National Cyber Security Centre
a part of GCHQ

Infinite Second

Malware Analysis Report

Version 1.0

03 September 2021

National Cyber Security Centre
a part of GCHQ

Infinite Second

PowerShell dropper used to deploy ComRAT

Executive summary

- PowerShell script that deploys ComRAT malware
- Uses multiple layers of obfuscation, including XOR and 3DES
- Modifies the scheduled task of the Microsoft Consolidator, a part of Customer Experience Improvement Program, to achieve persistence on user log-in

Introduction

Infinite Second is a malicious PowerShell script that is used to drop the ComRAT malware (also referred to as Chinch). ComRAT is well documented in open-source reporting and is beyond the scope of this report.

This script uses the registry to store configuration and necessary data, including an obfuscated version of ComRAT. It bootstraps by modifying the scheduled task of the Microsoft Customer Experience Improvement Program known as the Microsoft Consolidator. Some functionality requires Administrator permissions and execution policies that allow any PowerShell script to be run (e.g. ‘Unrestricted’), and which depend on PowerShell version 2 being installed. This report does not cover how Infinite Second is deployed to a target or how it interacts with referenced files that are not traceable to this malware’s installation. These files may be dropped by a parent component.

National Cyber Security Centre
a part of GCHQ

Malware details

Metadata

Filename	N/A
Description	Infinite Second PowerShell script
Size	2172814 bytes
MD5	aaedebe3574f07e81650437739ac9357
SHA-1	2030075a29da9c6c1c44aeb4862588480a64509c
SHA-256	d532610528ad75d80c17f0372c5af8d2e417e0e2e08a055bf9923b71c149c51c

Filename	N/A
Description	Script stored in registry (Base64 encoded)
Size	3090282 bytes
MD5	187b78e6ac908b43556b3f97587a5071
SHA-1	d67af71a2c5d717b5e299653a5c1b2d097919b43
SHA-256	9a3c57b080cb7b55d56b263a0bb2de592681e78716d025b3cdc895ee2dc144a1

Filename	N/A
Description	Script stored in registry (decoded)
Size	1158853 bytes
MD5	486bd55b1e8b28d16367ce342bc6b733
SHA-1	f63ae2ca45badde26b8950628a0dd3f3a307bcf1
SHA-256	27fb6e28589ddff239d2616fde8169204da3f8365b9670e68f733424ecab93fe

Filename	N/A
Description	Modified version of the PowerSploit Invoke-ReflectivePEInjection PowerShell script
Size	106601 bytes
MD5	c310c43be2b837cd18d33356936254b0
SHA-1	cf978c1dfe192485733223a6aa9b66797f7c1f08
SHA-256	84a8d297f2ca73164df69084a216f479fd2ce1052f465deb0300ba1f69614e8b

Filename	N/A
Description	ComRAT payload dropped by Infinite Second. Windows DLL (PE) x86
Size	1544192 bytes
MD5	1d626b48ae7062bd319cb768a8ca979d
SHA-1	d117643019d665a29ce8a7b812268fb8d3e5aadb
SHA-256	b93484683014aca8e909c9b5648d8f0ac21a45d0c193f6ca40f0b01d2464c1c4
Compile time	2018-09-27 13:13:20

Filename	N/A
Description	ComRAT payload dropped by Infinite Second. Windows DLL (PE) x64
Size	1825792 bytes
MD5	82fc41b3e9ce722d4e6b21580e873d52
SHA-1	8a93f216607f43bd39164096728ae2bbcdbc2de8
SHA-256	e209d07f8992892bf4776c897a8f054849b12464d9f58bcb78602217b3fd98bc
Compile time	2018-09-27 13:14:19

National Cyber Security Centre
a part of GCHQ

MITRE ATT&CK®

This report has been compiled with respect to the MITRE ATT&CK® framework, a globally accessible knowledge base of adversary tactics and techniques based on real-world observations.

