Detect and Prevent Web Shell Malware

Summary

Cyber actors have increased the use of web shell malware for computer network exploitation [1][2][3][4]. Web shell malware is software deployed by a hacker, usually on a victim's web server. It can be used to execute arbitrary system commands, which are commonly sent over HTTP or HTTPS. Web shell attacks pose a serious risk to DoD components. Attackers often create web shells by adding or modifying a file in an existing web application. Web shells provide attackers with persistent access to a compromised network using communication channels disguised to blend in with legitimate traffic. Web shell malware is a long-standing, pervasive threat that continues to evade many security tools.

Cyber actors deploy web shells by exploiting web application vulnerabilities or uploading to otherwise compromised systems. Web shells can serve as persistent backdoors or as relay nodes to route attacker commands to other systems. Attackers frequently chain together web shells on multiple compromised systems to route traffic across networks, such as from internet-facing systems to internal networks [5].

It is a common misperception that only internet-facing systems are targeted for web shells. Attackers frequently deploy web shells on non-internet facing web servers, such as internal content management systems or network device management interfaces. Internal web applications are often more susceptible to compromise due to lagging patch management or permissive security requirements.

Though the term "web shells" is predominantly associated with malware, it can also refer to web-based system management tools used legitimately by administrators. While not the focus of this guidance, these benign web shells may pose a danger to organizations as weaknesses in these tools can result in system compromise. Administrators should use system management software leveraging enterprise authentication methods, secure communication channels, and security hardening.

Mitigating Actions (DETECTION)

Web shells are difficult to detect as they are easily modified by attackers and often employ encryption, encoding, and obfuscation. A defense-in-depth approach using multiple detection capabilities is most likely to discover web shell malware. Detection methods for web shells may falsely flag benign files. When a potential web shell is detected, administrators should validate the file's origin and authenticity. Detection techniques include:

"Known-Good" Comparison

Web shells primarily target existing web applications and rely on creating or modifying files. The best method of detecting these web shells is to compare a verified benign version of the web application (i.e., a "known-good") against the production version. Discrepancies should be manually reviewed for authenticity. Additional information and scripts to enable known-good comparison are available in **Appendix A** and are maintained on https://github.com/nsacyber/Mitigating-Web-Shells.

When adjudicating discrepancies with a known-good image, administrators are cautioned against trusting timestamps on suspicious systems. Some attackers use a technique known as "timestomping" [6] to alter *created* and *modified* times in order to add legitimacy to web shell files. Administrators should not assume that a modification is authentic simply because it appears to have occurred during a maintenance period. However, as an initial triage method, administrators may choose to prioritize verification of files with unusual timestamps.

Web Traffic Anomaly Detection

While attackers often design web shells to blend in with normal web traffic, some characteristics are difficult to imitate without advanced knowledge. These characteristics include user agent strings and client Internet Protocol (IP) address space. Prior to having a presence on a network, attackers are unlikely to know which user agents or IP addresses are

typical for a web server, so web shell requests will appear anomalous. In addition, web shells routing attacker traffic will default to the web server's user agent and IP address, which should be unusual in network traffic. Uniform Resource Identifiers (URIs) exclusively accessed by anomalous user agents are potentially web shells. Finally, some attackers neglect to disguise web shell request "referer [sic] headers"[1] as normal traffic. Consequently, requests with missing or unusual referer headers could indicate web shell presence. Centralized log-querying capabilities, such as Security Information and Event Management (SIEM) systems, provide a means to implement this analytic. If such a capability is not available, administrators may use scripting to parse web server logs to identify possible web shell URIs. Example Splunk®[2] queries (**Appendix B**), scripts for analyzing log data (**Appendix C**), and additional information about detecting web traffic anomalies are maintained at https://github.com/nsacyber/Mitigating-Web-Shells.

Signature-Based Detection

From the host perspective, signature-based detection is unreliable because web shells may be obfuscated and are easy to modify. However, some cyber actors use popular web shells (e.g., China Chopper, WSO, C99, B374K, R57) with minimal modification. In these cases, fingerprint or expression-based detection may be possible. A collection of Snort®[3] rules to detect common web shell files, scanning instructions, and additional information about signature-based detection are maintained at https://github.com/nsacyber/Mitigating-Web-Shells.

From the network perspective, signature-based detection of web shells is unreliable because web shell communications are frequently obfuscated or encrypted. Additionally, "hard-coded" values like variable names are easily modified to further evade detection. While unlikely to discover unknown web shells, signature-based network detection can help identify additional infections of a known web shell. **Appendix D** provides a collection of signatures to detect network communication from common, unmodified or slightly modified web shells sometimes deployed by attackers. This list is also maintained at https://github.com/nsacyber/Mitigating-Web-Shells.

Unexpected Network Flows

In some cases, attackers use web shells on systems other than web servers (e.g., workstations). These web shells operate on rogue web server applications and can evade file-based detection by running exclusively in memory (i.e., fileless execution). While functionally similar to a traditional Remote Access Tool (RAT), these types of web shells allow attackers to easily chain malicious traffic through a uniform platform. These types of web shells can be detected on well-managed networks because they listen and respond on previously unused ports. Additionally, if an attacker is using a perimeter web server to tunnel traffic into a network, connections would be made from a perimeter device to an internal node. If administrators know which nodes on their network are acting as web servers, then network analysis can reveal these types of unexpected flows. A variety of tools including vulnerability scanners (e.g., Nessus®[4]), intrusion detection systems (e.g., Snort®), and network security monitors (e.g., Zeek™[5] [formerly "Bro"]) can reveal the presence of unauthorized web servers in a network. Maintaining a thorough and accurate depiction of expected network activity can enhance defenses against many types of attack. The Snort® rule in **Appendix E** and maintained at https://github.com/nsacyber/Mitigating-Web-Shells can be tailored for a specific network to identify unexpected network flows.

Endpoint Detection and Response (EDR) Capabilities

Some EDR and enhanced host logging solutions may be able to detect web shells based on system call or process lineage abnormalities. These security products monitor each process on the endpoint including invoked system calls. Web shells usually cause the web server process to exhibit unusual behavior. For instance, it is uncommon for most benign web servers to launch the ipconfig utility, but this is a common reconnaissance technique enabled by web shells. EDRs have different automated capabilities and querying interfaces, so organizations are encouraged to review documentation or discuss web shell detection with the vendor. **Appendix F** illustrates how Sysmon's enhanced process logging data can

---

[1] "Referer" is an HTTP header specified in Internet Engineering Task Force RFC 7231

[2] Splunk is a registered trademark of Splunk, Inc.

