MALWARE/
TOOLS
PROFILE

Recorded Future®

By Insikt Group®

December 14, 2021

Full Spectrum Detections for 5 Popular Web Shells: Alfa, SharPyShell, Krypton, ASPXSpy, and TWOFACE

Recorded Future®

This report provides a technical overview of 5 prominent web shells: Alfa, Krypton, SharPyShell, ASPXSpy, and TWOFACE. It contains details on the capabilities of the web shells and host-based and network-based detections. This report is intended for security operations audiences who focus on detection engineering. Sources include the Recorded Future Platform®, GreyNoise, Shodan, and BinaryEdge.

Executive Summary

Web shells are common and powerful tools used by threat actors to maintain access to public-facing web servers. They are lightweight, sometimes containing as few as 4 lines of code, and let threat actors execute secondary payloads, escalate privileges, exfiltrate data, and move laterally within the compromised network. Web shells often go undetected due to the small footprint left during their use, an organization's limited visibility of their public-facing servers, and the ability for web shell-associated network traffic to blend in with normal web server activity. Our research provides a full-spectrum approach to detecting web shells, combining log analysis, network analysis, and web shell scanning techniques. We focus on a subset of web shells recently used by state-sponsored and criminal threat actors: Alfa, SharPyShell, Krypton, ASPXSpy, and TWOFACE. Our methodology and detections can be applied internally for defenders but also by security researchers hunting for the presence of web shells on externally facing servers.

Key Judgments

- Web shells will continue to be used by both APTs and financially motivated threat actors, primarily due to their ease of use and their difficulty in being detected.
- We identified 4 techniques to detect web shells that can be used together: YARA rules, Sigma rules, network traffic patterns, and internal/external scanning. While these methods are not foolproof, they provide diverse opportunities for defenders to look for web shells on their systems.
- Security teams with limited host and network visibility can still detect web shells on their systems using HTTP scanning techniques.
- As long as threat actors can viably exploit public-facing servers, they will continue to use web shells to maintain persistence and provide additional capabilities.

Background

Web shells are pieces of malicious code planted by a threat actor on a web server that allow the threat actor to execute commands or access files on the remote server. They are most often written for PHP or Active Server Pages (ASP) as these are currently the most common website programming languages. Web shells can be employed for various purposes, including gaining persistence, executing commands, downloading files, or dropping another tool for a subsequent stage of an attack.

A common scenario in which web shells are deployed is that a threat actor, either opportunistically or in a targeted intrusion, will exploit a vulnerability in a public-facing application or server. Depending on the exploit, an attacker may have limited privileges to the system; additionally, if the attacker's connection terminates, they will have to rerun the exploit to gain access. Deploying a web shell provides an attacker with a persistent connection and additional capabilities. The graphic below from the Microsoft Threat Intelligence Center shows a high-level overview of this approach.

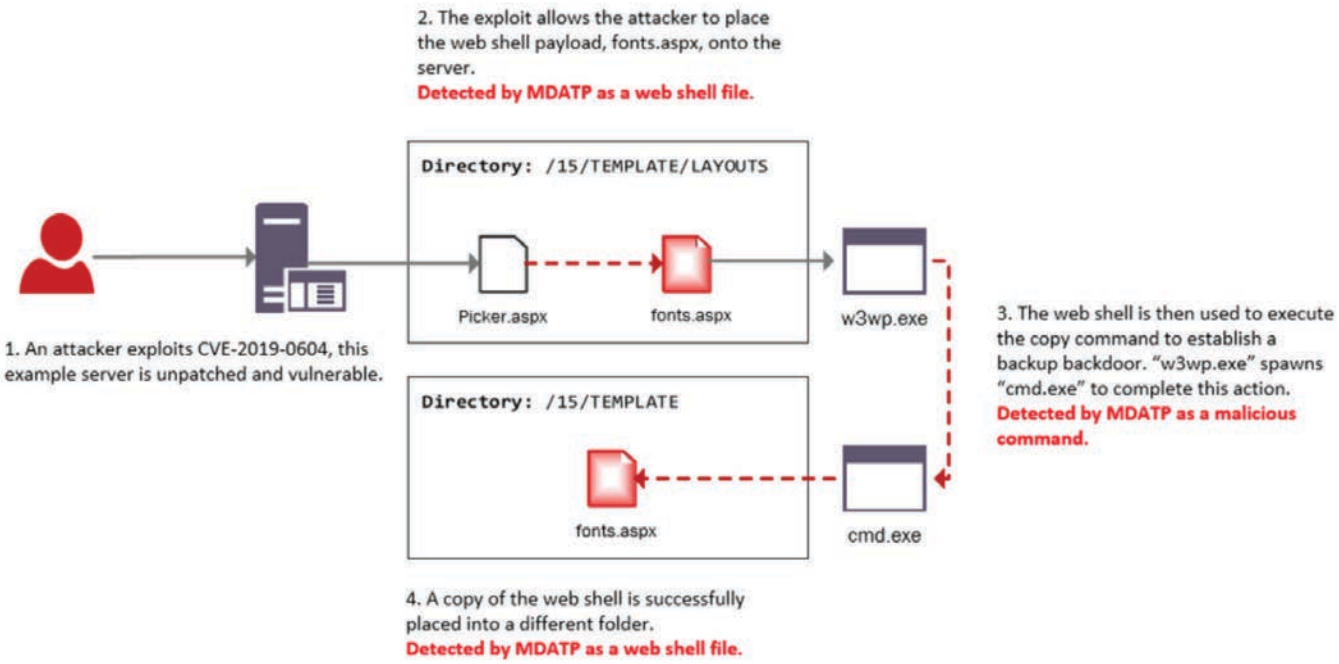

Figure 1: Common web shell Installation (Source: Microsoft)

Over the last year, we have observed several cyberattacks in which web shells were used:

- HAFNIUM, the Chinese APT group, uploaded the China Chopper web shell to compromised Microsoft IIS servers earlier this year. China Chopper allows threat actors to execute JScript code on the victim machine, in turn allowing them to access files, execute processes, or create a reverse shell.
- The compromise of the Accellion File Transfer Appliance (FTA) file-sharing service affecting nearly 100 clients was primarily enabled by 4 zero-day vulnerabilities in the tool. Those vulnerabilities allowed threat actors to place the DEWMODE web shell on victim servers and exfiltrate files from those servers. DEWMODE enabled the threat actor to view or download files of interest.
- The SUPERNOVA web shell was deployed to servers vulnerable to CVE-2020-10148 in late 2020 by threat actors linked to the Spiral threat group. The Spiral threat group is suspected to be of Chinese origin. In the incident, a compromised SolarWinds server was used to deploy the web shell. The use of SUPERNOVA was unrelated to the SUNBURST supply chain attack that was discovered in December 2020. While both use SolarWinds Orion components, SUPERNOVA leverages a vulnerability and is not digitally signed. Those two factors differentiate SUPERNOVA from SUNBURST.