Tactic	ID	Technique	Procedure
Execution	T1059.001	Command and Scripting Interpreter: PowerShell	Infinite Second is a PowerShell script that deploys and set ups up persistence for ComRAT.
Persistence	T1053.005	Scheduled Task/Job: Scheduled Task	Infinite Second creates or modifies a scheduled task to run as SYSTEM to launch ComRAT.
Defense Evasion	T1112	Modify Registry	Infinite Second stores cryptographic keys, additional scripts and obfuscated payloads in the registry.
	T1140	Deobfuscate/Decode Files or Information	Infinite Second deobfuscates information stored in the registry using 3DES, XOR, and Base64 encoding.
	T1036.005	Masquerading: Match Legitimate Name or Location	Infinite Second uses registry values that masquerade as legitimate software.
	T1070.004	Indicator Removal on Host: File Deletion	Infinite Second deletes `C:\Windows\Temp\tmp4071.tmp` and `C:\Users\Public\Documents\thumbs.ini`

Functionality

Overview

Infinite Second drops and sets up persistence for a variant of ComRAT. It hijacks a legitimate Windows scheduled task to bootstrap execution of a malicious PowerShell script upon user log-in and every 6 hours, appending this script to the existing command line to ensure that the legitimate scheduled task is not affected. The PowerShell script extracts and executes encoded data from the registry, which then decodes and injects ComRAT into the `explorer` process using a modified version of the PowerSploit[1] `Invoke-ReflectivePEInjection` script.

The overall process is shown in Figure 1 below:

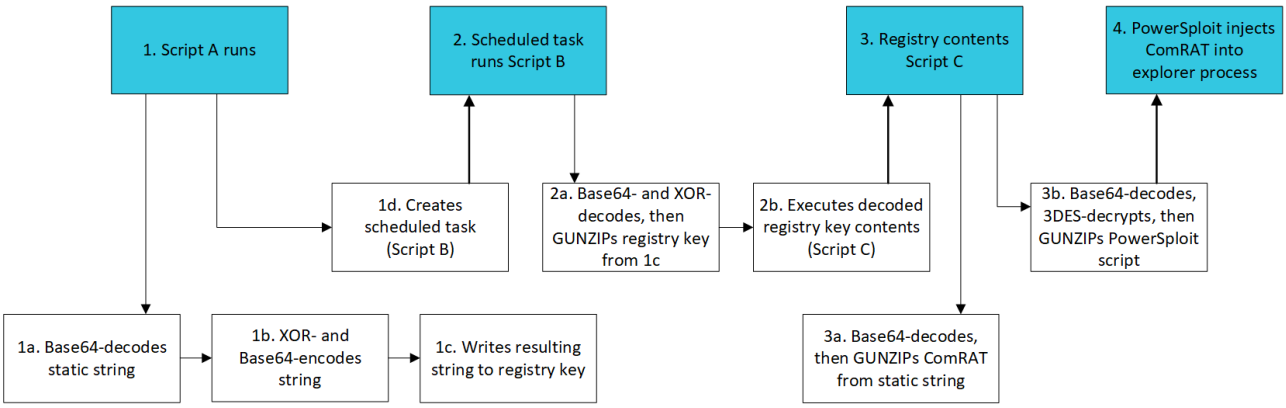

Figure 1: High-level logical flow

[1] https://github.com/PowerShellMafia/PowerSploit/blob/master/CodeExecution/Invoke-ReflectivePEInjection.ps1

Installation

The outermost PowerShell script (referred to as 'Script A' in Figure 1) is responsible for installing Infinite Second by storing configuration data to the registry and installing persistence.

The main Infinite Second PowerShell script ('Script C') is contained in this installer as an obfuscated payload. Two versions – containing ComRAT binaries compiled for 32- and 64-bit architectures – are present in the installer, and the appropriate one for the target system is selected based on the size of `System.IntPtr` during installation.

During installation this payload is XOR-encoded using a randomly generated key consisting of between 6 and 9 ASCII characters corresponding to the following regular expression:

```
/[A-Z]{2,3}[0-9]{2,3}[a-z]{2,3}/
```

This encoded payload is then Base64-encoded and saved to the registry in the `WSqmCons` value under `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SQMClient\Windows`. This value is created by Infinite Second, however the name appears to have been selected to appear legitimate based on its similarity to the Consolidator executable `wsqmcons.exe`.

