BPMTK V0.1.5.0

(Basic Process Manipulation Toolkit)
https://DidierStevens.com

Introduction

The bpmtk has a series of commands to manipulate processes. It consists of one executable, bpmtk.exe, which takes two optional arguments.

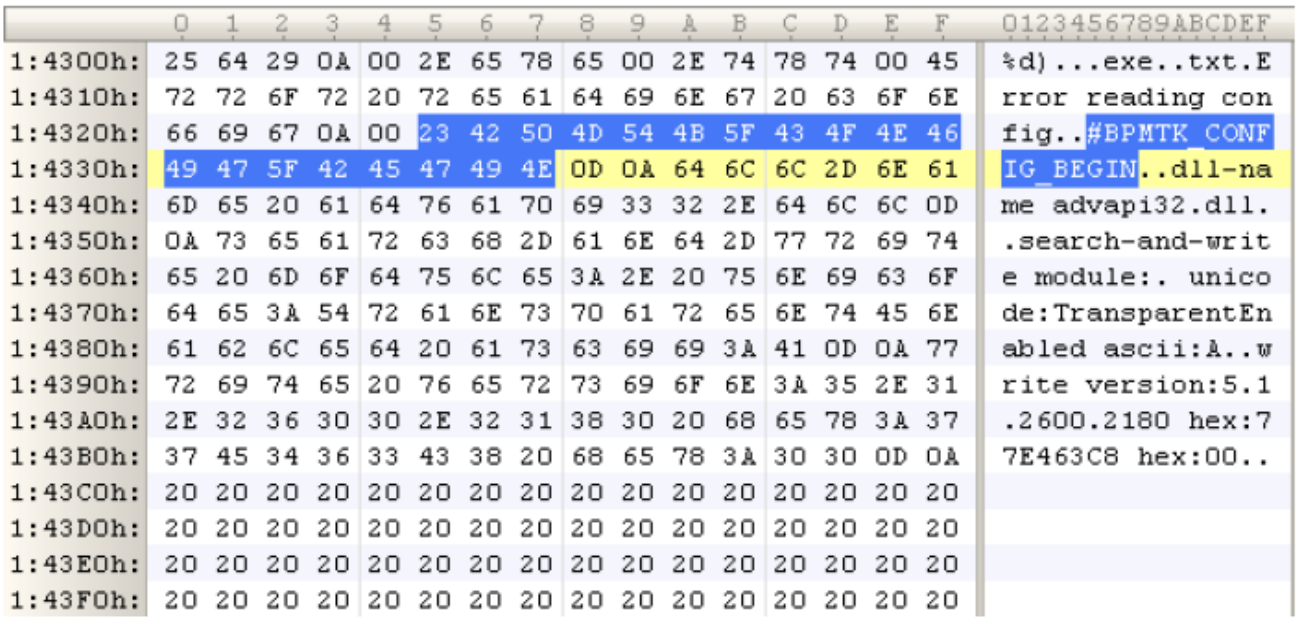
bpmtk [configfile [exefile]]

The first argument is the name of the config file that defines the commands to be executed by the toolkit. If no argument is supplied, the executable will first look if there is an embedded config file (embedded as a resource) and use this. If there is no embedded config file, it will try to open a config file with the same name as the executable, but with extension .TXT, in the working directory. E.g., starting bpmtk.exe without argument will make it open bpmtk.txt. The second optional argument specifies the name of the file to create a version of bpmtk with an embedded config file. Example: bpmtk.exe disable-srp.txt no-srp.exe

This command will create a new version of bpmtk.exe, called no-srp.exe. The only difference between no-srp.exe and bpmtk.exe, is that no-srp.exe contains config file disable-srp.txt. The script is parsed before it is embedded as a resource, allowing you to catch syntax errors. When no-srp.exe is executed without arguments, it will execute the embedded config file. This is handy if you want to execute bpmtk on a remote system with psexec (sysinternals) where you can only transfer 1 file.

Bpmtk is a command-line application, all output is written to the standard output of the console, except for the error messages when parsing the config file, which are written to standard error output.