[3] Snort is a registered trademark of Cisco Technologies, Inc.

[4] Nessus is a registered trademark of Tenable Network Security, Inc.

[5] Zeek is a trademark of the Zeek Project

be used to identify process abnormalities in a Microsoft® Windows®[6] environment. Similarly, **Appendix G** illustrates how auditd can be used to identify process abnormalities in a Linux®[7] environment. Guidance for these identifying process abnormalities in these environments is also maintained at https://github.com/nsacyber/Mitigating-Web-Shells.

Other Anomalous Network Traffic Indicators

Web shell traffic may exhibit other detectable abnormal characteristics depending on the attacker. In particular, unusually large responses (possible data exfiltration), recurring off-peak access times (possible non-local work schedule), and geographically disparate requests (possible foreign operator) could indicate URIs of potential web shells. However, these characteristics are highly subjective and likely to flag many benign URIs. Administrators may choose to implement these detection analytics if the baseline characteristic is uniform for their environment.

Mitigating Actions (PREVENTION)

Preventing web shells should be a priority for both internet-facing and internal web servers. Good cyber hygiene and a defense-in-depth approach based on the mitigations below provide significant hardening against web shells. Prevention techniques include:

Web Application Update Prioritization

Attackers sometimes target vulnerabilities in internet-facing and internal web applications within 24 hours of a patch release. Update these applications as soon as patches are available. Whenever possible, enable automatic updating and configure frequent update cadence (at least daily). Deploy manual updates on a frequent basis when automatic updating is not possible. **Appendix H** lists some commonly exploited vulnerabilities.

Web Application Permissions

Web services should follow the least privilege security paradigm. In particular, web applications should not have permission to write directly to a web accessible directory or modify web accessible code. Attackers are unable to upload a web shell to a vulnerable application if the web server blocks access to the web accessible directory. To preserve functionality, some web applications require configuration changes to save uploads to a non-web accessible area. Prior to implementing this mitigation, consult documentation or discuss changes with the web application vendor.

File Integrity Monitoring

If administrators are unable to harden web application permissions as described above, file integrity monitoring can achieve a similar effect. File integrity software can block file changes to web accessible directories or alert when changes occur. Additionally, monitoring software has the benefit of allowing certain file changes but blocking others. For example, if an internal web application handles only Portable Document Format (PDF) files, integrity monitoring can block uploads without a ".pdf" extension. **Appendix I** provides a set of Host Intrusion Prevention System (HIPS) rules for use with McAfee®[8] Host Based Security System (HBSS) to enforce file integrity on web accessible directories. These rules, implementation instructions, and additional information about file integrity monitoring are maintained at https://github.com/nsacyber/Mitigating-Web-Shells.

Intrusion Prevention

Intrusion Prevention Systems (IPS) and Web Application Firewalls (WAF) each add a layer of defense for web applications by blocking some known attacks. Organizations should implement these appliances to block known malicious uploads. If possible, administrators are encouraged to implement the OWASP™[9] Core Rule Set, which includes patterns for blocking certain malicious uploads. As with any signature-based blocking, attackers will find ways to evade detection,

---

[6] Microsoft and Windows are registered trademarks of the Microsoft Corporation

[7] Linux is a registered trademark of the Linux Foundation

[8] McAfee is a registered trademark of McAfee, LLC

[9] OWASP is a trademark of the OWASP Foundation

so this approach is only one part of a defense-in-depth strategy. Note that IPS and WAF appliances may block the initial compromise but are unlikely to detect web shell traffic.

To maximize protection, security appliances should be tailored to individual web applications rather than using a single solution across all web servers. For instance, a security appliance configured for an organization's content management system can include application specific rules to harden targeted weaknesses that should not apply to other web applications. Additionally, security appliances should receive updates to enable real time mitigations for emerging threats.

Network Segregation

Network segregation is a complex architectural challenge that can have significant benefits when done correctly. Network segregation hinders web shell propagation by preventing connections between unrelated network segments. The simplest form of network segregation is isolating a demilitarized zone (DMZ) subnet to quarantine internet-facing servers. Advanced forms of network segregation use software-defined networking (SDN) to enable a Zero Trust[10] architecture, which requires explicit authorization for communication between nodes. While web shells could still affect a targeted server, network segmentation prevents attackers from chaining web shells to reach deeper into an organization's network. For additional information about network segregation, see *Segregate Networks and Functions* [7] on nsa.gov.

Harden Web Servers

Secure configuration of web servers and web applications can prevent web shells and other compromises. Administrators should block access to unused ports or services. Employed services should be restricted to expected clients if possible. Additionally, routine vulnerability scans can help to identify unknown weaknesses in an environment. Some host-based security systems provide advanced features, such as machine learning and file reputation, which provide some protection against web shells. Organizations should take advantage of these advanced security features when possible.

Mitigating Actions (RESPONSE and RECOVERY)

While some web shells do not persist, running entirely from memory, and others exist only as binaries or scripts in a web directory, still others can be deeply rooted with sophisticated persistence mechanisms. Regardless, they may be part of a much larger intrusion campaign. A critical focus once a web shell is discovered should be on how far the attacker penetrated within the network. Packet capture (PCAP) and network flow data can help to determine if the web shell was being used to pivot within the network, and to where. If such a pivot is cleaned up without discovering the full extent of the intrusion and evicting the attacker, that access may be regained through other channels either immediately or at a later time.

[10] Zero Trust is a model where both internal and external resources are treated as potentially malicious and thus each system verifies all access

Appendix A: Scripts to Compare a Production Website to a Known-Good Image

The scripts below can be used to compare the directory of an active website against a known-good image of that site. This script requires file level access to both the production site and the known-good image, so it should be run on the web server hosting the site or on a connected system that has a mapped drive to the web server. The script should be run with sufficient privileges to read the files in both directories. Alternatively, for Windows systems, Microsoft® developed the WinDiff utility (available at https://support.microsoft.com/en-us/help/159214/how-to-use-the-windiff-exe-utility), which allows directory comparison using a Graphical User Interface (GUI).