Threat Analysis

For this research, Insikt Group selected 5 web shells to create detections for: AlfaShell, KRYPTON, SharPyShell, ASPXSPY, and TWOFACE. We chose these web shells for their popularity and use among state-sponsored and criminal threat actors.

Alfa Team Shell

AlfaShell (Alfa Team Shell) has been publicly available since at least 2013, notably appearing on the Persian-language forums Ashiyane and Iranian Dark Coders Team Forum. APT33 has been a prominent user of AlfaShell. Version 4.1 of the tool (dubbed Tesla) is available on GitHub and includes extensive functionality. The tool has a very verbose user interface, making it easy for less experienced operators to use on compromised servers. AlfaShell initially gathers a large amount of data to provide information about the victim host to the user.

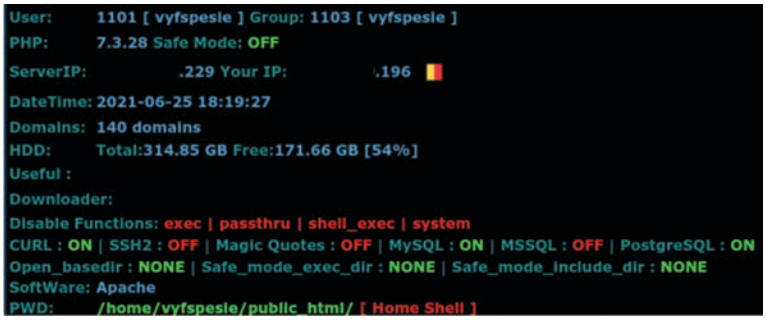

Figure 2: AlfaShell GUI showing information about victim host (Source: Recorded Future)

AlfaShell can upload and download files, as well as execute commands. The tool also includes more advanced features, such as pre-built tooling to send spearphishing emails, deface a victim's domains, implement a fake web page, dump databases, upload a backdoor, and inject a web shell elsewhere on the host. However, the advanced features come at a cost, inflating the web shell's file size to over 150 KB.

SharPyShell

SharPyShell is an open-source ASP.NET web shell that only supports C# applications running on .NET Framework >=2.0. SharPyShell executes commands by compiling them in-memory at runtime. SharPyShell receives encrypted commands, decrypts them, runs the commands, and returns the response. Commands are sent from the SharPyShell Python client and include downloading files, executing shell commands via cmd.exe, running PowerShell scripts, escalating privileges, running Mimikatz, and enabling lateral movement via WMIC. A full list of commands can be found in Table 1.

Command	Function
download	Download a file from the server
exec_cmd	Run a cmd.exe /c command on the server
exec_ps	Run a powershell.exe -nop -noni -enc 'base64command' on the server
inject_dll_reflective	Inject a reflective DLL in a new (or existing) process
inject_dll_srdi	Inject a generic DLL in a new (or existing) process
inject_shellcode	Inject shellcode in a new (or existing) process
invoke_ps_module	Run a ps1 script on the target server
invoke_ps_module_as	Run a ps1 script on the target server as a specific user
lateral_psexec	Run psexec binary to move laterally
lateral_wmi	Run builtin WMI command to move laterally
mimikatz	Run an offline version of mimikatz directly in memory
net_portscan	Run a port scan using regular sockets, based (pretty) loosely on nmap
privesc_juicy_potato	Launch InMem Juicy Potato attack trying to impersonate NT AUTHORITY\SYSTEM
privesc_powerup	Run Powerup module to assess all misconfiguration for privesc
runas	Run a cmd.exe /c command spawning a new process as a specific user
runas_ps	Run a powershell.exe -enc spawning a new process as a specific user
upload	Upload a file to the server

Table 1: SharPyShell commands (Source: GitHub)

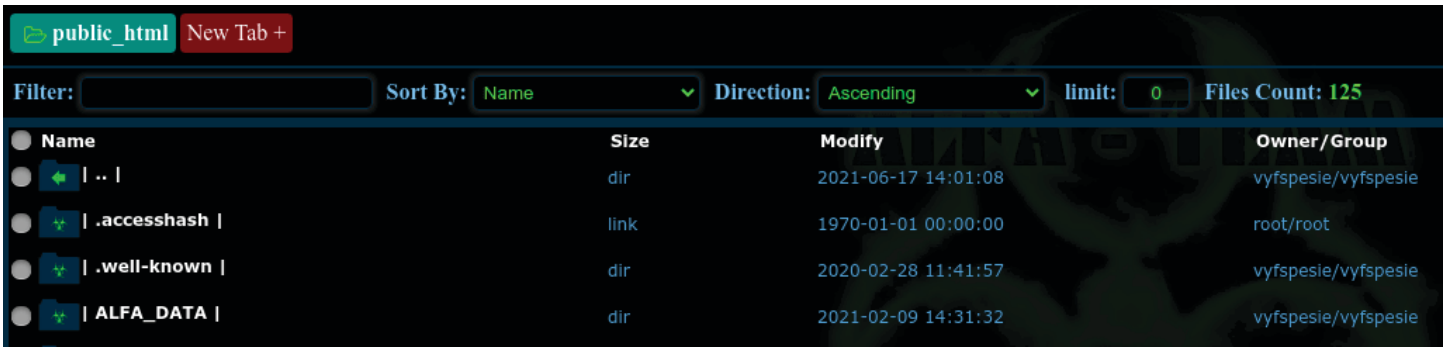

Figure 3: AlfaShell GUI showing directory information (Source: Recorded Future)

KRYPTON

Krypton is a web shell used by Turla operators as an initial foothold. The web shell is protected, meaning that it will only function when keys are passed via HTTP(S) request to the web shell; otherwise, the web shell will not resolve or respond to commands. Krypton is a C# web shell, but unlike ASPXSPY, Alfa Team Shell, SharPyShell, and TWOFACE, we did not have access to the client portion needed to interact with the web shell. After analyzing the KRYPTON ASP code, Insikt Group developed a Python script to interact with the Krypton web shell, which can be downloaded from our GitHub repository. The Krypton web shell accepts 6 parameters to run commands on the victim server.

