Mo’ Shells Mo’ Problems – Network Detection

March 28, 2014 Danny Lungstrom and Andy Schworer

From The Front Lines, Research & Threat Intel

Disclaimer: CrowdStrike derived this information from investigations in non-classified environments. Since we value our client’s privacy and interests, some data has been redacted or sanitized.

In previous posts of this “Mo’ Shells Mo’ Problems” blog series we discussed web shells with specific Deep Panda adversary examples as well as how to detect them within your enterprise using file stacking and web log analysis. This blog entry completes the series with related methodology for detecting web shells using network packet capture data.

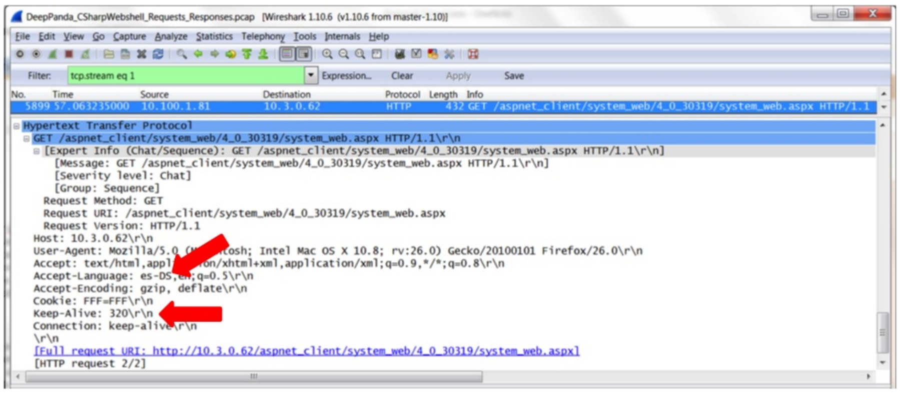
Network detection of web shells can be a particularly important line of defense; especially if stacking and web log analysis are not feasible in your environment. Host-based indicators of the initial compromise may be hard to come by if the adversary already has a long-standing presence in an environment where the logs were either erased or rolled over due to time. This is particularly true with web shells as they may be used very sporadically once installed and are often a fallback option in case another, noisier, backdoor is discovered. During their inactivity, web shells do not generate any traffic on the network, as you would normally find with a Remote Access Tool (RAT) beaconing out to a command and control server on a regular interval. It's important that you are able to detect web shells when given the chance, as they can be as dangerous to your environment as any RAT. In the rest of this blog post we walk through a variety of ways to detect web shells via the network.

Network Signatures

Signature-based intrusion detection systems (IDSs), like Snort, can be very powerful for identifying known web shells. Such solutions are entirely dependent, however, on the signatures they use (much like anti-virus products are only as good as their virus definitions). If a web shell is known, analyzed, and has signatures written and implemented for it, the false positive rate and amount of content required for manual review can be very low. Companies such as CrowdStrike spend significant time and effort creating such signatures based on malicious traffic they find in the wild. Subscribing to such a set of Snort signatures can increase a security team's visibility for many/most known past and current threats. The usual issue applies, however, where future web shell characteristics cannot be predicted, nor can an effective generic web shell signature be written that would do any justice or raise detection confidence. If there were, you wouldn't be seeing this blog series with multiple methodologies used for web shell detection, as we wouldn't need any others and adversaries would quickly shift to a different family of malware.

As discussed in Part 1 of this blog series, Mo' Shells Mo' Problems – Deep Panda Web Shells, the Deep Panda web shell exhibits unique characteristics in the HTTP header in addition to specific HTML content that make using custom Snort IDS signatures possible. Below, we've provided a few such signatures that detect the Deep Panda web shell. By showing the original packet captures, we hope to demonstrate how you might develop your own signatures after encountering unknown malware.

The following packet capture is a GET request for the Deep Panda web shell, system_web.aspx. You'll notice the uniquely identifying “Keep-Alive” value of “320” along with the “Accept-Language” value of “es-DS”.