MICROSOFT® POWERSHELL®[11]

USAGE: *.\dirChecker.ps1 -knownGood <known-good image path> -productionImage <production image path>*

SCRIPT:

```
<#
.DESCRIPTION
The script looks for files changes/additions between a production directory (target) and a known-good directory.

.PARAMETER knownGood
Path of the known-good directory.

.PARAMETER productionImage
Path of the production directory (target).

-- Output --
File analysis started.
Any file listed below is a new or changed file.

C:\inetput\wwwroot\index2.aspx

File analysis completed.
#>
param (
    [Parameter(Mandatory=$TRUE)][ValidateScript({Test-Path $_ -PathType 'Container'})][String] $knownGood,
    [Parameter(Mandatory=$TRUE)][ValidateScript({Test-Path $_ -PathType 'Container'})][String] $productionImage
)

# Recursevely get all files in both directories, for each file calculate hash.
$good = Get-ChildItem -Force -Recurse -Path $knownGood | ForEach-Object { Get-FileHash -Path $_.FullName }
$prod = Get-ChildItem -Force -Recurse -Path $productionImage | ForEach-Object { Get-FileHash -Path $_.FullName }

Write-Host "File analysis started."
Write-Host "Any file listed below is a new or changed file.`n"

# Compare files hashes, select new or changed files, and print the path+filename.
(Compare-Object $good $prod -Property hash -PassThru | Where-Object{$_.SideIndicator -eq '=>'}).Path

Write-Host "`nFile analysis completed."
```

LINUX® DIFF UTILITY

USAGE: *diff -r -q <known-good image path> <production image path>*

CMD:

```
diff -r -q /path/to/good/image /path/to/production/site
```

[11] PowerShell is a registered trademark of Microsoft Corporation

Appendix B: Splunk® Queries for Detecting Anomalous URIs in Web Traffic

Prior to having a presence on the network, attackers are unlikely to be able to disguise web shell traffic as typical traffic for a targeted web server. In these cases, requests to the web shell are likely to have an unusual user agent string. In some environments, the attacker's IP address may also appear uncharacteristic for typical network traffic. The queries below can highlight URIs requested by unusual user agents and client IP addresses. Administrators are encouraged to tailor these queries to individual environments including targeting individual web applications or servers rather than running the query for an entire network. In rare cases, certain web applications may generate unique URIs per request which would limit the effectiveness of these queries.

SPLUNK® QUERY TO IDENTIFY URIS ACCESSED BY FEW USER AGENTS AND IP ADDRESSES

RATIONALE	*Unlike benign URIs, web shell URIs are likely to have few user agents*
QUERY (APACHE®[12])	sourcetype="access_combined" | fillnull value=- 'comment("Fill all empty fields with -")' | search status>="200" status <"300" uri!=- clientip!=- `comment("Only successful codes 200-299, eliminate blank URIs and client IPs")` | stats min(_time) as start max(_time) as stop dc(useragent) as dc_user_agent values(useragent) as values_user_agent dc(clientip) as dc_src values(clientip) as values_src count by uri `comment("Find first and last time the grouping was found, number of distinct User Agent strings and IP addresses used to access that URI")` | convert ctime(start) ctime(stop) `comment("Convert the times to a readable format")` | search dc_src<=5 OR dc_user_agent<=5 `comment("Only URIs with <=5 unique user agents or IP addresses")` | table start stop uri dc_user_agent values_user_agent dc_src values_src
QUERY (IIS™[13])	sourcetype="iis" | fillnull value=- 'comment("Fill all empty fields with -")' | search sc_status>="200" sc_status <"300" cs_uri_stem!=- c_ip!=- `comment("Only successful codes 200-299, eliminate blank URIs and client IPs")` | stats min(_time) as start max(_time) as stop dc(cs_User_Agent) as dc_user_agent values(cs_User_Agent) as values_user_agent dc(c_ip) as dc_src values(c_ip) as values_src count by cs_uri_stem `comment("Find first and last time the grouping was found, number of distinct User Agent strings and IP addresses used to access that URI")` | convert ctime(start) ctime(stop) `comment("Convert the times to a readable format")` | search dc_src<=5 OR dc_user_agent<=5 `comment("Only URIs with <=5 unique user agents or IP addresses")` | table start stop cs_uri_stem dc_user_agent values_user_agent dc_src values_src

SPLUNK® QUERY TO IDENTIFY USER AGENTS UNCOMMON FOR A TARGET WEB SERVER

RATIONALE	*Particularly for internal web applications, uncommon user agents can indicate web shell activity*
QUERY (APACHE®)	sourcetype="access_combined" | fillnull value=- 'comment("Fill all empty fields with -")' | search status>="200" status <"300" ` comment("Only successful codes 200-299")` | stats count by useragent `comment("Group User Agent strings to determine frequency")` | sort + count `comment("Sort count in ascending order")` | head 10 `comment("Limit results to top 10. This can be changed to see more or fewer results")`
QUERY (IIS™)	sourcetype="iis" sc_status>="200" AND sc_status<"300" ` comment("Only successful codes 200-299")` | fillnull value=- 'comment("Fill all empty fields with -")' | search sc_status>="200" sc_status <"300" `comment("Only successful codes 200-299")` | stats count by cs_User_Agent `comment("Group User Agent strings to determine frequency")` | sort + count `comment("Sort count in ascending order")` | head 10 `comment("Limit results to top 10. This can be changed to see more or fewer results")`

[12] Apache is a registered trademark of the Apache Software Foundation

[13] Internet Information Services (IIS) is a trademark of the Microsoft Corporation