Case	Command / Parameter
cmd	Run a command in cmd.exe
put	Upload a file
update	Modify content of a file
time	Time stomp a file
del	Delete a file
get	Download a file

Table 2: KRYPTON commands and parameters (Source: Recorded Future)

The web shell encodes data in base64 and encrypts it with AES to conceal its network traffic. The sample tested by Insikt Group used the key "J8fs4F4rnP7nFI#f" and the IV "D68gq#5p0(3Ndsk!". Turla has previously relied on password-protected web shells to enable intrusion operations, using them to operate hacked WordPress sites as command and control infrastructure.

ASPXSpy

ASPXSpy is an open-source web shell written in C# that allows a threat actor to accomplish various post-exploitation tasks, including file access and command execution. ASPXSpy has been used by high-end espionage groups such as APT39, APT41, and HAFNIUM.

In addition to running commands on the victim host, the web shell can run SQL queries, extract credentials from the infected server, identify running processes, and use nmap to scan other address spaces.

TWOFACE

TWOFACE, also called SEASHARPEE, HighShell, and HyperShell, is a two-stage web shell originally used by APT34 operators. The web shell had its code leaked by Lab Dookhtegan and has since been borrowed by UNC215 (with rough links to APT27) and co-opted by Turla, after Turla took over APT34 infrastructure to support their operations.

The web shell is written in C# and features a password-protected loader that drops the main web shell component. The loader component uses an evasion technique that resolves to a decoy web page if accessed via a web browser; its functionality is only activated when specific data is passed to the web shell in an HTTP(S) request. The loader then waits for an HTTP(S) request containing a salted decryption key in the body to decrypt and load the payload web shell to a specified location.

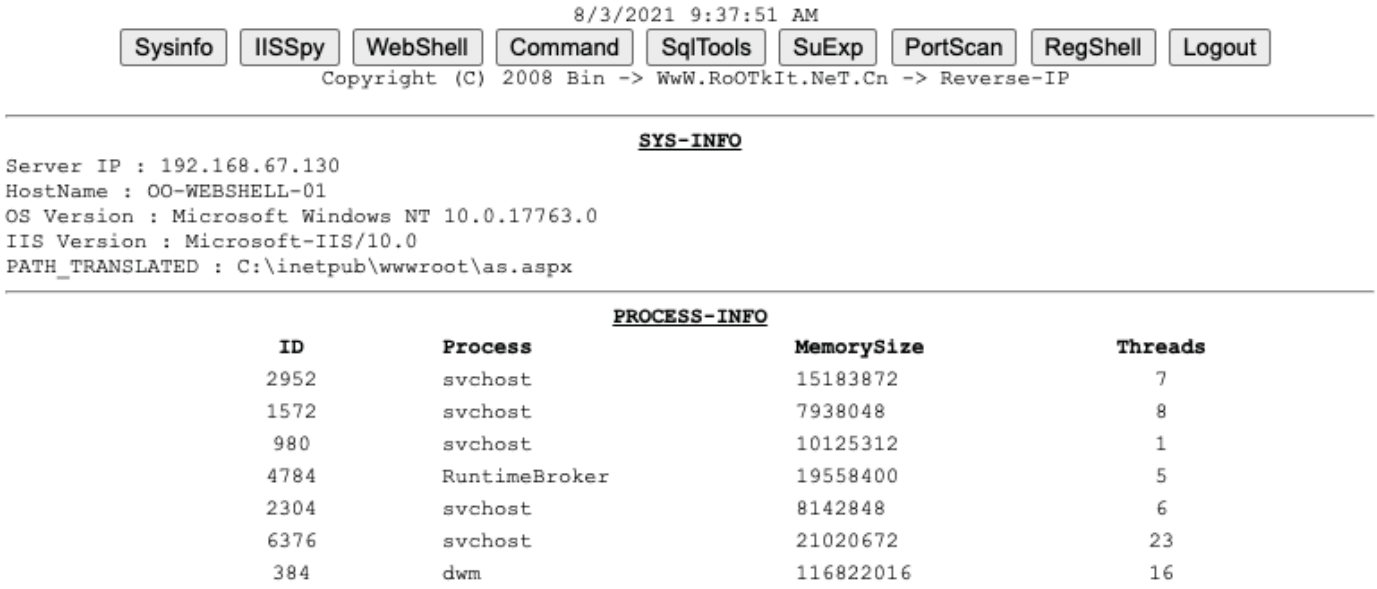

Figure 4: ASPXSpy user interface (Source: Recorded Future)

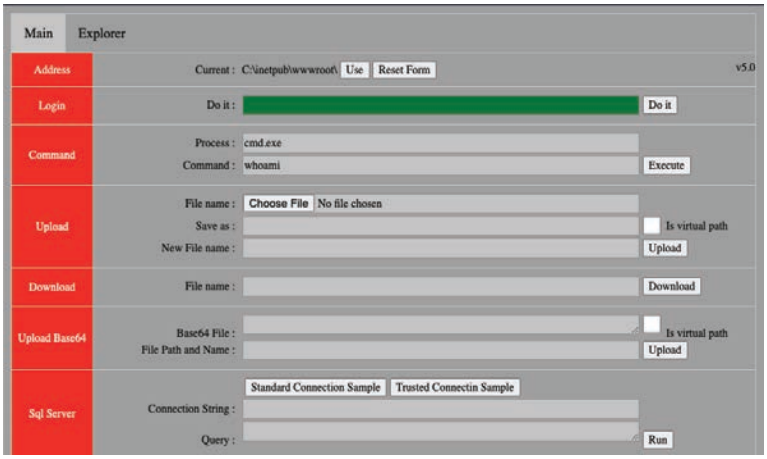

Figure 5: TWOFACE user interface (Source: Recorded Future)

The decrypted web shell component, shown in Figure 5, is password-protected to prevent anyone who may stumble upon the web shell from issuing commands. Once the password is provided (and saved in the cookie field), the full functionality of the web shell is available. The features include file upload and download, running shell commands in a specified process, the ability to timestomp files at a given location, and querying a SQL database. TWOFACE recognizes the following commands:

Field	Command / Parameter
Do it	Login with supplied password
Execute	Command execution
Upload	Upload file to server. Can also upload file base64 encoded
Download	Download file
Run	SQL Server connection and Query
Get/Set	Get or Set timestamps

Table 3: TWOFACE commands (Source: Recorded Future)

Full Spectrum Web Shell Detection

Cybersecurity teams, defenders, and security researchers looking to detect web shells have options for host-based detection with Sigma, file detection with YARA, network detections with IDS, and external scanning for anomalous and suspicious indicators.