DeepPanda_CSharpWebshell_Requests_Responses.pcap [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp.stream eq 1
Expression... Clear Apply Save

```
No.   Time             Source        Destination   Protocol Length Info
5899 57.063235000  10.100.1.81   10.3.0.62     HTTP     432 GET /aspnet_client/system_web/4_0_30319/system_web.aspx HTTP/1.1

Hypertext Transfer Protocol
  GET /aspnet_client/system_web/4_0_30319/system_web.aspx HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /aspnet_client/system_web/4_0_30319/system_web.aspx HTTP/1.1\r\n]
      [Message: GET /aspnet_client/system_web/4_0_30319/system_web.aspx HTTP/1.1\r\n]
      [Severity level: Chat]
      [Group: Sequence]
    Request Method: GET
    Request URI: /aspnet_client/system_web/4_0_30319/system_web.aspx
    Request Version: HTTP/1.1
  Host: 10.3.0.62\r\n
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:26.0) Gecko/20100101 Firefox/26.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: es-DS,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Cookie: FFF=FFF\r\n
  Keep-Alive: 320\r\n
  Connection: keep-alive\r\n
  \r\n
  [Full request URI: http://10.3.0.62/aspnet_client/system_web/4_0_30319/system_web.aspx]
  [HTTP request 2/2]
```

The following Snort signature takes advantage of these unique characteristics and will trigger an alert if encountered.

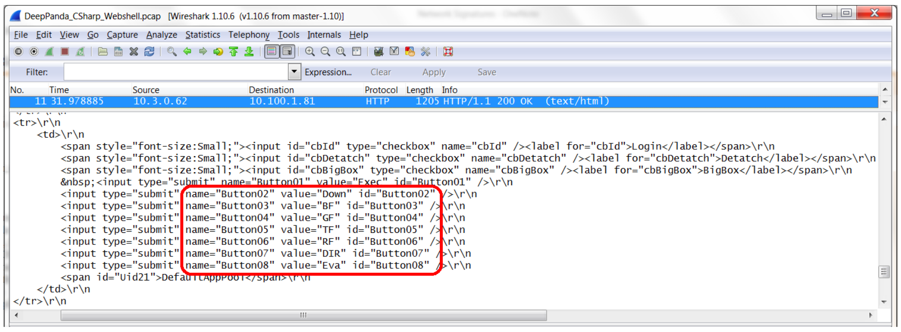
```
alert tcp $EXTERNAL_NET any -> $WEB_SERVERS $HTTP_PORTS (msg:
"CrowdStrike Deep Panda CSharp Webshell Headers"; content:
"Keep-Alive: 320"; http_raw_header; content: "es-DN";
http_raw_header; flow: established, to_server; classtype:
trojan-activity; metadata: service http; sid: xxx; rev: xxx; )
```

This next packet capture is another GET request for the Deep Panda web shell, system_web.aspx, but this time using the cookie value referenced in Part 1 of this blog series. This particular cookie content appears unique to this web shell and is used in the following Snort signature, where a 1 or 2 digit number is assigned to the cookie value for “zWiz”.

The corresponding Snort signature is as follows.

```
alert tcp $EXTERNAL_NET any <> $WEB_SERVERS $HTTP_PORTS (msg:
"CrowdStrike Deep Panda CSharp Webshell Cookie"; content:
"zWiz="; http_raw_cookie; pcre: "/zWiz=[0-9]{1,2}/K"; flow:
established; classtype: trojan-activity; metadata: service
http; sid: xxx; rev: xxx;)
```