SPLUNK® QUERY TO IDENTIFY URIS WITH AN UNCOMMON HTTP REFERER

RATIONALE	*Web shell URIs are likely to have uncommon HTTP referers*
QUERY (APACHE®)	sourcetype="access_combined" | fillnull value=- 'comment("Fill all empty fields with - (needed to make blank referer fields searchable")' | search status>="200" status <"300" `comment("Only successful codes 200-299")` | stats dc(uri) as dc_URIs values(uri) as All_URIs count by referer `comment("Counts number of times each URI request is associated with a unique referer")` | table referer, All_URIs, dc_URIs | sort + dc_URIs `comment("Sort count in ascending order")` | head 10 `comment("Limit results to top 10. This can be changed to see more or fewer results")`
QUERY (IIS™)	sourcetype=" iis" | fillnull value=- 'comment("Fill all empty fields with - (needed to make blank referer fields searchable")' | search sc_status>="200" sc_status<"300" `comment("Only successful codes 200-299")` | stats dc(cs_uri_stem) as dc_URIs values(cs_uri_stem) as All_URIs count by cs_Referer `comment("Counts number of times each URI request is associated with a unique referer")` | table cs_Referer, All_URIs, dc_URIs | sort + dc_URIs `comment("Sort count in ascending order")` | head 10 `comment("Limit results to top 10. This can be changed to see more or fewer results")`

SPLUNK® QUERY TO IDENTIFY URIS MISSING AN HTTP REFERER

RATIONALE	*Web shell URIs are likely to have missing HTTP referrers*
QUERY (APACHE®)	sourcetype="access_combined" | fillnull value=- 'comment("Fill all empty fields with - (needed to make blank referer fields searchable")' | search status>="200" status<"300" referrer=- uri!="/" `comment("Only successful codes 200-299 and blank referrer not from root webpage") | stats count by referer, uri `comment("Counts number of times each URI request is associated with a unique referer")` | table uri, count | sort - count `comment("Sort count in descending order")` | head 10 `comment("Limit results to top 10. This can be changed to add more or fewer results")`
QUERY (IIS™)	sourcetype="iis" | fillnull value=- 'comment("Fill all empty fields with - (needed to make blank referer fields searchable")' | search sc_status>="200" sc_status<"300" sc_Referer=- cs_uri_stem!="/" `comment("Only looking for successful status codes 200-299 and blank referer not from the root webpage")` | stats count by cs_Referer, cs_uri_stem `comment("Counts number of times each URI request is associated with a unique referer")` | table cs_uri_stem, count | sort - count `comment("Sort count in descending order")` | head 10 `comment("Limit results to top 10. This can be changed to add more or fewer results")`

Appendix C: Internet Information Services™ (IIS) Log Analysis Tool

Prior to having a presence on the network, attackers are unlikely to be able to disguise web shell traffic as typical traffic for a targeted web server. In these cases, requests to the web shell are likely to have an unusual user agent string. In some environments, the attacker's IP address may also appear uncharacteristic for typical network traffic. The PowerShell and Python scripts below can highlight URIs requested by unusual user agents and client IP addresses. In rare cases, certain web applications may generate unique URIs per request, which would limit the effectiveness of these queries.

MICROSOFT® POWERSHELL® SCRIPT TO ANALYZE IIS™ LOGS

USAGE

```
.\LogCheck.ps1 -logDir <path to IIS log directory>
```

SCRIPT

```
#Default parameters
Param  (
    [ValidateScript({Test-Path $_ -PathType 'Container'})][string]$logDir = "C:\inetpub\logs\",
    [ValidateRange(1,100)][int]$percentile = 5
)

If ($ExecutionContext.SessionState.LanguageMode -eq "ConstrainedLanguage")
   { Throw "Use Full Language Mode (https://devblogs.microsoft.com/powershell/powershell-constrained-language-
mode/)" }

function analyzeLogs ( $field ) {
   $URIs = @{}
   $files = Get-ChildItem -Path $logDir -File -Recurse
   If ($files.Length -eq 0)  { "No log files at the given location `n$($_)"; Exit }

   #Parse each file for relevant data. If data not present, continue to next file
   $files | Foreach-Object {
      Try {
         $file = New-Object System.IO.StreamReader -Arg $_.FullName
         $Cols = @()
         While ($line = $file.ReadLine()) {
            If ($line -like "#F*") {
               $Cols = getHeaders($line)
            } ElseIf ($Cols.Length -gt 0 -and $line -notlike "#*" ) {
               $req = $line | ConvertFrom-Csv -Header $Cols  -Delimiter ' '
               If ( IrrelevantRequest $req ) { Continue; }
               #If target field seen for this URI, update our data; otherwise create data object for this URI/field
               If ($URIs.ContainsKey($req.uri) -and $URIs[ $req.uri ].ContainsKey($req.$field) )
                  { $URIs[ $req.uri ].Set_Item( $req.$field, $URIs[ $req.uri ][ $req.$field ] + 1 ) }
               ElseIf ($URIs.ContainsKey($req.uri))
                  { $URIs[ $req.uri ].Add( $req.$field, 1 ) }
               Else
                  { $URIs.Add($req.uri, @{ $($req.$field) = 1 }) }
            }
         }
         $file.close()
      } Catch {
         Echo "Unable to parse log file $($_.FullName)`n$($_)"
      }
   }

   Echo "These URIs are suspicious because they have the least number of $($field)s requesting them:"
   $nth_index = [math]::ceiling( ($URIs.Count) * ([decimal]$percentile / 100))

   #Count the unique fields for each URI
   ForEach ($key in $($uris.keys)) { $uris.Set_Item( $key, $uris.$key.Count) }

   $i = 0;
   $URIs.GetEnumerator() | sort Value | Foreach-Object {
      $i++
```

```
        If($i -gt $nth_index) { Break; }
        Echo "   $($_.Name) is requested by $($_.Value) $($field)(s)"
    }
}

Function getHeaders ( $s ) {
    $s = (($s.TrimEnd()) -replace "#Fields: ", "" -replace "-","" -replace "\(","" -replace "\)","")
    $s = $s -replace "scstatus","status" -replace "csuristem","uri" -replace "csUserAgent","agent" -replace "cip","ip"
    Return $s.Split(' ')
}

Function IrrelevantRequest ( $req ) {
    #Skip requests missing required fields
    ForEach ($val in @("status", "uri","agent","ip"))
        { If ($val -notin $req.PSobject.Properties.Name) { Return $True} }
    #We only care about requests where the server returned success (codes 200-299)
    If ($req.status -lt 200 -or $req.scstatus -gt 299)
        { Return $True }
    Return $False
}

analyzeLogs "agent"
analyzeLogs "ip"
```