The figure below shows a high-level overview of where you can apply our detections to provide full-spectrum detection of web shells. Our detections fall into three categories:

- **Network:** Using network triggers (IDS rules) to identify authentication or command execution of web shells.
- **Host:** Using Sigma rules to detect behavior related to commands being executed from a web shell.

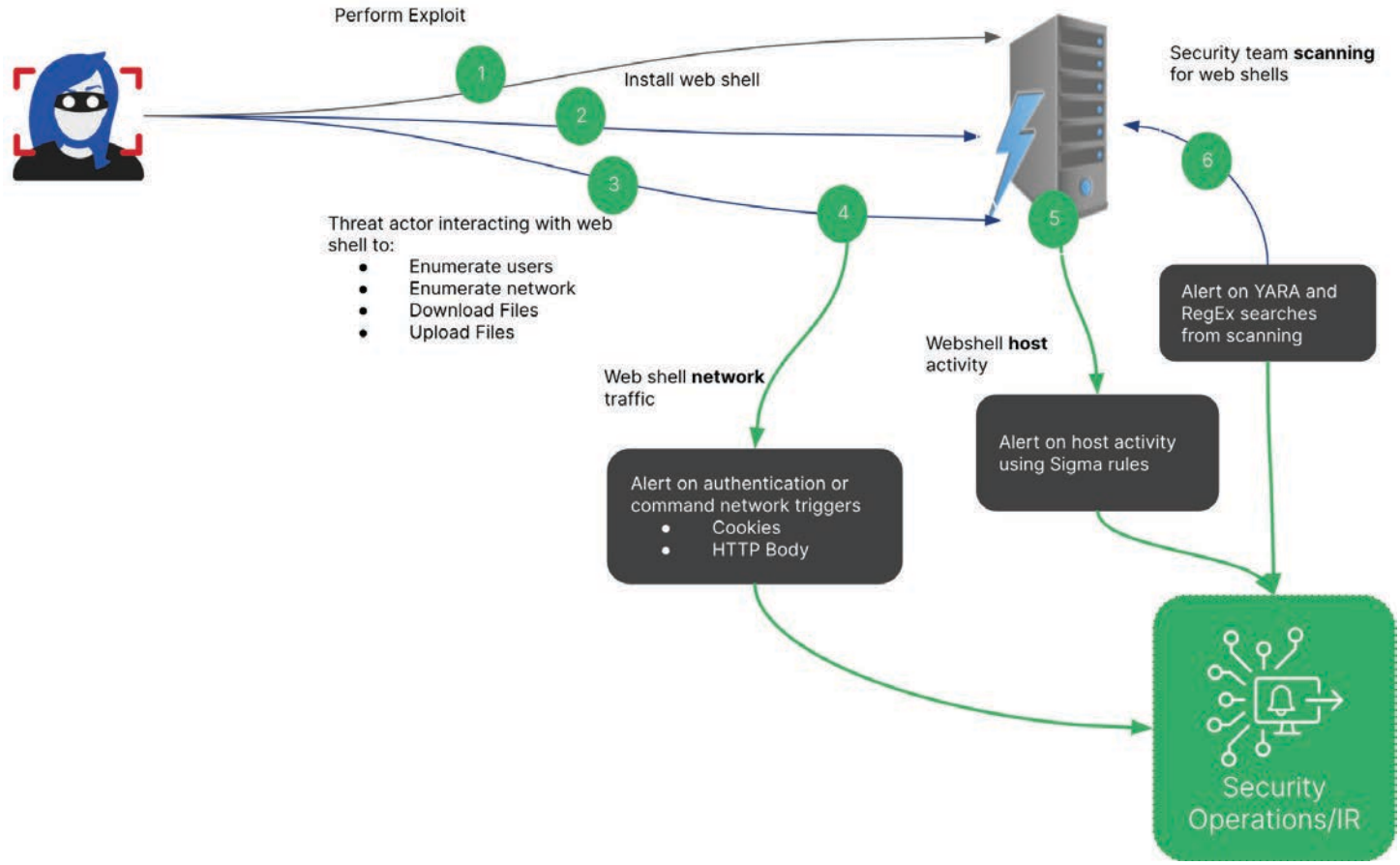

Figure 6: Full-spectrum web shell detection (Source: Recorded Future)

- **Scanning (external or internal):** Identifying vulnerable servers and scanning for common URLs used by web shells. Performing YARA and pattern-matching searches on retrieved content to detect web shells.

The more visibility and logging you have in each of these detection points, the greater the chance you have to detect web shell activity.

Web shells often are used after initial access is obtained. Threat actors will move laterally and may deploy additional tooling to achieve their objective. For this reason, once a web shell detection has been confirmed, Insikt Group recommends additional analysis be performed to identify the full scope of the intrusion, including but not limited to:

- The vector of infection
- The type of web shell used
- Evidence of credential harvesting, data exfiltration, or lateral movement

Web Shell Activity Emulation

The detections Insikt Group created are based on our emulation of each web shell in a lab environment and do not specifically detect all of the ways a web shell could be created or run on the system. We focused on generating logs based on web shell use that is common across attackers to build detections. We focused on the steps identified below that have been regularly observed in web shell compromises for our detections.

1. Authentication
2. Recon commands
    a. whoami
    b. ipconfig
    c. net user
3. File movement or modification
    a. Upload a file
    b. Download a file
    c. Time stomp
4. Miscellaneous Lateral Movement
    a. Mimikatz
    b. Read and modify registry
    c. Scan a port

Sigma

Insikt Group created Sigma rules for each web shell covered here by evaluating Sysmon logs (using the SwiftOnSecurity configuration for Sysmon here) generated during the adversary emulation process.

For the Windows-based web shells, it was possible to create Sigma detections using Sysmon logging for the reconnaissance commands run during the emulation. Most detections were based on process creation events of "cmd.exe" where the command line value included the particular command the threat actor would run — whoami, ipconfig, or net user. While these commands can also be run by administrators on a Windows system, when the commands are executed by the web shells, there are unique artifacts in either the parent process information, the directory in which the command was executed, or the user executing the command. For instance, commands issued by ASPXSPY and TWOFACE run with the current directory value of "c:\windows\system32\inetsrv\" by default. Additionally, commands executed from KRYPTON and SharPyShell contain the parent image "C:\WINDOWS\System32\inetsrv\w3wp.exe" and ASPXSPY, TWOFACE, KRYPTON and SharPyShell all operate under the default application pool identity, "IIS APPPOOL\DefaultAppPool". Table 4 summarizes the triggers we used to build our Sigma detections.

