ef4ea000 bcryptPrimitives
ModLoad: 00007ff8`efe90000 00007ff8`eff98000 COMDLG32
ModLoad: 00007ff8`f02a0000 00007ff8`f034a000 shcore
ModLoad: 00007ff8`e41b0000 00007ff8`e4417000 COMCTL32
ModLoad: 00007ff8`f06a0000 00007ff8`f06f1000 SHLWAPI
ModLoad: 00007ff8`f0870000 00007ff8`f1ca7000 SHELL32
ModLoad: 00007ff8`ef5f0000 00007ff8`ef639000 cfmgr32
ModLoad: 00007ff8`eed80000 00007ff8`ef471000 windows.storage
ModLoad: 00007ff8`eec80000 00007ff8`eec91000 kernel.appcore
ModLoad: 00007ff8`eec30000 00007ff8`eec7c000 powrprof
ModLoad: 00007ff8`eecd0000 00007ff8`eecd5000 profapi
ModLoad: 00007ff8`eafd0000 00007ff8`eb166000 PROPSYS
ModLoad: 00007ff8`f1cb0000 00007ff8`f1d70000 OLEAUT32
ModLoad: 00007ff8`ebad0000 00007ff8`ebb5a000 WINSPOOL
ModLoad: 00007ff8`e49b0000 00007ff8`e4b77000 urlmon
ModLoad: 00007ff8`eeb70000 00007ff8`eeb95000 bcrypt
ModLoad: 00007ff8`e3d40000 00007ff8`e3fca000 iertutil
ModLoad: 00007ff8`ee6a0000 00007ff8`ee6ab000 CRYPTBASE
ModLoad: 00007ff8`f2320000 00007ff8`f234d000 IMM32
```

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 ideologically 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>