PYTHON®[14] SCRIPT TO ANALYZE APACHE® LOGS

USAGE: *./LogCheck.py <path to Apache log file>*

CMD:

```
import sys
import os.path
import csv
# Script will generate a list of URLs from Apache web access log that have least unique IP address or unique user-agents
# Written for Python 3

urlpercentage = 0.05 # Bottom Percentile of URLs to display
weblogfileName = None
apachelogsfields = ['ip', 'identd', 'frank', 'time_part0', 'time_part1', 'request', 'status', 'size', 'referer', 'user_agent']
def analyze_weblog(filename): # function output the url based on low unique ip address and low unique user-agents
    uniqueurlcount = 0                  # count of unique URL in web log
    urls = []                           # list of unique URL, also index into lists of lists of unique ip address and user-agents
    uniqueipcount = []                  # list of unique ip address count for URL
    uniqueuseragentscount = []          # list of unique use agents for URL
    iplist = []                         # list of list of ip address per unique URL to keep track of unique URL
    useragentlist = []                  # list of list of user-agents per unique URL to keep track of unique user-agents

    print("The weblog file to analyze is %s" % filename)
    with open(filename, mode='r') as csv_file:                  # read in web log as csv file
        csv_reader = csv.reader(csv_file, delimiter=' ')
        for row in csv_reader:
            if (row[0][0] != '#'):                  # handles simple case where file has comments start with "# "
                ipaddress = row[apachelogsfields.index('ip')]       # ip address
                request = row[apachelogsfields.index('request')]    # request (URL part of request)
                status = row[apachelogsfields.index('status')]      # user-agent
                user_agent = row[apachelogsfields.index('user_agent')]
                url = (request.partition(' ')[2]).partition(' ')[0]  # extract URL from request field
                if (status >= '200' and status <= '299'):           # only request with status of 200 - 299
                    if (url not in urls):                   # determine if URL is already been seen
```

[14] Python is a registered trademark of the Python Software Foundation

```
                uniqueurlcount += 1                    # if not increment unique URL count
                urls.append(url)                       # append new URL to the unique URL list
                uniqueipcount.append(0)                # append an element of zero for the unique ip count list
                uniqueuseragentscount.append(0)        # append an element of zero for the unique user-agents count list
                newiplist = []                         # new empty element list for ip address tracking per URL
                iplist.append(newiplist)               # append empty list to list of list of ip per URL
                newuseragentlist = []                  # new empty element list for user-agents tracking per URL
                useragentlist.append(newuseragentlist)  # append empty list to list of user-agents per URL
            if (user_agent not in useragentlist[urls.index(url)]):  # determine if user-agents is in the particular URL list
                useragentlist[urls.index(url)].append(user_agent)  # if not append to user-agents list for the URL list
                temp = uniqueuseragentscount[urls.index(url)] + 1  # also increment unique user-agents count
                uniqueuseragentscount[urls.index(url)] = temp
            if (ipaddress not in iplist[urls.index(url)]):          # determine if ip address is in the particular URL list
                iplist[urls.index(url)].append(ipaddress)          # if not append ip address to list for the particular URL list
                temp = uniqueipcount[urls.index(url)] + 1          # also increment unique ip address count for that URL
                uniqueipcount[urls.index(url)] = temp

    numberofurltodisplay = urlpercentage * uniqueurlcount      # Determine line that represents percentile desired
    intnumberofurltodisplay = int(numberofurltodisplay)
    if (numberofurltodisplay > intnumberofurltodisplay):       # Round up
        intnumberofurltodisplay += 1
    tempuniqueuseragentscount = uniqueuseragentscount.copy()   # temp copy of unique user-agents count to sort
    tempuniqueuseragentscount.sort()
    useragentcounttodisplay = tempuniqueuseragentscount[(intnumberofurltodisplay -1)] # determine count to display
    tempuniqueipcount = uniqueipcount.copy()                   # Create a temporary copy unique ip address count to sort
    tempuniqueipcount.sort()
    ipcounttodisplay = tempuniqueipcount[(intnumberofurltodisplay -1)]          # determine the count to display

    print(----------'URL with least user agents----------')
    for count in range (0, (useragentcounttodisplay + 1)):  # Increment unique user-agents
        index = 0
        for elementuseragentcount in uniqueuseragentscount:       # Increment thru unique user-agents count list
            if (elementuseragentcount == count):                   #   List URL where user-agents is equal to count
                print(urls[index])
            index += 1
    print(----------'URL with least user agents----------')
    for count in range (0, (ipcounttodisplay + 1)):   # Increment count to count of unique ip
        index = 0
        for elementipcount in uniqueipcount:                   # Increment thru unique ip address count list
            if (elementipcount == count):                      #   List URL where user-agents is equal to count
                print(urls[index])
            index += 1

if __name__ == '__main__':
  try:
    if len(sys.argv) == 2:                                 # Simple check if an argument is passed (assume weblog file)
      weblogfileName=sys.argv[1]
      print ("Web log file to read is %s" % weblogfileName)
      if(os.path.isfile(weblogfileName)):
          analyze_weblog(weblogfileName)
    else:
      print ('Usage: python3 %s <weblogfile>' % sys.argv[0])       # Print usage statement
  except Exception as e:
    print("You must provide a valid filename (path) of a web logfile")
    raise
```

Appendix D: Network Signatures of Traffic for Common Web Shells

Web shell traffic is often obfuscated or encrypted. If organizations have inspection into Transport Layer Security (TLS) encrypted sessions for their network, such as via reverse proxy or Web Application Firewall (WAF), then the signatures in the table below may be able to identify network traffic for some common web shells that have not been significantly modified. These fingerprints are subject to change as attackers are likely to alter encoding techniques to evade these signatures. This table is not comprehensive and should be used only as part of a defense-in-depth strategy.