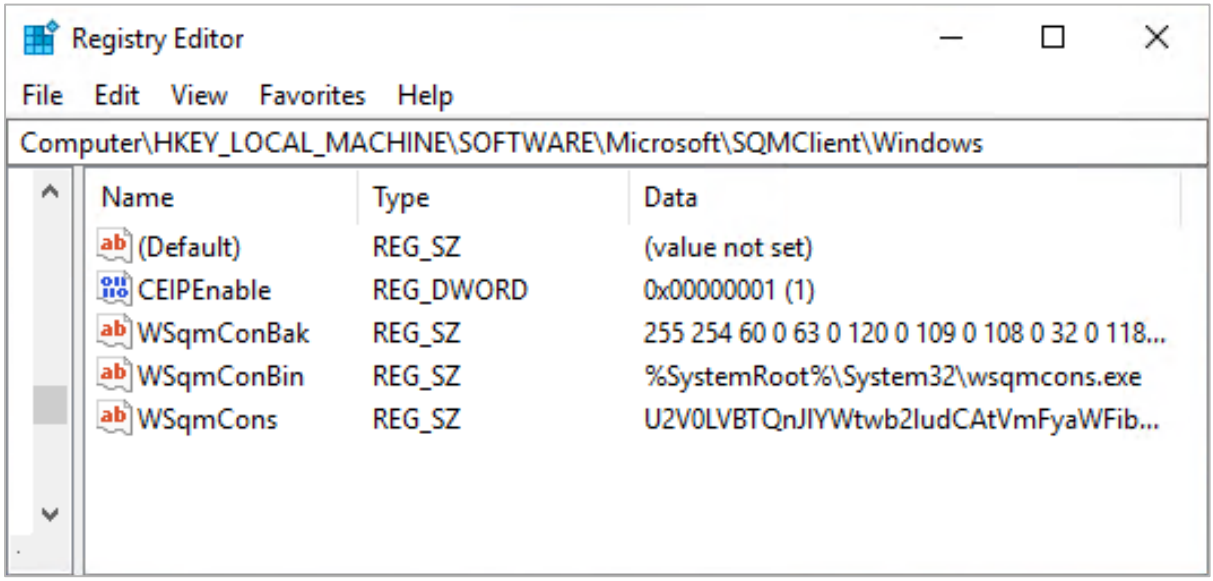

Figure 2: Registry values created during installation

Encoding the payload in this way ensures that the stored payload will be different for each Infinite Second installation.

The contents of this registry value are decoded and executed by a subsequent PowerShell script ('Script B'), run by a modified Microsoft Consolidator scheduled task as described in this report under 'Functionality (Persistence)'.

Other registry values are also populated during installation with the encryption password and salt for the embedded PowerSploit functionality – these are described in this report under 'Functionality (Execution)'.

Persistence

Infinite Second uses the Microsoft Consolidator scheduled task to bootstrap itself upon user log-in. The Microsoft Consolidator is part of Microsoft's Customer Experience Improvement Program and is enabled by default on Windows 10.

The default scheduled task is stored in XML format in `%WinDir%\System32\Tasks\Microsoft\Windows\Customer Experience Improvement Program\Consolidator`, and is configured to execute the following command every 6 hours:

```
%SystemRoot%\System32\wsqmcons.exe <arguments>
```

Infinite Second modifies the Microsoft Consolidator scheduled task, ensuring that the log-in trigger is enabled and changing the configured command to also execute a malicious PowerShell script (referred to as 'Script B' in Figure 1):

```
cmd /c "%SystemRoot%\System32\wsqmcons.exe <arguments> & powershell.exe -v 2 <malicious script>"
```

This script reads, Base64-decodes, XOR-decodes, GUNZIPs and executes the contents of the `WSqmCons` registry value (referred to as 'Script C' in Figure 1), as described in this report under 'Functionality (Script deobfuscation)'.

Prior to installing persistence, the following registry values are created in the same location alongside `WSqmCons`:

- `WSqmConBak` – the scheduled task configuration, used to re-create the scheduled task if it does not exist. If neither the scheduled task nor this value exist, Infinite Second will exit.
- `WSqmConBin` – the Microsoft Consolidator executable path.
- `WSqmConHex` – (optional) any configured arguments to the Microsoft Consolidator.

If the Consolidator scheduled task is not present, the Infinite Second installer script will attempt to use these values to recreate it if they already exist. Since they are created as part of the Infinite Second installation process this suggests that their purpose is to enable reinstallation in the event that the scheduled task is removed.

