

# Looking at Processes, Modules, and Threads with Powershell 2.0 Part I

Ryan M. Ferris RMF Network Security rev.07/30/10

These are some notes I have gathered on querying objects concerning process,modules,threads in PowerShell. This paper is primarily concerned with comparing changes in loaded processes, modules, and threads over time. My notes concern these TypeName's in Powershell 2.0:

- System.Diagnostics.Process
- System.Diagnostics.ProcessModuleCollection
- System.Diagnostics.ProcessThreadCollection
- System.Management.ManagementObject#root\cimv2\Win32\_Process
- System.Management.ManagementObject#root\cimv2\Win32\_Thread

## PS (Get-Process)

The 'Get-Process' cmdlet (alias 'ps') for the .NET class **System.Diagnostics.Process** has both Module and Thread collection derived classes. The 'ps' cmdlet has 51 properties including the ProcessModule and the ProcessThread Collections:

```
$a=ps
```

```
$a | gm | ? {$_.MemberType -eq "Property"}
```

```
TypeName: System.Diagnostics.Process
```

Name	MemberType	Definition
BasePriority	Property	System.Int32 BasePriority {get;}
Container	Property	System.ComponentModel.IContainer Container {get;}
EnableRaisingEvents	Property	System.Boolean EnableRaisingEvents {get;set;}
ExitCode	Property	System.Int32 ExitCode {get;}
ExitTime	Property	System.DateTime ExitTime {get;}
Handle	Property	System.IntPtr Handle {get;}
HandleCount	Property	System.Int32 HandleCount {get;}
HasExited	Property	System.Boolean HasExited {get;}
Id	Property	System.Int32 Id {get;}
MachineName	Property	System.String MachineName {get;}
MainModule	Property	System.Diagnostics.ProcessModule MainModule {get;}
MainWindowHandle	Property	System.IntPtr MainWindowHandle {get;}
MainWindowTitle	Property	System.String MainWindowTitle {get;}
MaxWorkingSet	Property	System.IntPtr MaxWorkingSet {get;set;}
MinWorkingSet	Property	System.IntPtr MinWorkingSet {get;set;}
Modules	Property	System.Diagnostics.ProcessModuleCollection Modules {get;}
NonpagedSystemMemorySize	Property	System.Int32 NonpagedSystemMemorySize {get;}
NonpagedSystemMemorySize64	Property	System.Int64 NonpagedSystemMemorySize64 {get;}
PagedMemorySize	Property	System.Int32 PagedMemorySize {get;}
PagedMemorySize64	Property	System.Int64 PagedMemorySize64 {get;}
PagedSystemMemorySize	Property	System.Int32 PagedSystemMemorySize {get;}
PagedSystemMemorySize64	Property	System.Int64 PagedSystemMemorySize64 {get;}
PeakPagedMemorySize	Property	System.Int32 PeakPagedMemorySize {get;}

All information and methods subject to revision and not guaranteed. Use at your own risk.

Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

```

PeakPagedMemorySize64    Property System.Int64 PeakPagedMemorySize64 {get;}
PeakVirtualMemorySize    Property System.Int32 PeakVirtualMemorySize {get;}
PeakVirtualMemorySize64  Property System.Int64 PeakVirtualMemorySize64 {get;}
PeakWorkingSet           Property System.Int32 PeakWorkingSet {get;}
PeakWorkingSet64         Property System.Int64 PeakWorkingSet64 {get;}
PriorityBoostEnabled      Property System.Boolean PriorityBoostEnabled {get;set;}
PriorityClass             Property System.Diagnostics.ProcessPriorityClass PriorityClass {get;set;}
PrivateMemorySize        Property System.Int32 PrivateMemorySize {get;}
PrivateMemorySize64      Property System.Int64 PrivateMemorySize64 {get;}
PrivilegedProcessorTime  Property System.TimeSpan PrivilegedProcessorTime {get;}
ProcessName              Property System.String ProcessName {get;}
ProcessorAffinity         Property System.IntPtr ProcessorAffinity {get;set;}
Responding               Property System.Boolean Responding {get;}
SessionId                Property System.Int32 SessionId {get;}
Site                     Property System.ComponentModel.ISite Site {get;set;}
StandardError            Property System.IO.StreamReader StandardError {get;}
StandardInput            Property System.IO.StreamWriter StandardInput {get;}
StandardOutput           Property System.IO.StreamReader StandardOutput {get;}
StartInfo                Property System.Diagnostics.ProcessStartInfo StartInfo {get;set;}
StartTime                Property System.DateTime StartTime {get;}
SynchronizingObject      Property System.ComponentModel.ISynchronizeInvoke SynchronizingObject {get;set;}
Threads                  Property System.Diagnostics.ProcessThreadCollection Threads {get;}
TotalProcessorTime       Property System.TimeSpan TotalProcessorTime {get;}
UserProcessorTime        Property System.TimeSpan UserProcessorTime {get;}
VirtualMemorySize        Property System.Int32 VirtualMemorySize {get;}
VirtualMemorySize64     Property System.Int64 VirtualMemorySize64 {get;}
WorkingSet               Property System.Int32 WorkingSet {get;}
WorkingSet64             Property System.Int64 WorkingSet64 {get;}

```

```
($a | gm | ? {$_.MemberType -eq "Property"}).count
51
```

We can get at all this process information with the 'ps' alias for defaults fields or we can call customized fields by name or by alias with the 'select-object' cmdlet. For example, the alias CPU below is 'TotalProcessorTime.TotalSeconds':

```
ps | Select Name,ID,VM,WS,CPU | ft -auto
```

```

Name           Id      VM      WS      CPU
----
ApMsgFwd       3320   52310016  5046272  5.9904384
ApntEx         504    69910528  6344704  0.1716011
Apoint         3464   105451520 11116544 13.5252867
audiodg        936    58421248 20250624
BDTUpdateService 1804   70451200  8712192  1.4352092
CCP            3836   210313216 42381312  4.3836281
chrome         1076   146034688 30068736 12.1836781
....

```

The 'sort-object' cmdlet can be applied to the pipeline as well:

```
ps | Select Name,ID,VM,WS,CPU | Sort ID | ft -auto
```

```

Name           Id      VM      WS CPU
----
Idle           0        0      24576
System         4   51748864 43925504
svchost        124   96813056 18530304 10.6860685

```

All information and methods subject to revision and not guaranteed. Use at your own risk.

Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

```
svchost      308 265789440 145620992 273.6569542
svchost      356 546394112  93216768  67.3300316
ApntEx       504  69910528   6344704  0.1716011
...
```

We can also use a 'foreach' language command to help us powerfully invoke the **System.Diagnostics.ProcessModuleCollection** and **System.Diagnostics.ProcessThreadCollection**. Note that the 'foreach' language command is distinct from the Powershell's 'foreach-object' cmdlet. Please see help 'about\_foreach' for more information.

```
foreach ($id in ( Get-Process | ? { $_.Modules } )) {write $id.MainModule}
(or alternatively)
foreach ($id in ( Get-Process | ? { $_.Name } )) {write $id.MainModule}
```

Size(K)	ModuleName	FileName
80	ApMsgFwd.exe	C:\Program Files\Apoint\ApMsgFwd.exe
32	Apntex.exe	C:\Program Files\Apoint\Apntex.exe
176	Apoint.exe	C:\Program Files\Apoint\Apoint.exe
112	BDTUpdateService.exe	C:\Program Files (x86)\Spyware Doctor\BDT\BDTUpdateService.exe

We can find "module information" (as opposed to "process information") about all currently loaded "MainModules" (e.g. binaries) with:

```
$findGP=foreach ($id in ( Get-Process | ? { $_.Modules } )) {write $id.MainModule}
$findGP | ft * | more
```