SNORT® RULES TO DETECT COMMON UNMODIFIED WEB SHELL MALWARE

RATIONALE	*Attackers sometimes use unmodified web shells which can be detected by network sensors*
RULES	# Be sure to put a valid SID in before implementing and test the signature for performance. # These signatures are targeted at the China Chopper web shell # Source: https://www.fireeye.com/blog/threat-research/2013/08/breaking-down-the-china-chopper-web-shell-part-ii.html alert tcp any any -> any any (msg: "China Chopper with first Command Detected"; flow:to_server,established; content: "FromBase64String"; content: "z1"; content:"POST"; nocase;http_method; reference:url,http://www.fireeye.com/blog/technical/botnet-activities-research/2013/08/breaking-down-the-china-chopper-web-shell-part-i.html; sid: 90000101;) alert tcp any any -> any any (msg: "China Chopper with all Commands Detected"; flow:to_server,established; content: "FromBase64String"; content: "z"; pcre: "/Z\d{1,3}/i"; content:"POST"; nocase;http_method; reference:url,http://www.fireeye.com/blog/technical/botnet-activities-research/2013/08/breaking-down-the-china-chopper-web-shell-part-i.html; sid: 90000102;) # These signatures are targeted at the C99 web shell # Source: https://github.com/jpalanco/alienvault-ossim/blob/master/snort-rules-default-open/rules/2.9.2/emerging.rules/emerging-web_server.rules alert tcp any any -> any any (msg:"ET WEB_SERVER c99 Shell Backdoor Var Override URI"; flow:to_server,established; content:"c99shcook["; nocase; http_uri; fast_pattern:only; pcre:"/[&?]c99shcook\[/Ui"; reference:url,thehackerblog.com/every-c99-php-shell-is-backdoored-aka-free-shells/; sid:2018601; rev:1; metadata:created_at 2014_06_24, updated_at 2014_06_24;) alert tcp any any -> any any (msg:"ET WEB_SERVER c99 Shell Backdoor Var Override Cookie"; flow:to_server,established; content:"c99shcook"; nocase; fast_pattern:only; pcre:"/c99shcook/Ci"; reference:url,thehackerblog.com/every-c99-php-shell-is-backdoored-aka-free-shells/; sid:2018602; rev:1; metadata:created_at 2014_06_24, updated_at 2014_06_24;) alert tcp any any -> any any (msg:"ET WEB_SERVER c99 Shell Backdoor Var Override Client Body"; flow:to_server,established; content:"c99shcook["; nocase; fast_pattern:only; http_client_body; pcre:"/(?:^|&)c99shcook\[/Pi"; reference:url,thehackerblog.com/every-c99-php-shell-is-backdoored-aka-free-shells/; sid:2018603; rev:1; metadata:created_at 2014_06_24, updated_at 2014_06_24;) #These signatures are targeted at the R57 web shell # Source: nsa.gov alert tcp any any -> any any (msg: "R57 Web shell Detected"; content: "<title>r57 Shell Version "; rev:1; sid: 90000201;) #These signatures are targeted at the B374k web shell # Source: nsa.gov alert tcp any any -> any any (msg: "B374k Web shell Detected"; content: "<title>b374k "; rev:1; sid: 90000301;) #These signatures are targeted at the WSO web shell # Source: nsa.gov alert tcp any any -> any any (msg: "WSO Web shell Detected"; content: "onclick=\"g('SelfRemove',null,'','')\">Self remove]"; rev:1; sid: 90000401;) # Source: https://rules.emergingthreatspro.com/9598411999529178/suricata-2.0/rules/web_server.rules alert tcp any any -> any any (msg:"ET WEB_SERVER WSO Web Shell Activity POST structure 2"; flow:established,to_server; content:"POST"; http_method; content:" name=|22|c|22|"; http_client_body; content:"name=|22|p1|22|"; http_client_body; fast_pattern; pcre:"/name=(?P<q>[\x22\x27])a(?P=q)[^\r\n]*\r\n[\r\n\s]+(?:S(?:e(?:lfRemove|cInfo)|tringTools|afeMode|ql)|(?:Bruteforc|Consol)e|FilesMan|Network|Logout|Php)/Pi"; sid:2016354; rev:2; metadata:created_at 2013_02_05, updated_at 2013_02_05;)

Appendix E: Identifying Unexpected Network Flows

The following Snort® rule can aid administrators in identifying unexpected network flows. Identifying unexpected network flows requires that administrators maintain an accurate understanding of the expected network architecture. The rule below is unlikely to be effective without tailoring it for a specific network.

SNORT® RULE TO IDENTIFY UNEXPECTED WEB SERVERS	
USAGE	*Replace "XXX.XXX.XXX.XXX/XX" with a target subnet (e.g., "192.168.1.0/24") and add the rule to Snort*
SCRIPT	alert tcp XXX.XXX.XXX.XXX/XX [443,80] -> any any (msg: "potential unexpected web server"; sid:4000921)

Appendix F: Identifying Abnormal Process Invocations in Sysmon Data

Microsoft® Sysmon is a logging tool that enhances logging performed on Windows® systems. Among other things, Sysmon logs information about how each process is created. The information is valuable for identifying anomalous behavior, such as in the case of malicious web shells. Sysmon can be obtained from Microsoft® at https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon and must be installed on a system in order to begin logging. Ideally, Sysmon and other Windows® logging should be mirrored to a central Security Information and Event Management (SIEM) server where it can be aggregated and queried.

The query below will simply report which executables were launched by an IIS™ web server. In many cases, a web application will cause IIS™ to launch a process for entirely benign functionality. However, there are several executables commonly used by attackers for reconnaissance purposes which are unlikely to be used by a normal web application. Some of these executables are listed in the table below. Administrators are encouraged to review the results of the PowerShell® query below and verify that the web application in question is intended to use the identified executables.