	Host Triggers

Alfa	• Alfa-specific strings in audited logs:
	  • Getheader.alfa (Alfa webpage header update)
	  • Alfa.zip (File compressor component)
	  • Symperl.alfa (Symlink creation component)

ASPXSPY	• Command execution from the directory "C:/WINDOWS/System32/inetsrv/" • Default user is "IIS APPPOOL\DefaultAppPool"
KRYPTON	• Parent image is "C:\WINDOWS\System32\inetsrv\w3wp.exe" • Default user is "IIS APPPOOL\DefaultAppPool"
SharPyShell	• Parent image is "C:\WINDOWS\System32\inetsrv\w3wp.exe" • Default user is "IIS APPPOOL\DefaultAppPool"
TWOFACE	• Command execution from the directory "C:/WINDOWS/System32/inetsrv/" • Default user is "IIS APPPOOL\DefaultAppPool"

Table 4: Summary of web shell host triggers used to build Sigma Rules (Source: Recorded Future)

For file transfer and lateral movement, there were no distinctive Sysmon events that could be used for detection. In addition, while Mimikatz left artifacts in the Sysmon logs, they were not unique to a particular web shell. It is only with SharPyShell that we can detect the loading of modules used for Mimikatz or port scanning. However, we cannot distinguish what module is being loaded. Although a Sigma rule for Mimikatz would be beneficial, there are already several open source detections currently available.

Traffic Patterns

Many web shells rely on network traffic triggers sent in plaintext. The most common methods use cookies in the body of the HTTP request and extensions of the URI string to deliver commands. Web shells can use HTTP or HTTPS depending on the configuration of the compromised server; for our analysis, we did not use HTTPS. The table below provides a summary of network traffic triggers for each web shell we analyzed.

Additional details for each trigger are provided in the following sections.

Cookie-Based Communication

After authentication with a password, ASPXSpy installations use a cookie parameter to validate the user's interaction with the shell. The use of a cookie name of ASP.NET_SessionID is not globally unique, but if such cookies are not used in the client environment, it can be honed in on for detection. ASPXSPY Commands can also be delivered in the cookie field.

Additionally, the TWOFACE web shell sends commands in the cookie field, delineated by vertical slash and pound symbols. The fields appear in plaintext, while the commands are encoded with base64. The field names can be found in Figure 3.

The Krypton web shell also uses cookies to transport commands and data to the web shell. As a part of this communication process, the HTTP cookie fields are all prepended with the string "cmd=" followed by a base64 encoded string, creating a detection opportunity for Krypton traffic.

Alfa Team Shell uses the cookie header to transmit commands in clear text; the "alfa-terminal-history" name will contain the command(s) executed in the current session. In the example shown in Figure 10, it contains the "whoami" command.

We recommend looking in cookie fields for inbound traffic to identify cookies not assigned by the webserver under normal operation and for evidence of commands being issued, either encoded or in plaintext.

	Cookies	HTTP Body	User-Agent
Alfa	Commands Hunting Tip: Look for the pattern "alfa-terminal-history=[<commands>]"		
ASPXSPY		Commands Hunting Tip: Look for HTTP POSTs with "boundary=" in the Content-Type header and "Content-Disposition: form-data" in the payload	
KRYPTON	Commands Hunting Tip: Look for the patten "cmd=<base64blob>"		
SharPyShell		Commands Hunting Tip: Look for HTTP POSTs with "boundary=" in the Content-Type header and "Content-Disposition: form-data; name="data"" in the payload	Configuration Hunting Tip: The User-agent below, while not unique to SharPyShell, is hardcoded and has to be manually changed at runtime. "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:62.0)" Gecko/20100101 Firefox/62.0"
TWOFACE	Commands Hunting Tip: Look for the pattern "data=pro#=#<base64blob>=#|cmd#=<base64blob>"		

Table 5: Summary of web shell network triggers (Source: Recorded Future)

```
POST /as.aspx HTTP/1.1
Host: 192.168.67.130
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:90.0) Gecko/20100101 Firefox/90.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 646
Origin: http://192.168.67.130
Connection: keep-alive
Referer: http://192.168.67.130/as.aspx
Cookie: ASP.NET_SessionId=32mbwgvc33mrlw5kjfp1qzco
Upgrade-Insecure-Requests: 1
```

Figure 7: ASPXSpy HTTP POST request with ASP.NET session ID authentication cookie (Source: Recorded Future)

```
POST /tf.aspx HTTP/1.1
Host: 192.168.67.130
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:90.0) Gecko/20100101 Firefox/90.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 33
Origin: http://192.168.67.130
Connection: keep-alive
Referer: http://192.168.67.130/tf.aspx
Cookie: data=pro#=##|#cmd#=##|#sav#=##|#vir#=##|#nen#=##|#don#=##|#tfil#=##|#ttar#=##|
#ttim#=#|#sqc#=#|#sqq#=#|#exadd#=#; p=VGghc04wdEYwckZBTg==
Upgrade-Insecure-Requests: 1
```

Figure 8: TWOFACE HTTP POST request with cookie containing blank delimited data fields (Source: Recorded Future)

```
Cookie
cmd=aw2t6jKTnxnZVp1HchbANg==
cmd=6aLzmWInlpmtiBxV94t4KVzydsqfxd2LfBqfdfa+S14=
cmd=xs8LJtEOuHq0ymLm9zrRq2CYlFjswo4K3Xo/1iOYTJo=
cmd=+8EwpQruMhyiwPER0+zaizYJ8I/hz2mf2iRuGq4eyIE=
cmd=uaJ3JgnR7NFjm7ImsbGG37cr+ucxvjmG9/09uMyMrebUO2ZWaFozBEF4Opis1WKh
cmd=HgT/OM8V6KzhAOTqnadTD5jLPh9hsOR94cLVilikxuA=
cmd=wxUVZ7owHESjv3ru/5+ejjkdBNMk8sxn8wAyGFIbUck=
cmd=HgT/OM8V6KzhAOTqnadTD5jLPh9hsOR94cLVilikxuA=
cmd=HgT/OM8V6KzhAOTqnadTD/VV6p6z2CmKB04NpiM/sQk=
cmd=HgT/OM8V6KzhAOTqnadTD5jLPh9hsOR94cLVilikxuA=
cmd=aw2t6jKTnxnZVp1HchbANg==
cmd=6aLzmWInlpmtiBxV94t4KVzydsqfxd2LfBqfdfa+S14=
cmd=+8EwpQruMhyiwPER0+zaizYJ8I/hz2mf2iRuGq4eyIE=
cmd=uaJ3JgnR7NFjm7ImsbGG37cr+ucxvjmG9/09uMyMrebUO2ZWaFozBEF4Opis1WKh
cmd=HgT/OM8V6KzhAOTqnadTD5jLPh9hsOR94cLVilikxuA=
```