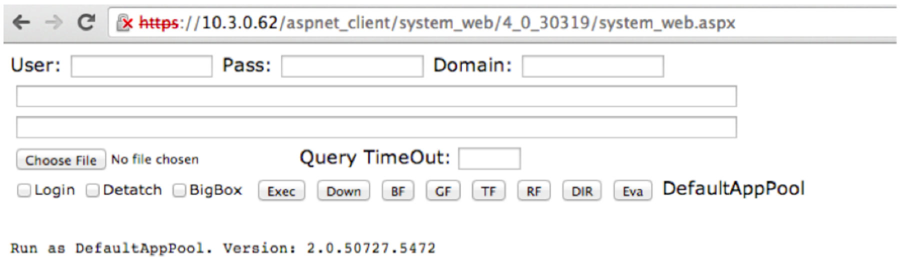
Beyond the HTTP header information identified above, the actual HTML content can be very useful in identifying the web shell via the network. In the case of the Deep Panda web shell, a specific set of fields and buttons are presented to the end user, and are labeled accordingly in the captured HTML. The following packet capture displays a snippet of this HTML that produces the web shell interface.


DeepPanda_CSharp_Webshell.pcap [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save

```
No.   Time        Source      Destination   Protocol Length Info
   11 31.978885   10.3.0.62   10.100.1.81   HTTP       1205 HTTP/1.1 200 OK  (text/html)
```

```
<tr>\r\n
    <td>\r\n
        <span style="font-size:Small;"><input id="cbId" type="checkbox" name="cbId" /><label for="cbId">Login</label></span>\r\n
        <span style="font-size:Small;"><input id="cbDetatch" type="checkbox" name="cbDetatch" /><label for="cbDetatch">Detatch</label></span>\r\n
        <span style="font-size:Small;"><input id="cbBigBox" type="checkbox" name="cbBigBox" /><label for="cbBigBox">BigBox</label></span>\r\n
        &nbsp;<input type="submit" name="Button01" value="Exec" id="Button01" />\r\n
        <input type="submit" name="Button02" value="Down" id="Button02" />\r\n
        <input type="submit" name="Button03" value="BF" id="Button03" />\r\n
        <input type="submit" name="Button04" value="GF" id="Button04" />\r\n
        <input type="submit" name="Button05" value="TF" id="Button05" />\r\n
        <input type="submit" name="Button06" value="RF" id="Button06" />\r\n
        <input type="submit" name="Button07" value="DIR" id="Button07" />\r\n
        <input type="submit" name="Button08" value="Eva" id="Button08" />\r\n
        <span id="Uid21">DefaultAppPool</span>\r\n
    </td>\r\n
</tr>\r\n
```

This HTML produces the following web shell.



```
Run as DefaultAppPool. Version: 2.0.50727.5472
```

The corresponding Snort signature based on this design is as follows.

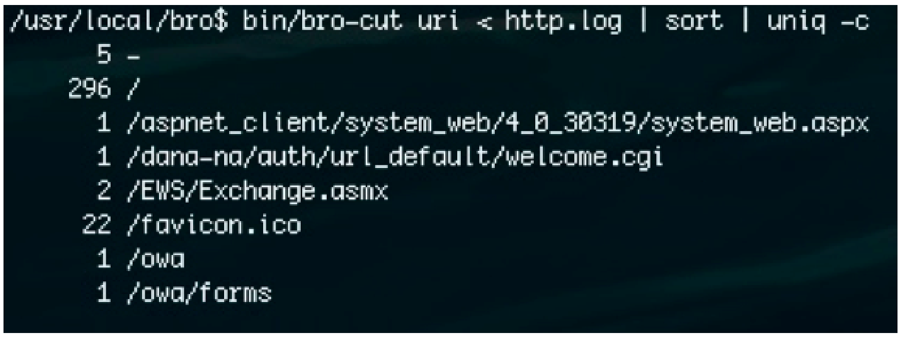
```
alert tcp $WEB_SERVERS $HTTP_PORTS -> $EXTERNAL_NET any (msg:
"CrowdStrike Deep Panda CSharp Webshell Field Response"; content:
"value=\"Down\" id=\"Button02\""; content: "value=\"BF\"
id=\"Button03\""; within: 100; content: "value=\"GF\"
id=\"Button04\""; within: 100; content: "value=\"TF\"
id=\"Button05\""; within: 100; flow: established, from_server;
classtype: trojan-activity; metadata: service http; sid: xxx; rev:
xxx;)
```

It is still possible to leverage or create more general web shell IDS signatures for your environment that go beyond specific web shells. One example would be a Snort signature that looks for a long Base64-encoded POST request (e.g. 30 or more continuous valid Base64 characters without a space). Such signatures will come with many more false positives and will likely require tailoring to your environment. The better you know your network and the traffic you expect, the better you'll be able to tailor such signatures and reduce false positives.