Starting from V0.1.3.0, a DLL version of bpmtk is also provided (bpmtk.dll). This DLL will execute the config file when the DLL is loaded (i.e. LoadLibrary). The config file is embedded inside the DLL as a string. To change the config file, recompile the DLL or use a binary editor to insert your script between the strings #BPMTK_CONFIG_BEGIN and #BPMTK_CONFIG_END.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
1:4300h:	25	64	29	0A	00	2E	65	78	65	00	2E	74	78	74	00	45	%d)...exe..txt.E
1:4310h:	72	72	6F	72	20	72	65	61	64	69	6E	67	20	63	6F	6E	rror reading con
1:4320h:	66	69	67	0A	00	23	42	50	4D	54	4B	5F	43	4F	4E	46	fig..#BPMTK_CONF
1:4330h:	49	47	5F	42	45	47	49	4E	0D	0A	64	6C	6C	2D	6E	61	IG_BEGIN..dll-na
1:4340h:	6D	65	20	61	64	76	61	70	69	33	32	2E	64	6C	6C	0D	me advapi32.dll.
1:4350h:	0A	73	65	61	72	63	68	2D	61	6E	64	2D	77	72	69	74	.search-and-writ
1:4360h:	65	20	6D	6F	64	75	6C	65	3A	2E	20	75	6E	69	63	6F	e module:. unico
1:4370h:	64	65	3A	54	72	61	6E	73	70	61	72	65	6E	74	45	6E	de:TransparentEn
1:4380h:	61	62	6C	65	64	20	61	73	63	69	69	3A	41	0D	0A	77	abled ascii:A..w
1:4390h:	72	69	74	65	20	76	65	72	73	69	6F	6E	3A	35	2E	31	rite version:5.1
1:43A0h:	2E	32	36	30	30	2E	32	31	38	30	20	68	65	78	3A	37	.2600.2180 hex:7
1:43B0h:	37	45	34	36	33	43	38	20	68	65	78	3A	30	30	0D	0A	7E463C8 hex:00..
1:43C0h:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
1:43D0h:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
1:43E0h:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
1:43F0h:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	

Configuration file

The config file defines a target (the process(es) to manipulate), some options and commands to manipulate the process(es). Each statement requires one line in the text file. Empty lines are allowed, and the line-comment character is #.

There are 4 mutually exclusive statements to define the target. By convention, the target statement is the first statement of the config file.

process-name PROCESS_NAME

the target is a process with name PROCESS_NAME. If PROCESS_NAME is *, all processes becomes targets.

example:

process-name cmd.exe

pid PID

the target is a process with process-id PID

example:

pid 1234

dll-name DLL_NAME

the target is each process that has loaded DLL with name DLL_NAME (don't use a path)

example:

dll-name advapi32.dll

start PROGRAM_NAME

this will create a new process by starting executable PROGRAM_NAME in a suspended state, inject kernel32.dll to force the loading of the DLLs, execute the process manipulation on the suspended process and finally resume the suspended process.

PROGRAM_NAME can contain a full pathname. If you want to specify the current directory, but

don't know is, prefix PROGRAM_NAME with currentdirectory: (e.g. currentdirectory:hack.exe).
This command has only been tested with cmd.exe
example:
start cmd.exe

Option statement verbose controls the verbosity of the output of the toolkit.
Add statement “verbose 1” to increase the verbosity. “verbose 0” is the default.

Option statement readonly instructs the toolkit to execute its commands without changing the target processes. This is useful for debugging and reconnaissance.

Option statement disable-console-output does exactly what it says, except for the parsing error messages. Watch out when combining this option with command pause or confirm, because you will not see the prompt.

Option statement output-to-file outputs all messages to the console and to a file. If you don't provide a filename, bpmtk will write to a file in the current directory, with filename bpmtk-yyyymmdd-hhmmss.txt. You can guess that yyyymmdd-hhmmss is a placeholder for the current date and time.

Option statement plugin allows you to load a plugin that defines functions to filter the output of the strings statement. Read the strings statement for further details. Plugin takes the filename of the plugin to load as an argument.

Option repeat wait [count] will execute the statements every “wait” seconds, at infinite. Unless you specify a “count”. Be careful with this command, bpmtk is still beta and there could be memory leaks in the program.