Figure 9: Krypton web shell shifting cookie values (Source: Recorded Future)

```
POST /alfa.php HTTP/1.1
Host: 192.168.67.131
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 74
Origin: http://192.168.67.131
Connection: keep-alive
Referer: http://192.168.67.131/alfa.php
Cookie: alfacgiapi_mode=off; alfa_canruncmd=true; alfa-terminal-history=["whoami"]

a=dGVybWluYWxFeGVj&c=L3Zhci93d3cvd2Vic2hlbGwv&alfa1=d2hvYW1p&ajax=dHJ1ZQ==HTTP/1.1 200 OK
Date: Tue, 14 Sep 2021 14:29:36 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 31
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

{"output":"www-data","path":""}
```

Figure 10: Alfa Team Shell web shell cookie command (Source: Recorded Future)

```
POST /as.aspx HTTP/1.1
Host: 192.168.67.130
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:90.0) Gecko/20100101 Firefox/90.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data;
boundary=---------------------------425007131340039662202626606179
Content-Length: 2731
Origin: http://192.168.67.130
Connection: keep-alive
Referer: http://192.168.67.130/as.aspx
Cookie: ASP.NET_SessionId=32mbwgvc33mrlw5kjfp1qzco
Upgrade-Insecure-Requests: 1

-----------------------------425007131340039662202626606179
Content-Disposition: form-data; name="__VIEWSTATE"
```

Figure 11: ASPXSpy HTTP POST request (Source: Recorded Future)

Figure 12: Truncated SharpyShell HTTP POST request body (Source: Recorded Future)

HTTP Body-Based Communication

After authentication with a password, ASPXSpy sends commands in the body of the HTTP POST request, delineated by a boundary flag.

SharpyShell similarly uses content-disposition and form boundaries to flag data being passed to the web shell. However, this data is encrypted with AES before being encoded with base64.

The China Chopper web shell has been widely used in attacks by numerous Chinese state-sponsored groups, dating from 2013 to the present day. China Chopper operators pass data to the web shell in plaintext in the body of the HTTP request.

URI-Based Communication

More common web shells will deliver command and authentication parameters in the body of the URI, particularly PHP web shells. While these are not considered the most sophisticated or stealthy threat, they are very common, and their detection should be taken seriously.

URI-based patterns can be used to detect web shell traffic in other ways. For example, a visible pattern for SharPyShell consists of repeated HTTP POST requests for an .aspx file, as seen in Figure 14.

Web Shell Scanning and Discovery

Security teams that have adequate visibility into their host and network activity will succeed in detecting web shells using a combination of our Sigma rules and our network traffic indicators. These detections are behavior-based and require that the appropriate logs be captured, sent to a SIEM, and then actioned.

Another approach that is less reliant on logging and can be taken by both internal security teams and security researchers is scanning endpoints for web shells. This approach varies depending on whether an internal team is scanning their infrastructure or a security researcher is scanning for web shells on an internet-wide scale.

Security Team Scanning

Security teams that lack host or network logging capabilities may still identify web shells by scanning their web or Exchange servers. Security teams with host and network logging can also benefit from this approach as another avenue for detection.

```
POST /owa/auth/i.aspx HTTP/1.1
Cache-Control: no-cache
X-Forwarded-For: 100.159.23.246
Referer: [redacted]
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: [redacted]
Content-Length: 1089
Connection: Close
Cookie: OutlookSession=5289564ad77444f4a851fc4e48751f6b

pass=Response.Write("->|");var err:Exception;try{eval(System.Text.Encoding.GetEncoding
(65001).GetString(System.Convert.FromBase64String
("dmFyIEQ9U3lzdGVtLlRleHQuRW5jb2RpbmcuR2V0RW5jb2RpbmcoNjUwMDEpLkdldFN0cmluZyhTeXN0ZW0uQ29udm
VydC5Gcm9tQmFzZTY0U3RyaW5nKFJlcXVlc3QuSXRlbVsiejEiXSkpO3ZhciBtPW5ldyBTeXN0ZW0uSU8uRGlyZWN0b3
J5SW5mbyhEKTt2YXIgcz1tLkdldERpcmVjdG9yaWVzKCk7dmFyIFA6U3RyaW5nO3ZhciBpO2Z1bmN0aW9uIEEocDpTdH
JpbmcpOkludDMye3JldHVybiBTeXN0ZW0uSU8uRmlsZS5HZXRBdHRyaWJ1dGVzKHApO31mdW5jdGlvbiBUKHA6U3Ryaw
5nKTpTdHJpbmd7cmV0dXJuIFN5c3RlbS5JTy5GaWxlLkdldExhc3RXcml0ZVRpbWUocCkuVG9TdHJpbmcoInl5eXktTU
0tZGQgSEg6bW06c3MiKTt9Zm9yKGkgaW4gcyl7UD1EK3NbaV0uTmFtZTtSZXNwb25zZS5Xcml0ZShzW2ldLk5hbWUrIi
9cdCIrVChQKSsiXHQwXHQiK0EoUCkrIlxuIik7fXU7Response.End();",
("dmFyIEQ9U3lzdGVtLlRleHQuRW5jb2RpbmcuR2V0RW5jb2RpbmcoNjUwMDEpLkdldFN0cmluZyhTeXN0ZW0uQ29udm
VydC5Gcm9tQmFzZTY0U3RyaW5nKFJlcXVlc3QuSXRlbVsiejEiXSkpO3ZhciBtPW5ldyBTeXN0ZW0uSU8uRGlyZWN0b3
J5SW5mbyhEKTt2YXIgcz1tLkdldERpcmVjdG9yaWVzKCk7dmFyIFA6U3RyaW5nO3ZhciBpO2Z1bmN0aW9uIEEocDpTdH
JpbmcpOkludDMye3JldHVybiBTeXN0ZW0uSU8uRmlsZS5HZXRBdHRyaWJ1dGVzKHApO31mdW5jdGlvbiBUKHA6U3Ryaw
5nKTpTdHJpbmd7cmV0dXJuIFN5c3RlbS5JTy5GaWxlLkdldExhc3RXcml0ZVRpbWUocCkuVG9TdHJpbmcoInl5eXktTU
0tZGQgSEg6bW06c3MiKTt9Zm9yKGkgaW4gcyl7UD1EK3NbaV0uTmFtZTtSZXNwb25zZS5Xcml0ZShzW2ldLk5hbWUrIi
9cdCIrVChQKSsiXHQwXHQiK0EoUCkrIlxuIik7fXU7Response.End();",
("dmFyIEQ9U3lzdGVtLlRleHQuRW5jb2RpbmcuR2V0RW5jb2RpbmcoNjUwMDEpLkdldFN0cmluZyhTeXN0ZW0uQ29udm
VydC5Gcm9tQmFzZTY0U3RyaW5nKFJlcXVlc3QuSXRlbVsiejEiXSkpO3ZhciBtPW5ldyBTeXN0ZW0uSU8uRGlyZWN0b3
J5SW5mbyhEKTt2YXIgcz1tLkdldERpcmVjdG9yaWVzKCk7dmFyIFA6U3RyaW5nO3ZhciBpO2Z1bmN0aW9uIEEocDpTdH
JpbmcpOkludDMye3JldHVybiBTeXN0ZW0uSU8uRmlsZS5HZXRBdHRyaWJ1dGVzKHApO31mdW5jdGlvbiBUKHA6U3Ryaw
5nKTpTdHJpbmd7cmV0dXJuIFN5c3RlbS5JTy5GaWxlLkdldExhc3RXcml0ZVRpbWUocCkuVG9TdHJpbmcoInl5eXktTU
0tZGQgSEg6bW06c3MiKTt9Zm9yKGkgaW4gcyl7UD1EK3NbaV0uTmFtZTtSZXNwb25zZS5Xcml0ZShzW2ldLk5hbWUrIi
9cdCIrVChQKSsiXHQwXHQiK0EoUCkrIlxuIik7fXU7Response.End();","unsafe");}catch(err){Response.Write("ERROR:// "%2Berr.message);}Response.Write("|
<-");Response.End
```