Size	Company	FileVersion	ProductVersion	Description	Product	ModuleName	FileName	BaseAddress	ModuleMemorySize	EntryPointAddress
80	Alps Electr...	7, 0, 0, 20	7, 0, 0, 20	ApMsgFwd	ApMsgFwd	ApMsgFwd.exe	C:\Program ...	4194304	81920	4205648
32	Alps Electr...	7.0.1.29	7.0.1.29	Alps Pointi...	Alps Pointi...	Apntex.exe	C:\Program ...	4194304	32768	4205088
176	Alps Electr...	7.0.7.156	7.0.7.156	Alps Pointi...	Alps Pointi...	Apoint.exe	C:\Program ...	4194304	180224	4243984
112	Threat Expe...	2, 0, 6, 15	2, 0, 6, 15	Browser Def...	Threat Expe...	BDTUpdateSe...	C:\Program ...	4194304	114688	4228632
40		1.0.0.0	1.0.0.0	CCP	CCP	CCP.exe	C:\Program ...	589824	40960	612350
944	Google Inc.	0.0.0.0	0.0.0.0	Google Chrome	Google Chrome	chrome.exe	C:\Users\Ad...	1900544	966656	2188746

The **get-Process** cmdlet also access **System.Diagnostics.ProcessThreadCollection**. Usually, this is done with 'ps':

```
$modules=ps
$modules | % { $_.Threads}
```

But we can also do this with the 'foreach' language command.

```
foreach ($id in ( Get-Process | ? { $_.Modules } )) {write $id.Threads}
```

```
BasePriority      : 6
CurrentPriority   : 11
Id               : 3324
IdealProcessor   :
PriorityBoostEnabled : True
PriorityLevel     : Normal
```

Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

```
PrivilegedProcessorTime : 00:00:04.6800300
StartAddress             : 2003135024
StartTime                : 7/21/2010 10:21:45 AM
ThreadState              : Wait
TotalProcessorTime       : 00:00:06.3960410
UserProcessorTime        : 00:00:01.7160110
...
```

Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

The syntax below allows the user to take full advantages of Powershell's flexible query, member coupling, and .NET classes reach. Below we are storing to and then displaying from the variable \$FWAPI all ModuleNames (binaries) that import "FirewallAPI.dll":

```
$FWAPI=foreach ($id in ( Get-Process | ? { $_.Modules -match "FirewallAPI.dll" } )) {write $id.MainModule}
$FWAPI | ft *
```

Size	Company	FileVersion	ProductVersion	Description	Product	ModuleName	FileName	BaseAddress	ModuleMemorySize	EntryPointAddress	FileVersionInfo	Site
3024	Microsoft Co...	6.0.6000.163...	6.0.6000.16386	Windows Expl...	Microsoftr W...	Explorer.EXE	C:\Windows\E...	4279566336	3096576	4279720368	File:	...
44	Microsoft Co...	6.0.6000.163...	6.0.6000.16386	Host Process...	Microsoftr W...	svchost.exe	C:\Windows\S...	4291166208	45056	4291175332	File:	...
44	Microsoft Co...	6.0.6000.163...	6.0.6000.16386	Host Process...	Microsoftr W...	svchost.exe	C:\Windows\S...	4291166208	45056	4291175332	File:	...
44	Microsoft Co...	6.0.6000.163...	6.0.6000.16386	Host Process...	Microsoftr W...	svchost.exe	C:\Windows\S...	4291166208	45056	4291175332	File:	...
44	Microsoft Co...	6.0.6000.163...	6.0.6000.16386	Host Process...	Microsoftr W...	svchost.exe	C:\Windows\S...	4291166208	45056	4291175332	File:	...
44	Microsoft Co...	6.0.6000.163...	6.0.6000.16386	Host Process...	Microsoftr W...	svchost.exe	C:\Windows\S...	4291166208	45056	4291175332	File:	...
44	Microsoft Co...	6.0.6000.163...	6.0.6000.16386	Host Process...	Microsoftr W...	svchost.exe	C:\Windows\S...	4291166208	45056	4291175332	File:	...