After the target and the options, bpmtk will execute the command statements in the order they appear in the config file.

pause
This command will pause the execution of the config file until you press return, whereupon it will continue with the next command.

confirm
This command will pause the execution of the config file until you enter Y (return is a shortcut for Y), whereupon it will continue with the next command. If you type N, bpmtk will abort the execution of the config file and start again with the next target. If there is no next target (for example, you specified only 1 target), bpmtk terminates.

print [text]
This is a trivial command, it displays the text you specified. If no text is provided, it displays an empty line. Use placeholders $date and $time in the text to display the current date and time.

info
Info will display some information: the version of the OS, the current directory of bpmtk, machinename and username, ... and a list of running processes (PID and processname). A missing process name indicates that you don't have the rights to manipulate that process.

adjust-token-privileges me

If you want to manipulate processes running under another user account than your account (or more precisely, the user account executing bpmtk), you need to enable the debug privilege with this command. By default, only local admins have the debug privilege, and hence, it can only be enabled for local admins. Issue this command (only once) before executing other commands like write and search-and-write.

suspend

This command will suspend all threats in the target process, effectively freezing the program. Suspending a thread in Windows suspends the threat and increases its suspend count with one.

resume

This command will resume all threats in the target process. Resuming a thread in Windows decreases its suspend count with one. If the count reaches zero, the thread is resumed.
Thus if you suspend a suspended thread, it will not be resumed when you use the resume command. Issuing several consecutive suspend commands requires that you issue the same amount of consecutive resume commands to effectively resuming the target process.
Use suspend and resume if you want to freeze a process during manipulation, for example when dumping the process memory.

dump

This command will dump the process memory of the target process. Each committed virtual memory page will be written to a file with filename PPP-AAAAAAAA.dump, where PPP is the process-id and AAAAAAAA is the base address of the page. A summary of all pages of the process will be written to file PPP.dump. All files are written to the current directory. In this version, guarded pages will not be dumped. Dump will also attempt to read guarded pages.
The process remains running when its memory is dumped. Consider surrounding the dump command with suspend/resume commands if you need to freeze the process while dumping its memory, like this:

```
suspend
dump
resume
```

write [version:version-number] hex:address [hex|ascii|unicode]:value

This command allows you to write into the memory of the target process(es). The first argument is the hexadecimal address to start writing (it must be prefixed with the constant hex:) The second argument is the data to write into the memory starting at the given address. The data can be expressed as a series of hexadecimal bytes (use prefix hex:), as an ascii string (no space characters allowed, use prefix ascii:) or as an unicode string (no space characters allowed, use prefix unicode:).
If the virtual page is protected against writing, VirtualProtectEx is called to change the protection.
If a version number is provided, the address has to be inside the memory space of a module with that same version number, otherwise the memory will not be written to. Use this option if the address you're writing to is different from version to version. The following example disables Microsoft .NET Code Access Security for all running .NET programs by writing value 1 to the appropriate variable. I looked up the address of the variable for 3 different versions of mscorwks.exe. Only one write command will effectively write to the memory (the one with the matching DLL version).

```
dll-name mscorwks.exe
write version:2.0.50727.42 hex:7A3822B0 hex:01000000
write version:2.0.50727.832 hex:7A38716C hex:01000000
write version:2.0.50727.1433 hex:7A3AD438 hex:01000000
```

search-and-write [module:module-name|*|.] [memory:writable] [hex|ascii|unicode]:value [hex|ascii|unicode]:value
This command allows you to search for a given sequence of bytes in the memory of the target process(es), and overwrite that sequence. If no module is specified, the command targets the virtual memory pages of the process. Use option memory:writable to limit the search to writable virtual pages. When a module-name is is specified, it will search in the provided module-name. A * performs a search in all loaded modules. A . performs a search in the target module: if the target is a process, search-and-write operates on the executable; if it is a DLL, it operates on the DLL. The module and memory option are mutually exclusive.
The first argument is the sequence we are looking for. This can be expressed in hexadecimal, ascii or unicode format (like for the write command). Each time this sequence is found in the module, the byte sequence specified by the second argument is written to memory starting at the address of the sequence we found. This too can be expressed in hexadecimal, ascii or unicode format. The size of the second byte sequence is independent of the size of the first sequence. For example, you can search for CMD and just write A, resulting in AMD.
If you just want to search for a sequence, but don't want to change it, use option readonly.
If the virtual page is protected against writing, VirtualProtectEx is called to change the protection.

inject-dll [currentdirectory:]dll_name
This command allows you to inject DLL with name dll_name in the target process(es). Since the target process will look (by default) for the DLL in system32 or in its own current directory, it's preferable to provide the full pathname to the dll. This will ensure that the process will find your DLL. If the DLL is in the current directory of bpmtk, but you don't know which directory this, prefix the name with currentdirectory:.