Figure 13: Data being transferred in HTTP body to China Chopper web shell (Source: Crowdstrike)

```
GET /examples/%252e%252e/webshell/index.jsp?cmd=ifconfig HTTP/1.1
Host: 192.168.141.136
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
Iceweasel/31.8.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.141.136/examples/%252e%252e/webshell/
Connection: keep-alive
```

Figure 14: Example web shell command delivered via URI (Source: PacketTotal)

```
17:50:48.267766 192.168.67.1      192.168.67.130    HTTP     708 POST /sharp.aspx HTTP/1.1
17:50:48.270823 192.168.67.130    192.168.67.1      TCP       56 80 → 38694 [ACK] Seq=1 Ack=2113 Win=2102272 Len=0
17:50:59.837073 192.168.67.130    192.168.67.1      HTTP     296 HTTP/1.1 200 OK  (text/html)
17:50:59.837110 192.168.67.1      192.168.67.130    TCP       56 38694 → 80 [ACK] Seq=2113 Ack=241 Win=64128 Len=0
17:50:59.839210 192.168.67.1      192.168.67.130    TCP     1516 38694 → 80 [ACK] Seq=2113 Ack=241 Win=64128 Len=1460
17:50:59.839222 192.168.67.1      192.168.67.130    TCP     1516 38694 → 80 [ACK] Seq=3573 Ack=241 Win=64128 Len=1460
17:50:59.839224 192.168.67.1      192.168.67.130    HTTP     568 POST /sharp.aspx HTTP/1.1
17:50:59.840538 192.168.67.130    192.168.67.1      TCP       56 80 → 38694 [ACK] Seq=241 Ack=5545 Win=2102272 Len=0
17:51:00.138319 192.168.67.130    192.168.67.1      HTTP     320 HTTP/1.1 200 OK  (text/html)
17:51:00.138358 192.168.67.1      192.168.67.130    TCP       56 38694 → 80 [ACK] Seq=5545 Ack=505 Win=64128 Len=0
17:51:36.900368 192.168.67.1      192.168.67.130    TCP     1516 38694 → 80 [ACK] Seq=5545 Ack=505 Win=64128 Len=1460
17:51:36.900383 192.168.67.1      192.168.67.130    TCP     1516 38694 → 80 [ACK] Seq=7005 Ack=505 Win=64128 Len=1460
17:51:36.900385 192.168.67.1      192.168.67.130    HTTP     372 POST /sharp.aspx HTTP/1.1
17:51:36.901733 192.168.67.130    192.168.67.1      TCP       56 80 → 38694 [ACK] Seq=505 Ack=8781 Win=2102272 Len=0
17:51:37.554158 192.168.67.130    192.168.67.1      HTTP     316 HTTP/1.1 200 OK  (text/html)
17:51:37.554255 192.168.67.1      192.168.67.130    TCP       56 38694 → 80 [ACK] Seq=8781 Ack=765 Win=64128 Len=0
17:51:41.973429 192.168.67.1      192.168.67.130    TCP     1516 38694 → 80 [ACK] Seq=8781 Ack=765 Win=64128 Len=1460
17:51:41.973443 192.168.67.1      192.168.67.130    TCP     1516 38694 → 80 [ACK] Seq=10241 Ack=765 Win=64128 Len=1460
```

Figure 15: Repetitive POST transmissions of SharPyShell (Source: Recorded Future)

We have developed a Python script that takes a list of domains or IPs and scans for web shell indicators. The way the script works is:

1. For each domain or IP provided, the script will append common web shell URI paths. This will create a list of URLs to scan.
2. The script will then perform a HTTP GET request on each URL to retrieve the content of the webpage.
3. YARA rules are used to identify suspicious web shell components in the content.
4. Regular expressions used in this repository are used to identify suspicious web shell components in the content.
5. Results are displayed in the console.

The only requirements are that the security team knows the IP addresses and domains of their public-facing servers and have the appropriate rights and permissions to perform the scanning. This script does not require local access to the servers as we are interacting with the hosts over HTTPS, similar to how a threat actor would interact with their web shell.

Performing External Scanning

Security researchers can use the same script as internal security teams; however, a security team scanning their own infrastructure is more practical than a security researcher scanning the whole internet, as the combination of URIs, domains, and IP addresses makes the number of URLs to scan unrealistic.

To more broadly scan for web shells, filtering has to be applied, and at a level where there is minimal quality loss of the data set being scanned. To do this, we filter on common exploits involving public-facing applications or servers.

As shown in Figure 16, CVE-2021-33766 appears to be a relevant vulnerability to target. This query in Shodan can help to identify hosts and domains that are running Microsoft Exchange. By downloading the results, additional filtering can be done to identify hosts with the CVE-2021-33766 vulnerability. Another filter can be applied based on location, such as United States or United Kingdom targets. The filtered data set should provide a more realistic number of hosts for web shell scanning.