POWERSHELL® SCRIPT TO IDENTIFY ANOMALOUS SYSMON ENTRIES FOR IIS™

USAGE	*Run the following command from a PowerShell® prompt with administrative access*
SCRIPT	Get-WinEvent -FilterHashtable @{logname="Microsoft-Windows-Sysmon/Operational";id=1;} | Where {$_.message -like "*ParentImage: C:\Windows\System32\inetsrv\w3wp.exe*"} | %{ $_.properties[4]} | Sort-Object -Property value -Unique

Windows® environment executables frequently used by attackers and rarely launched by benign IIS™ apps			
arp.exe	hostname.exe	ntdutil.exe	schtasks.exe
at.exe	ipconfig.exe	pathping.exe	systeminfo.exe
bitsadmin.exe	nbtstat.exe	ping.exe	tasklist.exe
certutil.exe	net.exe	powershell.exe	tracert.exe
cmd.exe	net1.exe	qprocess.exe	ver.exe
dsget.exe	netdom.exe	query.exe	vssadmin.exe
dsquery.exe	netsh.exe	qwinsta.exe	wevtutil.exe
find.exe	netstat.exe	reg.exe	whoami.exe
findstr.exe	nltest.exe	rundll32.exe	wmic.exe
fsutil.exe	nslookup.exe	sc.exe	wusa.exe

Appendix G: Identifying Abnormal Process Invocations with Auditd

Auditd is the userspace component of the Linux® Auditing System. Auditd can provide users with insight into process creation logs. The information is valuable for identifying anomalous behavior, such as in the case of malicious web shells. Auditd is available in default repositories for many Linux® distributions and must be installed and configured to log relevant web server process data. Ideally, auditd and other Linux® logging should be mirrored to a central Security Information and Event Management (SIEM) server where it can be aggregated and queried.

The query below will simply report which applications were launched by an Apache® web server. In many cases, a web application will cause Apache® to launch a process for entirely benign functionality. However, there are several applications commonly used by attackers for reconnaissance purposes which are unlikely to be used by a normal web application. Some of these executables are listed in the table below. Administrators are encouraged to review the results and verify that the web application in question is intended to use the identified applications.

Configuring Auditd

1. *Determine the web server uid:*
   After installing auditd (for example using "apt -y install auditd"), determine the uid of web server using:
   apachectl -S
   This will return apache details including the user id in a line such as:
   User: name="www-data" id=33
   Here the uid is "33"

2. *Add the following auditd rules (/etc/audit/rules.d/audit.rules) replacing "XX" with the uid identified above*:
   -a always,exit -F arch=b32 -F uid=**XX** -S execve -k apacheexecve
   -a always,exit -F arch=b64 -F uid=**XX** -S execve -k apacheexecve

3. *Restart auditd*:
   service auditd restart

Review Auditd Log

1. *Applications launched by Apache® can be identified with:*
   cat /var/log/auditd/audit.* | grep "apacheexecve"
   This will return the path to the launched application (see bolded path in the example output below)

   type=SYSCALL msg=audit(1581519503.841:47): arch=c000003e syscall=59 success=yes exit=0 a0=563e412cbbd8 a1=563e412cbb60 a2=563e412cbb78 a3=7f065d5e5810 items=2 ppid=15483 pid=15484 auid=4294967295 uid=33 gid=33 euid=33 suid=33 fsuid=33 egid=33 sgid=33 fsgid=33 tty=(none) ses=4294967295 comm="cat" exe="**/bin/cat**" key="apacheexecve"

2. *Results can be analyzed to determine if unusual applications are launched (see table below)*

3. *Detailed information, including call arguments, can be obtained using:*
   cat /var/log/auditd/audit.* | grep "msg=audit(1581519503.841:47)"
   Replace the value of "msg=audit" with the value returned in step 1 above

Linux® environment applications frequently used by attackers and rarely launched by benign Apache® applications			
cat	ifconfig	ls	route
crontab	ip	netstat	uname
hostname	iptables	pwd	whoami

Appendix H: Commonly Exploited Web Application Vulnerabilities

The list below shows some web application vulnerabilities that are commonly exploited to install web shell malware. This list is not intended to be exhaustive, but it provides insight on some frequently exploited cases. Organizations are encouraged to patch both internet-facing and internal web applications rapidly to counter the risks from "n-day" vulnerabilities.

Vulnerability Identifier	Affected Application	Reported
CVE-2019-0604	Microsoft® SharePoint®[15]	15 May 2019 [8]
CVE-2019-19781	Citrix®[16] Gateway, Citrix® Application Delivery Controller, and Citrix® SD-WAN WANOP appliance	22 Jan 2020 [9]
CVE-2019-3396	Atlassian® Confluence®[17] Server	20 May 2019 [10]
CVE-2019-3398	Atlassian® Confluence Server and Atlassian® Confluence Data Center	26 Nov 2019 [11]
CVE-2019-9978	WordPress®[18] "Social Warfare" Plugin	22 Apr 2019 [12]
CVE-2019-18935 CVE-2017-11317 CVE-2017-11357	Progress® Telerik®[19] UI	7 Feb 2019 [13]
CVE-2019-11580	Atlassian® Crowd and Crowd Data Center	15 July 2019 [14]
CVE-2020-10189	Zoho® ManageEngine®[20] Desktop Central	6 Mar 2020 [15]
CVE-2019-8394	Zoho® ManageEngine® ServiceDesk Plus	18 Feb 2019 [16]
CVE-2020-0688	Microsoft® Exchange®[21] Server	10 Mar 2020 [17]
CVE-2018-15961	Adobe® ColdFusion®[22]	8 Nov 2018 [18]

[15] SharePoint is a registered trademark of the Microsoft Corporation

[16] Citrix is a registered trademark of Citrix Systems, Inc.

[17] Atlassian and Confluence are registered trademarks of Atlassian Pty Ltd.

[18] WordPress is a registered trademark of the WordPress Foundation

[19] Progress and Telerik are registered trademarks of Progress Software EAD

[20] Zoho and ManageEngine are registered trademarks of ZOHO Corporation

[21] Exchange is a registered trademark of the Microsoft Corporation

[22] Adobe and ColdFusion are registered trademarks of Adobe Systems Incorporated

Appendix I: HIPS Rules for Blocking Changes to Web Accessible Directories

McAfee® HBSS allows specification of custom HIPS rules, which are then enforced by endpoint McAfee® agents. These rules can be used to block file creation and file changes to web accessible directories effectively neutering the primary infection vector for web shell malware. If necessary, these rules can be temporarily disabled during site updates or web application patches. As with any new HIPS rule, administrators should begin enforcement at Level 1 (informational) in order to identify potential conflicts with existing applications. Enforcement should be raised to Level 4 (high) once impact assessment is deemed acceptable.