Reject-dll dll_name
This command allows you to free a DLL from the target process(es). You cannot provide a full pathname, only a filename. You are not restricted to freeing DLLs that have been injected with the toolkit. The DLL will be unloaded if its reference count is zero (no other module needs the DLL).

strings [module:module-name|*] [memory:writable] [address:[on|off]] [minimum-length:number] [alpha-percentage:number] [start-address:address] [end-address:address] [regex:expression] [filter:plugin-function]
This command will display all printable strings found in the memory of the target process.
The module and memory options are the same as for the search-and-write command.
The address option controls the display of address and type (A for ASCII, U for UNICODE) of the strings.
Displayed strings must have a minimum length specified by the minimum-length option (default 4).
You can limit the inspected memory range by specifying a start-address (default 0x00000000) and/or a end-address (default 0xFFFFFFFF).
The alpha-percentage is an option I'm tinkering with. It's my attempt to limit the command from displaying nonsense strings. Alpha-percentage specifies the percentage of characters in the string that must be alphabetical or white-space. The default value is 0. If you're searching for text inside a process, consider setting this option to a high value, like 75, to filter-out strings that are unlikely to be natural language text.
The last option, regex, allows you to specify a regular expression that must match the strings to be displayed, allowing you to further filter the output. When you use this option, the strings command will only display strings that contain the string you provided for the regex option (this is case-sensitive).

I use PCRE for regex matching, which is included via the pcre.dll. BPMTK will load this dll dynamically when you specify a regex option. If this dll cannot be found, the regex option will be downgraded to a simple substring search.
With the filter option, you can also use your own C function to filter the output. Compile a DLL with a function with the following signature:

__stdcall BOOL FilterFunction(char *pszString)

This function is called for each string found in the process, and must return TRUE to get the string displayed and FALSE to ignore it. Example: you compiled a DLL named audit.dll with a function called BankAccount to search for bank account numbers in memory. Issue these commands to use this plugin:

plugin audit.dll
strings filter:BankAccount

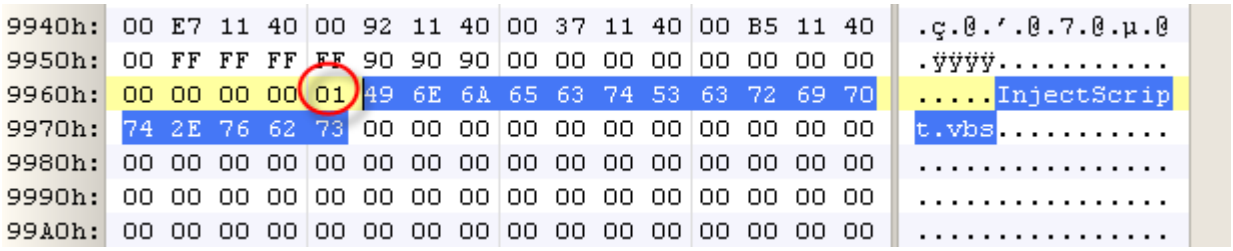
UNICODE strings are converted to ASCII strings (simply by retaining the least significant byte) before they are passed to the regex and filter options.

inject-code [minimum-length:size] [execute:yes|no] [hex|ascii|unicode:file]:shellcode
This command allows you to inject shellcode in the target process(es). By default, the injected shellcode is executed (CreateRemoteThread), but option execute:no allows you to prevent execution (useful to debug shellcode). Option minimum-length allows you to specify to inject the shellcode in a memory space of at least the size specified by minimum-length. Use it when your shellcode needs some extra memory to execute properly.

test-function
This command should not be used, I use it to develop and test new features.

Injecting VBScript
I've worked out a way to inject VBScript in a process (regardless of the program's support for VBA). The trick is to inject a special DLL (InjectScript.dll) inside a process, that will then create a COM instance of the VBScript engine. This engine is then instructed to execute our injected VBScript. This allows you to prototype your injected code in VBScript in stead of C/C++.

The VBScript to execute can be embedded in the InjectScript.dll DLL or the filename of the VBScript can be embedded (with absolute or relative path). By default, InjectScript.dll will execute VBScript InjectScript.vbs found in the working directory of the process.
To embed a VBScript inside InjectScript.dll, update the source code and recompile, or use a hex editor. Search for string InjectScript.vbs. The byte before InjectScript.vbs has to be 1 or 2.
Value 1 instructs InjectScript.dll to load the script with the filename following it. This filename (with optional path, relative or absolute) has to be ASCII.
Value 2 instructs InjectScript.dll to use the text following it as the VBScript to execute (i.e. the embedded script). This script has to be UNICODE.



Because VBScript cannot execute win32 calls (contrary to VBA), I added some custom methods. These methods are exposed by the scriptengine object.