HTTP Request Analysis

Hunting through network packet capture data can be tough if you don't know exactly what you're looking for, but it is possible to identify a web shell using techniques similar to those discussed in Part 3 of this blog series, Mo' Shells Mo' Problems – Web Server Log Analysis. In that post we used statistical analysis on web server logs to identify anomalous web server activity. Using Bro IDS (https://www.bro.org/) to generate log data from network traffic captures, an analyst can apply the same principles of stacking and statistical analysis to network traffic data with the goal of identifying outliers.

Bro IDS is a powerful network analysis framework which parses common network protocols and generates verbose log data characterizing the network traffic state. This data is extremely valuable for network forensics and hunting for suspicious behavior within network traffic. Using bro-cut to examine the Bro http.log file generated from the Deep Panda web shell network packet captures we can see that the malicious requests stick out because of both the web shell's extended file path length and the low number of requests to the web shell.



```
/usr/local/bro$ bin/bro-cut uri < http.log | sort | uniq -c
      5 -
    296 /
      1 /aspnet_client/system_web/4_0_30319/system_web.aspx
      1 /dana-na/auth/url_default/welcome.cgi
      2 /EWS/Exchange.asmx
     22 /favicon.ico
      1 /owa
      1 /owa/forms
```

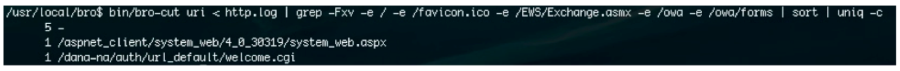
Command: bro-cut uri < http.log | sort | uniq -c

The above bro-cut command counts the number of unique *uri* field values in the http.log file and displays the respective counts. It may also be possible to discover anomalous web server activity, like a covert web shell, by racking and stacking the http.log bro log based on the following fields: source ip, destination port, http user-agent and http referrer.

Useful bro-cut commands:

```
bro-cut uri < http.log | sort | uniq -c
bro-cut id.org_h < http.log | sort | uniq -c
bro-cut id.resp_p < http.log | sort | uniq -c
bro-cut user_agent < http.log | sort | uniq -c
bro-cut referrer < http.log | sort | uniq -c
```

It is important to note that performing statistical analysis on log data such as this may very well lead to the discovery of malicious activity, but at other times may lead to an analytical dead end. A proactive approach can be taken to minimize some of the noise and increase the odds of identifying anomalous activity as malicious. Whitelisting the known good and uninteresting URIs prior to stacking the http.log data may provide a less noisy dataset. It is worth stating again that the better you know your network the more effectively your hunting activities will be.



```
/usr/local/bro$ bin/bro-cut uri < http.log | grep -Fxv -e / -e /favicon.ico -e /EWS/Exchange.asmx -e /owa -e /owa/forms | sort | uniq -c
      5 -
      1 /aspnet_client/system_web/4_0_30319/system_web.aspx
      1 /dana-na/auth/url_default/welcome.cgi
```

Command: A quick grep -Fxv will whitelist out all known good URI paths in the http.log

IP Address Analysis

Tracking adversary activity in your network may sometimes be as simple as identifying a known bad IP address communicating with one of your systems. This flagged traffic could very well be the adversary communicating with their web shell, or another backdoor in your environment. Often times, however, legitimate traffic may be flagged as suspicious requiring significant time to weed out false positives. The trick here is managing the list of “known bad” or suspicious IP addresses as best as possible. A variety of categories for such IP addresses of interest are discussed in the sections below.