We can also use the 'Select-object' cmdlet as so:

```
$FWAPI | Select ModuleName,ModuleMemorySize,EntryPointAddress,BaseAddress | ft -auto
```

ModuleName	ModuleMemorySize	EntryPointAddress	BaseAddress
Explorer.EXE	3096576	4279720368	4279566336
svchost.exe	45056	4291175332	4291166208
svchost.exe	45056	4291175332	4291166208
svchost.exe	45056	4291175332	4291166208
svchost.exe	45056	4291175332	4291166208
svchost.exe	45056	4291175332	4291166208
svchost.exe	45056	4291175332	4291166208

It is also possible to use complicated comparative operators for MainModule queries:

```
$findMM=foreach ($id in ( Get-Process | ? { $_.Modules -like "(rsaenh.dll)*" -and $_.Modules -like "(iphlpapi.dll)*" } )) {write $id.MainModule}
$findMM | Select Modulename,FileName,ModuleMemorySize,Size,EntryPointAddress,BaseAddress,Description,Company | ft -auto
```

ModuleName	FileName	ModuleMemorySize	Size	EntryPointAddress	BaseAddress	Description	Company
EvtEng.exe	C:\Program Files\Intel\WiFi\bin\EvtEng.exe	1421312	1388	5369266128	5368709120	Intel(R) PROSet/Wireless Event Log Service	Intel(R) Corporation
explorer.exe	C:\Windows\explorer.exe	3096576	3024	4285946288	4285792256	Windows Explorer	Microsoft Corporation
lsass.exe	C:\Windows\system32\lsass.exe	28672	28	4280556092	4280549376	Local Security Authority Process	Microsoft Corporation
powershell.exe	C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe	479232	468	10464548	10420224	Windows PowerShell	Microsoft Corporation
SearchIndexer.exe	C:\Windows\system32\SearchIndexer.exe	626688	612	4280555876	4280156160	MicroSoft Windows Search Indexer	Microsoft Corporation
svchost.exe	C:\Windows\system32\svchost.exe	45056	44	4290388900	4290379776	Host Process for Windows Services	Microsoft Corporation
svchost.exe	C:\Windows\system32\svchost.exe	45056	44	4290388900	4290379776	Host Process for Windows Services	Microsoft Corporation
svchost.exe	C:\Windows\system32\svchost.exe	45056	44	4290388900	4290379776	Host Process for Windows Services	Microsoft Corporation
svchost.exe	C:\Windows\system32\svchost.exe	45056	44	4290388900	4290379776	Host Process for Windows Services	Microsoft Corporation
svchost.exe	C:\Windows\system32\svchost.exe	45056	44	4290388900	4290379776	Host Process for Windows Services	Microsoft Corporation
VAIOUpdt.exe	C:\Program Files\Sony\VAIO Update 4\VAIOUpdt.exe	1212416	1184	4572064	4194304	VAIO Update	Sony Corporation
WinMail.exe	C:\Program Files\Windows Mail\WinMail.exe	421888	412	4283198448	4283170816	Windows Mail	Microsoft Corporation
WLANExt.exe	C:\Windows\system32\WLANExt.exe	110592	108	4293084340	4293001216	Windows Wireless LAN 802.11 Extensibility Framework	Microsoft Corporation
wmpnetwk.exe	C:\Program Files\Windows Media Player\wmpnetwk.exe	1236992	1208	4292013948	4291035136	Windows Media Player Network Sharing Service	Microsoft Corporation

Now, we query every module (dll) loaded by the process svchost.exe:

```
$SVChost_lm=foreach ($id in ( Get-Process | ? { $_.Name -like "svchost" } )) {write $id.Modules}
$SVChost_lm | ft *
```

All information and methods subject to revision and not guaranteed. Use at your own risk.

Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

Size	Company	FileVersion	ProductVersion	Description	Product	ModuleName	FileName	BaseAddress	ModuleMemorySiz	EntryPointAddre	FileVersionInfo	Site
44	Microsoft Co...	6.0.6000.163...	6.0.6000.16386	Host Process...	Microsoft W...	svchost.exe	C:\Windows\S...	4291166208	45056	4291175332	File:	...
1560	Microsoft Co...	6.0.6001.180...	6.0.6001.18000	NT Layer DLL	Microsoft W...	ntdll.dll	C:\Windows\s...	2002976768	1597440	0	File:	...
1204	Microsoft Co...	6.0.6001.180...	6.0.6001.18000	Windows NT B...	Microsoft W...	kernel32.dll	C:\Windows\s...	2001731584	1232896	2001843376	File:	...
624	Microsoft Co...	7.0.6002.180...	7.0.6002.18005	Windows NT C...	Microsoft W...	msvcrt.dll	C:\Windows\s...	8791785996288	638976	8791786005352	File:	...
1056	Microsoft Co...	6.0.6002.180...	6.0.6002.18005	Advanced Win...	Microsoft W...	ADVAPI32.dll	C:\Windows\s...	8791782391808	1081344	8791782638856	File:	...

But we can do better yet. There are multiple svchost.exe per session. We want to prepend the svchost process ID prior to dumping the modules specific to each:

```
$SVChost_lm=foreach ($sid in ( Get-Process | ? { $_.Name -like "svchost" } )) {write $sid.Name,$sid.ID,$sid.Modules}
$SVChost_lm | ft
```

```
svchost
124
```

Size(K)	ModuleName	FileName
44	svchost.exe	C:\Windows\System32\svchost.exe
1560	ntdll.dll	C:\Windows\system32\ntdll.dll
1204	kernel32.dll	C:\Windows\system32\kernel32.dll
624	msvcrt.dll	C:\Windows\system32\msvcrt.dll
1056	ADVAPI32.dll	C:\Windows\system32\ADVAPI32.dll
1292	RPCRT4.dll	C:\Windows\system32\RPCRT4.dll
1476	wevtsvc.dll	c:\windows\system32\wevtsvc.dll
152	USERENV.dll	c:\windows\system32\USERENV.dll
112	Secur32.dll	c:\windows\system32\Secur32.dll
820	USER32.dll	C:\Windows\system32\USER32.dll
400	GDI32.dll	C:\Windows\system32\GDI32.dll

We can find "module information" about all currently loaded "main modules" (binaries) and their modules (dlls) with:

```
$findAllModules=foreach ($sid in ( Get-Process | ? { $_.Modules } )) {write $sid.MainModule,$sid.Modules}
$findAllModules | ft * | more
```

Size	Company	FileVersion	ProductVersion	Description	Product	ModuleName	FileName	BaseAddress	ModuleMemorySi	EntryPointAddr	FileVersionInfo	Site
80	Alps Electr...	7, 0, 0, 20	7, 0, 0, 20	ApMsgFwd	ApMsgFwd	ApMsgFwd.exe	C:\Program ...	4194304	81920	4205648		
80	Alps Electr...	7, 0, 0, 20	7, 0, 0, 20	ApMsgFwd	ApMsgFwd	ApMsgFwd.exe	C:\Program ...	4194304	81920	4205648		
1560	Microsoft C...	6.0.6001.18...	6.0.6001.18000	NT Layer DLL	Microsoftftr	ntdll.dll	C:\Windows\...	2004090880	1597440	0		
1204	Microsoft C...	6.0.6001.18...	6.0.6001.18000	Windows NT ...	Microsoftftr	kernel32.dll	C:\Windows\...	2001993728	1232896	2002105520		
1056	Microsoft C...	6.0.6002.18...	6.0.6002.18005	Advanced Wi...	Microsoftftr	ADVAPI32.dll	C:\Windows\...	8791772364800	1081344	8791772611848		
1292	Microsoft C...	6.0.6001.18...	6.0.6001.18000	Remote Proc...	Microsoftftr	RPCRT4.dll	C:\Windows\...	8791760961536	1323008	8791761215904		
820	Microsoft C...	6.0.6001.18...	6.0.6001.18000	Multi-User ...	Microsoftftr	USER32.dll	C:\Windows\...	2003238912	839680	2003343380		
400	Microsoft C...	6.0.6002.18...	6.0.6002.18005	GDI Client DLL	Microsoftftr	GDI32.dll	C:\Windows\...	8791787765760	409600	8791787790452		
460	Microsoft C...	6.0.6000.16...	6.0.6000.16386	Shell Light...	Microsoftftr	SHLWAPI.dll	C:\Windows\...	8791765942272	471040	8791766014032		
624	Microsoft C...	7.0.6002.18...	7.0.6002.18005	Windows NT ...	Microsoftftr	msvcrt.dll	C:\Windows\...	8791787110400	638976	8791787119464		