Output takes one string argument. This string is passed to OutputDebugString.

Peek takes one numerical argument, the address, and returns the byte found at this address. If the address is not mapped, -1 is returned.

Poke takes two numerical arguments: and address and a byte. It will write the byte at the specified address. Poke will not raise an error if the address cannot be written to.

Suspend will suspend all threads of the process, except the thread of the VBScript engine.

Resume will resume all threads of the process, except the thread of the VBScript engine.

Examples
Some examples of config files:

info
pause
This will display the information and wait for you to press the return key.

start cmd.exe
search-and-write module:. unicode:DisableCMD hex:41

This will start cmd.exe in a suspended state, search for string DisableCMD and overwrite the D with A (41), effectively renaming it to AisableCMD. And then cmd.exe will be resumed. The effect is that cmd.exe will run, even if the policy has been set to disable it. DisableCMD is the name of the registry key that disables the execution of cmd.exe. But since now cmd.exe is looking for AisableCMD, it will not find it and conclude that there is no policy and hence that it is allowed to run.

dll-name advapi32.dll
#rename TransparentEnabled to AransparentEnabled
search-and-write mod:. unicode:TransparentEnabled ascii:A
#set _g_bInitializedFirstTime to 0
write hex:77E463C8 hex:00
pause

This config file will look for each process that has loaded advapi32.dll. The commands will be execute for each process, with a pause for each process. The TransparentEnabled registry key

defines if software restriction policies have been defined. Each time a new process is created, the parent process will read this key to determine if SRP are enabled and thus if it needs to analyse if it is allowed to create the new process. If the parent process doesn't find this registry key, it will assume that no SRP are enabled and that it is allowed to create the new process. So we mislead the parent process by renaming the registry it is looking for (search-and-write unicode:TransparentEnabled ascii:A).
For performance reasons, the parent process will not access the registry each time it creates a new process, but it will cache the registry data and only update the cache when it believes the registry data has been modified. This can be forced with GPUPDATE /force, but if you cannot execute GPUPDATE /force, us the following trick. I discovered that I can invalidate the cache data by setting variable _g_bInitializedFirstTime to zero. For advapi32.dll of Windows XP SP2, this variable is located at address 77E463C8. Hence the command write hex:77E463C8 hex:00

start cmd.exe
inject-dll hook-cmd.dll
This will start cmd.exe in a suspended state, inject dll hook-cmd.dll, and then cmd.exe will be resumed. The injected DLL will place a hook on the API function that reads a string from the registry (RegQueryValueEx). This hook will call the original function, except if the process reads value DisableCMD, then it will return “key not found”. Like the second example, this will also allow the execution of cmd.exe ignoring the policy.

process-name notepad.exe
verbose 1
inject-dll injectscript.dll
This will inject special DLL InjectScript.dll inside process notepad.exe, thereby executing VBScript InjectScript.vbs:

```
address = &h7FFDF000
MsgBox "Hello from inside the InjectScript.dll"
scriptengine.Output("Some text output from inside the InjectScript.dll")
scriptengine.Suspend
MsgBox "Suspended"
MsgBox scriptengine.Peek(address)
scriptengine.Poke address, 67
MsgBox scriptengine.Peek(address)
MsgBox scriptengine.Peek(0)
scriptengine.Poke 0, 67
scriptengine.Resume
```

Peeking and poking at address 0 will not succeed (peek(0) will return -1), but won't raise an error.

process-name notepad.exe
verbose 1
inject-code
hex:E8000000005B8DB334010000568DB33001000056680100000068888F0300E8190000006800000008D8338010000505068000000000FF9334010000C35589E55156578B4D0C8B75108B7D14FF36FF7508E819000000890781C70400000081C604000000E2E65F5E5989EC5DC210005589E55356575164FF3530000000588B400C8B480C8B118B413068020000008B7D085750E86A00000085C0740789D1E9E1FFFFFF8B4118508B583C01D88B5878585001C38B4B1C8B53208B5B2401C101C201C38B3

2585001C66801000000FF750C56E82C00000085C0741181C20400000081C302000000E9D7FFFFFF
5831D2668B13C1E20201D10301595F5E5B89EC5DC208005589E551535231C931DB31D28B45088
A1080CA6001D3D1E30345108A0884C9E0EE31C08B4D0C39CB7401405A5B5989EC5DC20C001
AB8060000000000048656C6C6F2066726F6D20696E6A65637465642073686656C6C20636F64652100
pause
This will inject shellcode in notepad.exe