Script deobfuscation

The main Infinite Second PowerShell script ('Script C') is stored as an encoded payload in the registry, as described in this report under 'Functionality (Installation)'. After being bootstrapped by the Consolidator scheduled task, it is deobfuscated and executed by 'Script B'.

Several layers of different encodings are used, including:

- Base64-decode
- XOR
- GZIP decompress

The XOR encoding uses a 6- to 9-byte ASCII key that is generated during installation, as described in this report under 'Functionality (Installation)'.

This script contains two further obfuscated components, which are described in the following section.

Payload deobfuscation

The main Infinite Second script ('Script C') extracts and deobfuscates two components: a ComRAT payload, and a modified version of the PowerSploit `Invoke-ReflectivePEInjection` script, which is used to inject ComRAT into the `explorer` process, as described in this report under 'Functionality (Execution)'.

ComRAT

The main Infinite Second script ('Script C') contains a ComRAT DLL that corresponds to the system's architecture (32- or 64-bit). This embedded payload is deobfuscated with the following operations:

- Base64-decoded
- GUNZIP

PowerSploit

As well as the Base64-encoding and GZIP compression used for ComRAT, the modified PowerSploit content is also 3DES encrypted.

This 3DES encryption uses a hardcoded Initialisation Vector of `FVADRCORAOSKBHPX`. The encryption key is derived using `PasswordDeriveBytes` with a hardcoded hash name of '`SHA1`' and using the password and salt stored in the `N` and `S` registry values in the following registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publish
ers\{cabe18a5-69b9-4eec-bed0-fa080ed05a3b}\ChannelReferences\0
```

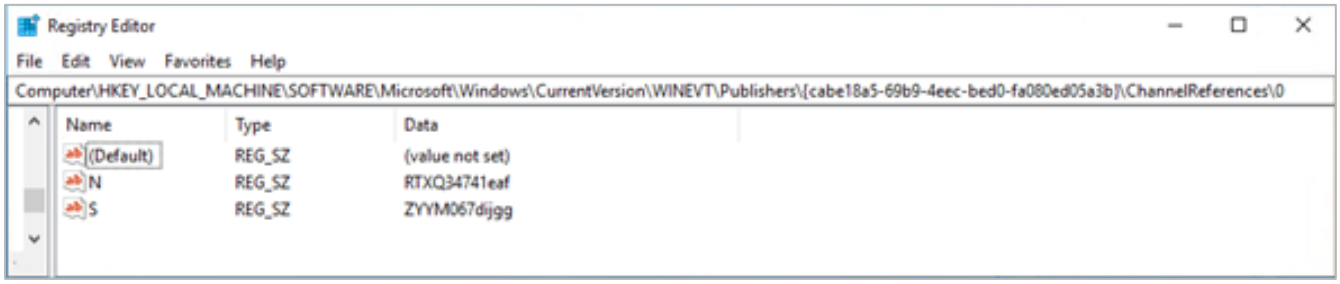

Figure 3: Registry values used for 3DES password and salt

The order of operations to deobfuscate the PowerSploit script is:

- Base64-decode
- 3DES-decrypt
- GUNZIP

Execution

Following the deobfuscation of the ComRAT and PowerSploit components, as described in the report under 'Functionality (Payload deobfuscation)', Infinite Second attempts to execute a file (`C:\Users\Public\Documents\thumbs.ini`) that is previously unreferenced. This file is executed in a way that suggests it is also a PowerShell script, and may be placed on target by another component or as the result of a later task. Its purpose is unclear.

Finally, the ComRAT DLL is executed in-memory by using a modified version of the PowerSploit `Invoke-ReflectivePEInjection` script to inject it into the `explorer` process. There is no immediate indication why explorer has been selected but a possible reason is so that ComRAT's C2 traffic is not blocked by the Windows Firewall.

Defence evasion

Infinite Second is obfuscated using random variable names, white-space has been removed and strings are encoded using methods such as the following example:

```
$PS061hh = New-Object System.Text.ASCIIEncoding;
$$PS061hh.GetString(Arguments)
```

As previously outlined, Infinite Second is encoded in multiple layers (Base64, XOR, GZIP, 3DES), with the payloads stored in the registry, to evade standard file-based scanning, as described in this report under 'Functionality (Execution)'.

The Infinite Second installer script deletes `C:\Windows\Temp\tmp4071.tmp`, however this file is not created by Infinite Second and therefore its purpose is not clear. This is likely an artefact of a parent component. Similarly, Infinite Second executes and then deletes `C:\Users\Public\Documents\thumbs.ini`.

Conclusion

Infinite Second appears designed to be deployed to multiple systems, given the randomly generated variable names and the way it dynamically modifies the content stored in the registry keys. It installs persistence for, and executes, an embedded ComRAT payload. It makes heavy use of obfuscation and employs methods to avoid detection, including the use of seemingly legitimate registry keys and hijacking a legitimate scheduled task to achieve persistence.

Detection

Indicators of compromise

Type	Description	Values
Registry value name	Used to store the obfuscated PowerShell script	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SQMClient\Windows\WSqmCons
Registry value name	Used to store hardcoded 3DES password for PowerSploit script	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers\{cabe18a5-69b9-4eec-bed0-fa080ed05a3b}\ChannelReferences\0\N
Registry value name	Used to store hardcoded 3DES salt for PowerSploit script	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers\{cabe18a5-69b9-4eec-bed0-fa080ed05a3b}\ChannelReferences\0\S
Registry value name	Backup of the previous Microsoft Consolidator executable e.g %SystemRoot%\System32\wsqmcons.exe	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SQMClient\Windows\WSqmConBin HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers\{cabe18a5-69b9-4eec-bed0-fa080ed05a3b}\ChannelReferences\0
Registry value name	Backup of Microsoft Consolidator arguments (not present if no arguments)	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SQMClient\Windows\WSqmConHex
Registry value name	Backup of Microsoft Consolidator scheduled task configuration	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SQMClient\Windows\WSqmConBak
Path	This file is deleted by Infinite Second	C:\Windows\Temp\tmp4071.tmp
Path	This file is executed and then deleted by Infinite Second	C:\Users\Public\Documents\thumbs.ini

National Cyber Security Centre
a part of GCHQ

Rules and signatures

Description	Identifies the XOR implementation
Precision	Low false-positive rate based on VirusTotal retro-hunting. Specific to observed implementation
Rule type	YARA

```
rule infinitesecond_custom_xor
{
    meta:
        author = "NCSC"
        description = "Identifies the XOR implementation"


    strings:
      $a =
"KGJ5dGVbXSBpbmNvbWVfYnl0ZXMsIGJ5dGVbXSBnYW1tYSl7Ynl0ZVtdIG91dHB1dCA9IG5l
dyBieXRlW2luY29tZV9ieXRlcy5MZW5ndGhdO2ZvciAoaW50IGkgPSAwOyBpIDwgaW5jb21lX
2J5dGVzLkxlbmd0aDsgKytpKXtvdXRwdXRbaV0gPSAoYnl0ZSkoaW5jb21lX2J5dGVzW2ldIF
4gZ2FtbWFbaSAlIGdhbW1hLkxlbmd0aF0pO31yZXR1cm4gb3V0cHV0O319"
      $b =
"hieXRlW10gaW5jb21lX2J5dGVzLCBieXRlW10gZ2FtbWEpe2J5dGVbXSBvdXRwdXQgPSBuZX
cgYnl0ZVtpbmNvbWVfYnl0ZXMuTGVuZ3RoXTtmb3IgKGludCBpID0gMDsgaSA8IGluY29tZV9
ieXRlcy5MZW5ndGg7ICsraSl7b3V0cHV0W2ldID0gKGJ5dGUpKGluY29tZV9ieXRlc1tpXSBe
IGdhbW1hW2kgJSBnYW1tYS5MZW5ndGhdKTt9cmV0dXJuIG91dHB1dDt9f"
      $c =
"oYnl0ZVtdIGluY29tZV9ieXRlcywgYnl0ZVtdIGdhbW1hKXtieXRlW10gb3V0cHV0ID0gbmV
3IGJ5dGVbaW5jb21lX2J5dGVzLkxlbmd0aF07Zm9yIChpbnQgaSA9IDA7IGkgPCBpbmNvbWVf
Ynl0ZXMuTGVuZ3RoOyArK2kpe291dHB1dFtpXSA9IChieXRlKShpbmNvbWVfYnl0ZXNbaV0gX
iBnYW1tYVtpICUgZ2FtbWEuTGVuZ3RoXSk7fXJldHVybiBvdXRwdXQ7fX"
    condition:
        any of them
}
```

Description	Identifies the random name generator
Precision	Low false-positive rate based on VirusTotal retro-hunting. Specific to observed implementation
Rule type	YARA