All information and methods subject to revision and not guaranteed. Use at your own risk.

## Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

To compare the listing of every executable and their modules over time we can use "compare-object" (alias "diff"):

```
$a=foreach ($id in ( Get-Process | ? {$_.Modules} )) {write $id.Name,$id.ID,$id.Modules}
sleep -seconds 10
# Start notepad now
$b=foreach ($id in ( Get-Process | ? {$_.Modules} )) {write $id.Name,$id.ID,$id.Modules}
$c=diff $a $b
$c | fl * | out-file modulelog
```

Raw output like the following is produced:

```
InputObject   : notepad
SideIndicator  : =>
```

```
InputObject   : 1712
SideIndicator  : =>
```

```
InputObject   : {System.Diagnostics.ProcessModule (notepad.exe), System.Diagnostics.ProcessModule (ntdll.dll), System.Diagnostics.ProcessModule
(kernel32.dll), System.Diagnostics.ProcessModule (ADVAPI32.dll
)...}
SideIndicator  : =>
```

We can 'unroll' all the 'diff' object with

```
$d=$c | % {$_.InputObject}
$d
notepad
1712
```

Size(K)	ModuleName	FileName
188	notepad.exe	C:\Windows\system32\notepad.exe
1560	ntdll.dll	C:\Windows\system32\ntdll.dll
1204	kernel32.dll	C:\Windows\system32\kernel32.dll
...		

or with selection and/or sorting of module and process specific properties:

```
$d=$c | % {$_.InputObject} | Select ModuleName,FileName,ModuleMemorySize,EntryPointAddress | Sort EntryPointAddress
$d | ft -wrap -auto
```

ModuleName	FileName	ModuleMemorySize	EntryPointAddress
ntdll.dll	C:\Windows\system32\ntdll.dll	1597440	0
kernel32.dll	C:\Windows\system32\kernel32.dll	1232896	2002105520
USER32.dll	C:\Windows\system32\USER32.dll	839680	2003343380
notepad.exe	C:\Windows\system32\notepad.exe	192512	4282241460
smum64.dll	C:\Program Files (x86)\Spyware Doctor\smum64.dll	282624	8791685212184...
...			

We can use the semantics below to create an object ("FindLots") that displays process name, process ID, process filename followed by the modules and threads that belong to that process.

All information and methods subject to revision and not guaranteed. Use at your own risk.

## Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

```
$FindLots=foreach ($sid in ( Get-Process | ? {$_.Name} )) {$sid.Name,$sid.ID,$sid.Path,$sid.Modules,$sid.Threads}  
$FindLots | gm | findstr TypeName
```

```
TypeName: System.String  
TypeName: System.Int32  
TypeName: System.Diagnostics.ProcessModuleCollection  
TypeName: System.Diagnostics.ProcessThreadCollection
```

This type of 'hybrid' object produces output that first lists the process name,PID and full path to the binary followed by all the modules and then all the threads loaded by the respective process:

```
$FindLots | ft
```

```
ApMsgFwd  
3320  
C:\Program Files\Apoin\ApMsgFwd.exe
```

Size(K)	ModuleName	FileName
80	ApMsgFwd.exe	C:\Program Files\Apoin\ApMsgFwd.exe
1560	ntdll.dll	C:\Windows\system32\ntdll.dll
1204	kernel32.dll	C:\Windows\system32\kernel32.dll
1056	ADVAPI32.dll	C:\Windows\system32\ADVAPI32.dll
1292	RPCRT4.dll	C:\Windows\system32\RPCRT4.dll
820	USER32.dll	C:\Windows\system32\USER32.dll
400	GDI32.dll	C:\Windows\system32\GDI32.dll

```
...
```

```
BasePriority      : 6  
CurrentPriority   : 8  
Id               : 3324  
IdealProcessor   :  
PriorityBoostEnabled : True  
PriorityLevel     : Normal  
PrivilegedProcessorTime : 00:00:02.9484189
```

```
...
```

Comparing 'hybrid' (mixed 'Typename') objects produces some problematic results since the original objects have to be reset to process and thread based arrays and then compared.

```
$a=foreach ($sid in ( Get-Process | ? {$_.Name} )) {$sid.Name,$sid.ID,$sid.Path,$sid.Modules,$sid.Threads}  
sleep -seconds 10  
# start mspaint  
$b=foreach ($sid in ( Get-Process | ? {$_.Name} )) {$sid.Name,$sid.ID,$sid.Path,$sid.Modules,$sid.Threads}  
$aa=$a | ? {$_.Item}  
$bb=$b | ? {$_.Item}  
$cc=compare-object $aa $bb
```

The resulting objects lose the linkage between process and module when they become '**ParameterizedProperties**':

```
$bb | gm | findstr Property
```

All information and methods subject to revision and not guaranteed. Use at your own risk.

## Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

```
Item      ParameterizedProperty System.Diagnostics.ProcessModule Item(int index) {get;}
Count     Property              System.Int32 Count {get;}
Item      ParameterizedProperty System.Diagnostics.ProcessThread Item(int index) {get;}
Count     Property              System.Int32 Count {get;}
```

However, we still have list of binaries, modules, and threads that have changed:

**\$cc**

```
InputObject                               SideIndicator
-----
{1808, 3132, 3140, 3240...}                =>
{System.Diagnostics.ProcessModule (mspaint.exe), System.Diagnostics.ProcessModule (ntdll.dll), S... =>
{3676, 3896, 3912, 1264...}                =>
{228, 236, 240, 304...}                   =>
{640, 636, 748, 1008...}                  =>
...
```

**\$cc | % {\$\_.InputObject} | more**

```
BasePriority      : 8
CurrentPriority   : 10
Id                : 1808
IdealProcessor    :
PriorityBoostEnabled : True
PriorityLevel     : Normal
PrivilegedProcessorTime : 00:00:01.1700075
StartAddress      : 1999661616
StartTime         : 7/30/2010 10:55:03 AM
ThreadState       : Wait
...
```

```
ModuleName       : mspaint.exe
FileName         : C:\Windows\system32\mspaint.exe
BaseAddress      : 4285464576
ModuleMemorySize : 614400
EntryPointAddress : 4285797576
...
```

**\$cc | % {\$\_.InputObject} | findstr ModuleName**

```
ModuleName       : mspaint.exe
ModuleName       : ntdll.dll
ModuleName       : kernel32.dll
ModuleName       : ADVAPI32.dll
ModuleName       : RPCRT4.dll
...
```

**\$cc | % {\$\_.InputObject} | Sort Id | findstr /V "IdealProcessor" | findstr Id**

```
Id               : 228
Id               : 236
Id               : 240
Id               : 304
...
```

All information and methods subject to revision and not guaranteed. Use at your own risk.