Web Shell Scanning YARA

YARA rules were created by Insikt Group to scan the HTTP responses for the presence of ASPXSPY, TWOFACE and Alfa Team Shell web shells. These rules are most effective when run against HTTP responses generated by interacting with a web shell or by our web shell scanner above and are less effective if run against static files. The reason is that various web shells will obfuscate their code statically to avoid detection; however, when processed through the web server's scripting engine, they are deobfuscated. An example of this is shown in the figure below using the Alfa Team Shell. The left side shows the obfuscated contents of the file, and the right side shows the HTTP response containing the deobfuscated code.

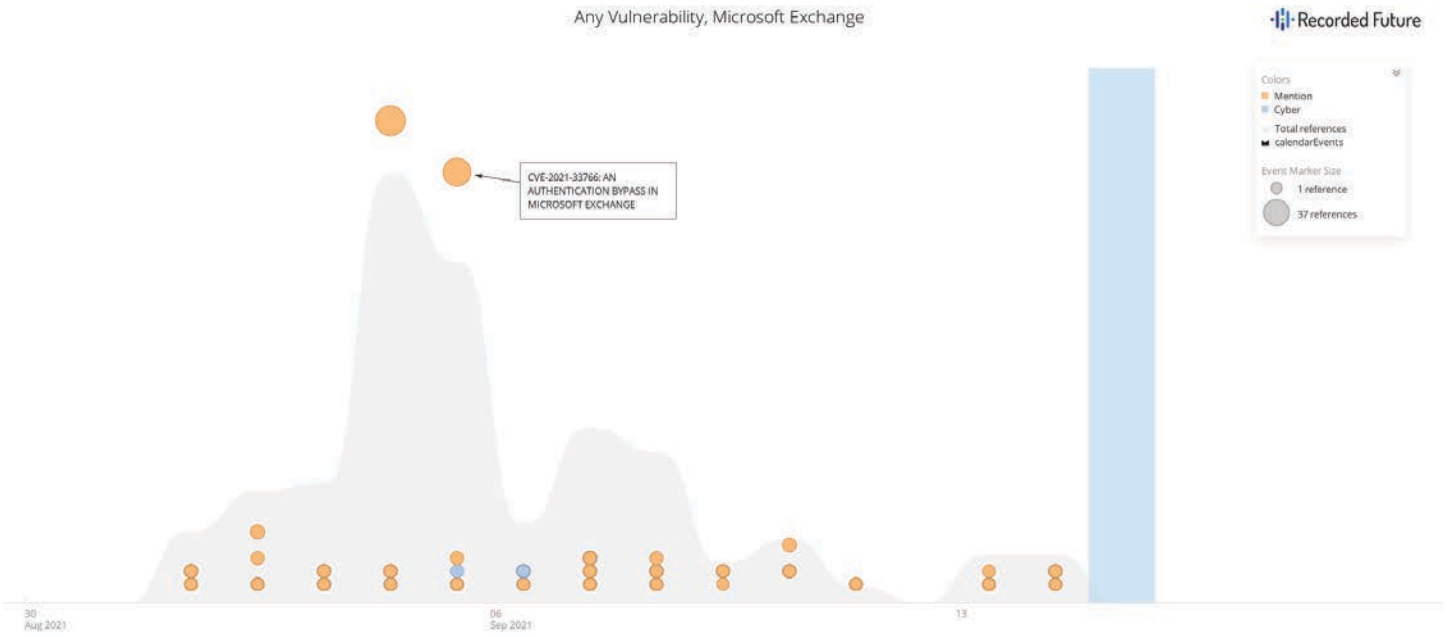

Figure 16: Vulnerabilities associated with Microsoft Exchange Servers (Source: Recorded Future)

Figure 17: Alfa Team Shell obfuscated versus deobfuscated (Source: Recorded Future)

Mitigations

There are many mitigation strategies to detect and prevent compromises of web servers involving web shells. As always, security strategies should be based upon specific requirements of the organization, but the following include some general concepts to mitigate this threat.

- **Least Privilege —** Limit privileges on the web server by controlling permissions for creation and execution of files in critical directories. By reducing privileges on the web server, defenders can reduce the attacker's ability to escalate privileges locally or move laterally.
- **Input Validation —** Use input validation to limit the ability of an attacker to conduct local and remote file inclusion.
- **File Integrity/EDR —** Use established offline backups to create "known good" versions of a server, and maintain a change-management process to monitor system file changes. Use EDR software to protect web servers.
- ***Logging* —** Visibility into what is occurring on the server is key. Ensure that logging with tools such as Sysmon on Windows servers or AuditD on Linux is occurring and the logs are being monitored via a SIEM or other such tool so that alerting can be appropriately seen and responded to. Without visibility, it is difficult or impossible to prevent a web shell attack as it is happening.
- **Patching —** Keep servers patched to prevent compromises enabled by known vulnerabilities. Attackers know that production servers such as web servers may have a delayed patching schedule. Prioritizing patches and updates for such devices is critical.
- **Traffic Monitoring —** Use an intrusion prevention system (IPS) and web application firewall (WAF), which may not detect novel attacks but can stop known attacks.

Many references provide additional guidance on web shell defense from sources such as the US National Security Agency, Cybersecurity and Infrastructure Agency, and Microsoft.

Outlook

Insikt Group expects threat actors to continue using web shells as a component of their intrusions, primarily to enable initial post-compromise actions, persistence, reconnaissance, or the dropping of additional tools. The 5 web shells evaluated during this research represent a subset of those that will continue to be used, and additional variants will likely be developed in the future. While protecting an organization against the threat of web shells cannot be done in a completely foolproof manner, using the detection strategies outlined in this report, along with regular monitoring of web server logs, patching of vulnerabilities, and other defensive tactics can help combat the threat.

About Recorded Future

Recorded Future is the world's largest provider of intelligence for enterprise security. By combining persistent and pervasive automated data collection and analytics with human analysis, Recorded Future delivers intelligence that is timely, accurate, and actionable. In a world of ever-increasing chaos and uncertainty, Recorded Future empowers organizations with the visibility they need to identify and detect threats faster; take proactive action to disrupt adversaries; and protect their people, systems, and assets, so business can be conducted with confidence. Recorded Future is trusted by more than 1,000 businesses and government organizations around the world.

Learn more at recordedfuture.com and follow us on Twitter at @RecordedFuture.