Network Indicators of Compromise (IOCs)

Your network IOCs may come from previous or current investigations at your company where you’ve been able to associate an IP address to malicious activity, as well as from subscription-based intelligence feeds for a more complete and proactive set. While network indicators can be great and lead you right to compromise, it is also quite common, especially in the case of a web shell, that attackers switch up their IPs regularly making this more of a challenge.

TOR, VPN and Proxy Services

A regularly updated list of TOR exit nodes could prove useful for identifying suspicious traffic in your environment if you would not expect or want to see TOR traffic connecting to your hosts. In addition, identifying traffic that comes from commonly-used VPN and proxy services, which are regularly leveraged by adversaries, will provide further avenues for analysis. In many cases, IP ranges for these VPN and proxy services are readily available. You may quickly decide, however, that tracking TOR exit nodes and VPN/proxy solutions comes with more upkeep than you have time for, but it remains a good tactic to be aware of should you have concerns in your environment. In a recent Deep Panda investigation we found the adversary using a popular US-based VPN solution to disguise their IP address and connect to their web shell. It is important to keep in mind that legitimate users may also take advantage of TOR or VPN/proxy solutions for privacy or security benefits, so their use alone is not an indicator of compromise. However, as knowledge is power, it is often this sort of additional information that quickly helps to identify an avenue for further investigation.

Physical Location

Geolocation services may be used to determine the approximate location of an IP address. Traffic coming from or going to

false positives depending on what is "normal" for your environment. The Deep Panda web shells discussed throughout this blog series are attributed to China, however, many companies have legitimate communications with China, making this a difficult indicator to use on its own. The effectiveness of geolocation will depend greatly on your environment and whether or not the adversary utilizes VPN or proxy services.

Internal IPs

Traffic flows that are internal to your own network may be just as valuable as the external IP address analysis discussed above. Web shells are often deployed laterally by an attacker that already has a foothold in your network and would like to have a web shell ready as a backup should the need arise. Monitoring anomalous internal traffic to your web servers would be a good start to detecting this sort of lateral web shell deployment, accompanied by appropriate network segmentation where possible. HTTP Request Stacking, discussed in this blog post, may be useful to find such anomalies.

Detecting the Initial Compromise

One of the most effective ways of identifying web shells (or any adversary activity) is by identifying the initial compromise that allowed it to be placed in your environment and then narrowing in on surrounding activity. File stacking and web log analysis covered earlier in this blog series are that much easier if you can identify a timeframe or specific host for analysis.

If you have architected your network to have good packet capture data and NetFlow available, most of the detection techniques discussed previously in this series are also viable during network analysis. As an example, SQL injection, remote file inclusion and directory enumeration can be easily detected using similar steps to those discussed in Part 3 of this blog series, Mo’ Shells Mo’ Problems – Web Server Log Analysis. The main difference in the network review is the platform you’ll leverage, as you’ll want a PCAP-capable tool such as Snort or Suricata to use for request analysis and alerting.

Summary

Detecting web shells via the network can prove to be a challenge, but is feasible. Using a combination of the methodologies presented in this blog series gives security teams a better chance for detecting web shells so they don’t go unnoticed – something the adversary has enjoyed for too long. To detect web shells via the network, consider the following network approaches:

- Keep your IDS signatures up to date with best-of-breed intelligence subscriptions
- Know your network and identify anomalies
- Monitor internal and external IP traffic
- Stack your HTTP requests and see what stands out
- Look for signs of initial compromise to narrow down your search

- Combine approaches to strengthen confidence

Tweet Share

Related Content

CrowdStrike Intelligence – Adversary-based Approach

Treating the problem, not the symptomsHaving spent the better part of the last 10 years dealing…

ARMv7/Thumb2 Inline Code Hooking

At Hackito Ergo Sum 2012, I presented about Exploitation of the RenderArena allocator in WebKit (PDF) with…

Unpacking Dynamically Allocated Code

BackgroundToday, most malware is obfuscated to make it more difficult for traditional antivirus engines to detect…

CROWDSTRIKE

Copyright © 2016 CrowdStrike | Privacy | Request Info | Blog | Contact Us | 1.888.512.8906