## GWMI (Windows Management Instrumentation)

A comparison between alias 'ps' ("get-process") and gwmi ("get-WMIObject") shows that gwmi uses the PID of each process as a handle to display much the same process information as ps. WMI is a comprehensive database of Windows objects of all types. A full discussion is beyond the scope of this document. However, WMI's database and management format does make time slice comparisons straightforward.

```
$i=ps
$j=gwmi win32_process
diff $i $j
```

```
InputObject                                                    SideIndicator
-----
\\RMFVISTA\root\cimv2:Win32_Process.Handle="0"                =>
\\RMFVISTA\root\cimv2:Win32_Process.Handle="4"                =>
\\RMFVISTA\root\cimv2:Win32_Process.Handle="524"              =>
\\RMFVISTA\root\cimv2:Win32_Process.Handle="656"              =>
...
System.Diagnostics.Process (ApmMsgFwd)                       <=
System.Diagnostics.Process (ApntEx)                           <=
System.Diagnostics.Process (Apoint)                           <=
System.Diagnostics.Process (audiodg)                          <=
System.Diagnostics.Process (BDTUpdateService)                 <=
...
```

Note how the 'Handle' field is the same as the process ID in the gwmi query below:

```
$j=gwmi win32_process
$j | Select ProcessId, Name, Handle, HandleCount | Sort ProcessId, HandleCount | ft -auto
```

```
ProcessId Name                Handle HandleCount
-----
0 System Idle Process         0      0
4 System                      4      1729
176 Tcpview.exe               176    308
432 svchost.exe               432    1439
512 smss.exe                  512    28
632 wmpnscfg.exe              632    133
...
```

Comparing process command lines is straightforward with WMI:

```
$a=foreach ($id in ( gwmi win32_process | ? { $_.HANDLE } )) {write $id.CommandLine}
sleep -seconds 10
# Start notepad now
$b=foreach ($id in ( gwmi win32_process | ? { $_.HANDLE } )) {write $id.CommandLine}
$c=diff $a $b
$c
```

```
InputObject                                                    SideIndicator
-----
```

Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

```
"C:\Windows\system32\notepad.exe" =>
$c | % {$_.InputObject}
"C:\Windows\system32\notepad.exe"
GWMI also has a win32_thread component with a feature not so easily access in System.Diagnostic.Process : a relative path link from each thread to process handle or 'PID'. This makes linking threads with their processes more straightforward.
```

```
$j=gwmi win32_thread
$j | Select ProcessHandle, Handle, StartAddress | Sort ProcessHandle, Handle | ft -auto
```

```
ProcessHandle Handle StartAddress
-----
0 0 54986864
0 1 54986864
1004 1392 1997105712
1004 1688 1997105712
1004 2160 1997105712
...
```

```
$j | % {$_.RELSPATH}
```

```
Win32_Thread.Handle="0",ProcessHandle="0"
Win32_Thread.Handle="1",ProcessHandle="0"
Win32_Thread.Handle="8",ProcessHandle="4"
Win32_Thread.Handle="16",ProcessHandle="4"
Win32_Thread.Handle="20",ProcessHandle="4"
Win32_Thread.Handle="24",ProcessHandle="4"
...
```

Because WMI's **win32\_thread** class links process handles (PID) explicitly with thread handles, we can definitively see which processes have changed or created new thread handles in the diff below.

```
$a=foreach ($id in ( gwmi win32_thread | ? {$_.HANDLE} )) {write $id.ProcessHandle}
sleep -seconds 10
# Start notepad now
$b=foreach ($id in ( gwmi win32_thread | ? {$_.HANDLE} )) {write $id.ProcessHandle}
$c=diff $a $b
$c | % {ps -id $_.inputobject} | Select ProcessName,ID | ft -auto
```

```
ProcessName Id
-----
services 748
svchost 664
svchost 700
svchost 1288
explorer 3088
SearchIndexer 3676
notepad 4116
```