WINDOWS® ENVIRONMENT

USAGE	*Replace "C:\\inetpub\\wwwroot*" with the target directory path (i.e., the web directory)*

RULE	Rule {
	Tag "Blocking Changes to Web Directory (Windows)"
	Class Files
	ID -1 # this will select the next free ID number in the 4XXX series
	Level 1
	files { Include "C:\\inetpub\\wwwroot*" }
	directives files:rename files:permissions files:create files:write
	}

LINUX® ENVIRONMENT

USAGE	*Replace "/var/www/html/*" with the target directory path (i.e., the web directory)*

RULE	Rule {
	Tag "Blocking Changes to Web Directory (Linux)"
	Class UNIX_file
	ID -1 # this will select the next free ID number in the 4XXX series
	Level 1
	files { Include "/var/www/html/*" }
	directives unixfile:symlink unixfile:create unixfile:mkdir unixfile:write
	}

Tuning Signatures:
Signatures should be tuned according to the operating environment. If there are any exemptions (e.g., web application uploads) an exception can be created by clicking on the HIPS custom signature under Policy Catalog, Host Intrusion Prevention IPS/IPS rules and selecting the name of the IPS signature usually under "My Default".

Once on the signature page, the signature can be found by typing in the Search box key words including the name of the signature. Once the signature is displayed, the check box to the left of it should be selected and the Exception Rule tab can be clicked to add a Parameter for a specific file type (e.g., *.pdf) that should be allowed.

Works Cited

[1] Microsoft Detection and Response Team (2020), Ghost in the shell: Investigating web shell attacks. [Online] Available at: https://microsoft.com/security/blog/2020/02/04/ghost-in-the-shell-investigating-web-shell-attacks/ [Accessed Apr. 6, 2020]

[2] Rascagneres, P. and Svajcer, V. (2019). China Chopper still active 9 years later. [Online] Available at: https://blog.talosintelligence.com/2019/08/china-chopper-still-active-9-years-later.html [Accessed Apr. 6, 2020]

[3] CISA (2017), Alert TA15-314A. [Online] Available at: https://www.us-cert.gov/ncas/alerts/TA15-314A [Accessed Apr. 6, 2020]

[4] CISA (2018), Alert AA18-284A. [Online] Available at: https://www.us-cert.gov/ncas/alerts/AA18-284A [Accessed Apr. 6, 2020]

[5] ACSC (2015), Web Shells – Threat Awareness and Guidance. [Online] Available at https://cyber.gov.au/sites/default/files/2019-03/ACSC_Web_Shells.pdf [Accessed Apr. 6, 2020]

[6] Dumont, R. (2017), MITRE ATT&CK Framework - Timestomp. [Online] Available at https://attack.mitre.org/techniques/T1099/ [Accessed Apr. 6, 2020]

[7] NSA (2016), Segregate Networks and Functions. [Online] Available at https://apps.nsa.gov/iaarchive/library/ia-guidance/security-tips/segregate-networks-and-functions.cfm [Accessed Apr. 6, 2020]

[8] TrendMicro (2019), Security Alert: China Chopper Malware targeting vulnerable SharePoint servers. [Online] Available at https://success.trendmicro.com/solution/000131747 [Accessed Apr. 6, 2020]

[9] Ballenthin et al. (2020), FireEye and Citrix Tool Scans for Indicators of Compromise Related to CVE-2019-19781. [Online] Available at https://www.fireye.com/blog/products-and-services/2020/01/fireeye-and-citrix-tool-scans-for-iocs-related-to-vulnerability.html [Accessed Apr. 6, 2020]

[10] Capuano, E. (2019), Analysis of Exploitation: CVE-2019-3396. [Online] Available at https://blog.reconinfosec.com/analysis-of-exploitation-of-cve-2019-3396/ [Accessed Apr. 6, 2020]

[11] Joshi, A. (2019), CVE-2019-3398: Atlassian Confluence Download Attachments Remote Code Execution. [Online] Available at https://blogs.juniper.net/en-us/threat-research/cve-2019-3398-atlassian-confluence-download-attachments-remote-code-execution [Accessed Apr. 6, 2020]

[12] Deng, Zhang, and Gao. (2019), Exploits in the Wild for WordPress Social Warfare Plugin CVE-2019-9978. [Online] Available at https://unit42.paloaltonetworks.com/exploits-in-the-wild-for-wordpress-social-warfare-plugin-cve-2019-9978 [Accessed Apr. 6, 2020]

[13] Wulftange, M. (2019), Telerik Revisited. [Online] Available at https://codewhitesec.blogspot.com/2019/02/telerik-revisited.html [Accessed Apr. 6, 2020]

[14] Narang, S. (2019), CVE-2019-11580: Proof-of-Concept for Critical Atlassian Crowd Remote Code Execution Vulnerability Now Available. [Online] Available at https://tenable.com/blog/cve-2019-11580-proof-of-concept-for-critical-atlassian-crowd-remote-code-execution [Accessed Apr. 6, 2020]

[15] ManageEngine (2020), Identification and mitigation of remote code execution vulnerability CVE-2020-10189. [Online] Available at https://manageengine.com/products/desktop-central/rce-vulnerability-cve-2020-10189.html [Accessed Apr. 6, 2020]

[16] CISA (2019), Bulletin SB19-056. [Online] Available at https://us-cert.gov/ncas/bulletins/SB19-056 [Accessed Apr. 6, 2020]

[17] CISA (2020), Unpatched Microsoft Exchange Servers Vulnerable to CVE-2020-0688. [Online] Available at https://us-cert.gov/ncas/current-activity/2020/03/10/unpatched-microsoft-exchange-servers-vulnerable-cve-2020-0688 [Accessed Apr. 6, 2020]

[18] Volexity Threat Research (2018), Active Exploitation of Newly Patched ColdFusion Vulnerability. [Online] Available at https://volexity.com/blog/2018/11/08/active-exploitation-of-newly-patched-coldfusion-vulnerability-cve-2018-15961/ [Accessed Apr. 6, 2020]

Disclaimer of Endorsement

Contact

NSA Client Requirements / General Cybersecurity Inquiries: Cybersecurity Requirements Center, 410-854-4200, Cybersecurity_Requests@nsa.gov
NSA Media Inquiries / Press Desk: 443-634-0721, MediaRelations@nsa.gov
ASD Australian Cyber Security Centre / General Cybersecurity or Media Enquiries: asd.assist@defence.gov.au or visit www.cyber.gov.au.