```
rule infinitesecond_random_name_function
{
    meta:
        author = "NCSC"
        description = "Identifies the random name generator"


    strings:
        $ = "function TVM730egf([string[]]$GP50afa) { $UC33gfa =
((1..(Get-Random -Min 2 -Max 4) | % {[Char](Get-Random -Min 0x41 -Max
0x5B)}) -join '');"
        $ = "$EQ33abh = ((1..(Get-Random -Min 2 -Max 4) | % {[Char](Get-
Random -Min 0x30 -Max 0x3A)}) -join '');"
        $ = "$OFK689fa = ((1..(Get-Random -Min 2 -Max 4) | % {[Char](Get-
Random -Min 0x61 -Max 0x6B)}) -join '');"
    condition: any of them
}
```

Description	Identifies hardcoded strings throughout the script
Precision	Low false-positive rate based on VirusTotal retro-hunting. Specific to observed implementation
Rule type	YARA

```
rule infinitesecond_hardcoded_strings
{
    meta:
        author = "NCSC"
        description = "Identifies hardcoded strings throughout the
script"


    strings:
        $ = "HKLM:\\SOFTWARE\\Microsoft\\SQMClient\\Windows"
        $ =
"\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\WINEVT\\Publishers\\{cab
e18a5-69b9-4eec-bed0-fa080ed05a3b}\\ChannelReferences\\0"
        $ = "C:\\Windows\\Temp\\tmp4071.tmp"
        $ = "C:\\Windows\\System32\\Tasks\\Microsoft\\Windows\\Customer
Experience Improvement Program\\Consolidator"
        $ = "Microsoft\\Windows\\Customer Experience Improvement Program"
        $ = "WSqmConHex"
        $ = "WSqmConBin"
        $ = "WSqmCons"
        $ = "WSqmConBak"


    condition: 4 of them
}
```

Description	3DES Initialisation Vector
Precision	This signature is believed to have a low false-positive rate but is likely to be limited by small variations in the code
Rule type	YARA

```
rule infinitesecond_3des_file
{
    meta:
        author = "NCSC"
        description = "3DES Initialisation Vector"


    strings:
            $IV = "FVADRCORAOSKBHPX"
    condition:
        all of them
}
```

Description	Identification of unusual PowerShell break point
Precision	This signature is believed to have a low false-positive rate but is likely to be limited by small variations in the code
Rule type	YARA

```
rule infinitesecond_psbreakpoint_luis_armstrong
{
    meta:
        author = "NCSC"
        description = "Identification of unusual PowerShell break point"


    strings:
        $  = "Set-PSBreakpoint -Variable luis_armstrong -Mode Write;"
wide ascii
    condition:
        all of them
}
```

National Cyber Security Centre
a part of GCHQ

Description	Identifies the custom variation in the handling of duplicate process names in the reflective injection script.
Precision	This signature is believed to have a low false-positive rate but is likely to be limited by small variations in the code
Rule type	YARA

```
rule infinitesecond_powershell_reflective_injection
{
    meta:
        author = "NCSC"
        description = "Identifies a modification made to reflective
injection script"


    strings:
            $process_number = {7b 54 68 72 6f 77 20 22 43 61 6e 27 74 20
66 69 6e 64 20 70 72 6f 63 65 73 73 20 24 50 72 6f 63 4e 61 6d 65 22 7d
65 6c 73 65 7b 24 50 72 6f 63 49 64 20 3d 20 24 50 72 6f 63 65 73 73 65
73 5b 30 5d 2e 49 44 7d 7d} // {Throw "Can't find process
$ProcName"}else{$ProcId = $Processes[0].ID}}
      condition:
            all of them
}
```