Web Shell Threat Hunting with Azure Sentinel

techcommunity.microsoft.com/t5/azure-sentinel/web-shell-threat-hunting-with-azure-sentinel/ba-p/2234968

March 25, 2021

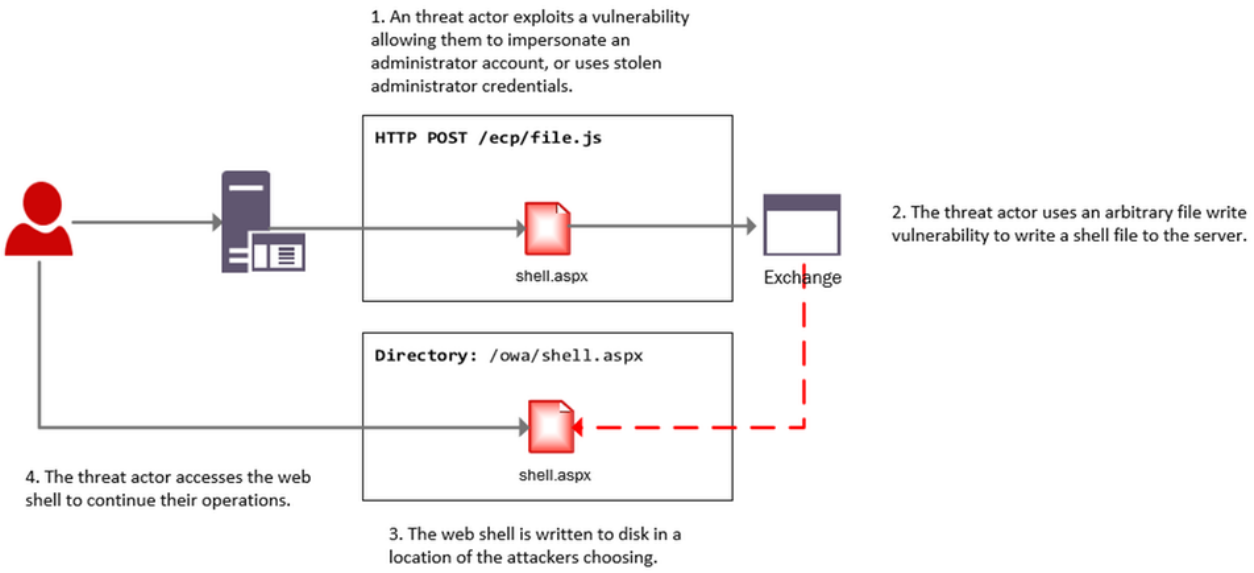
In this blog post we will provide Microsoft Azure Sentinel customers with hunting queries to investigate possible on-premises Exchange Server exploitation and identify additional attacker IOCs (Indicators of compromise) such as IP address and User Agent. These hunting techniques can also be applied to web shell techniques targeting other web applications.

The techniques we discuss below have been adapted from the June 2020 blog post: Web shell threat hunting with Azure Sentinel and Microsoft Threat Protection. The previous blog post analysed an attack against a SharePoint server, however, many of the techniques can also be applied to Exchange servers since it also uses IIS to host its web interfaces.

Recent vulnerabilities in on-premises Microsoft Exchange servers have led to deployment of web shells by threat actors. More information on these vulnerabilities can be found in this MSRC blog, details on threat actor HAFNIUM using these vulnerabilities can be found in this MSTIC blog. MSRC has also provided guidance for responders, a one-click tool for remediation and automatic remediation is delivered through Microsoft Defender for Endpoint.

Our colleagues in Microsoft Defender Threat Intelligence have authored another blog that provides additional details on use of web shells in attacks taking advantage of the Exchange Server.

The below diagram provides a high-level overview of an attacker leveraging these vulnerabilities to install a web shell on an Exchange server.



Investigating web shell alerts

Microsoft 365 Defender (M365D) detects web shell installation and execution activity. Security alerts and incidents generated by M365D can be written to the SecurityAlert table in Azure Sentinel by enabling the appropriate connector. An example of a web shell installation alert in the Azure Sentinel SecurityAlert table can be seen below.

Completed. Showing results from the last 24 hours.

TimeGenerated [UTC]	DisplayName	AlertName
3/20/2021, 1:08:24.000 AM	Possible web shell installation	Possible web shell installation

TimeGenerated [UTC]	2021-03-20T01:08:24Z
DisplayName	Possible web shell installation
AlertName	Possible web shell installation
Description	A suspicious web script was written in a folder containing a web application.
ProviderName	MDATP
ProductName	Microsoft Defender Advanced Threat Protection
Type	SecurityAlert
Entites	[{"$id":"4", "DnsDomain":"mstic-demo-test.com","HostName","test-host-0", "I

These alerts can be enriched in Azure Sentinel with new information from other log sources. When dealing with remote attacks on web application servers, one of the best enrichment sources available are the web logs that have been generated. In the case that the application server is Microsoft Exchange the W3CIISLog can be used to enrich M365D alerts with potential attacker information. Information on collecting IIS logs using the Log Analytics agent can be found here.

Identifying the Attacker IP address from Microsoft 365 Defender alerts

The query below extracts alerts from M365D where a web script file has been observed as part of the alert. In the below example, alerts containing ASP, ASPX, ASMX and ASAX files will be extracted; these are web script files commonly used by Exchange servers.

After extracting relevant web shell alerts the query will join the alert information with the W3CIIS log, this allows the query to identify any clients that have accessed the potential shell file, allowing the potential attacker to be identified. A version of the query below is already available as an Azure Sentinel detection and can be found here.

```
let timeWindow = 3d;
//Script file extensions to match on, can be expanded for your environment
let scriptExtensions = dynamic([".asp", ".aspx", ".asmx", ".asax"]);
SecurityAlert
| where TimeGenerated > ago(timeWindow)
| where ProviderName == "MDATP"
//Parse and expand the alert JSON
| extend alertData = parse_json(Entities)
| mvexpand alertData
| where alertData.Type == "file"
//This can be expanded to include more file types
| where alertData.Name has_any(scriptExtensions)
| extend FileName = tostring(alertData.Name), Directory =
tostring(alertData.Directory)
| project TimeGenerated, FileName, Directory
| join (
W3CIISLog
| where TimeGenerated > ago(timeWindow)
| where csUriStem has_any(scriptExtensions)
| extend splitUriStem = split(csUriStem, "/")
| extend FileName = splitUriStem[-1]
| summarize StartTime=min(TimeGenerated), EndTime=max(TimeGenerated) by
AttackerIP=cIP, AttackerUserAgent=csUserAgent, SiteName=sSiteName,
ShellLocation=csUriStem, tostring(FileName)
) on FileName
| project StartTime, EndTime, AttackerIP, AttackerUserAgent, SiteName, ShellLocation
```

Identifying Exchange Servers & Associated Security Alerts

Exchange servers can be challenging to identify in default log data; however using data available in W3CIISLog, Exchange servers can be identified using predictable URI strings without relying on the hostname or site name.

The query below extracts the host name from W3CIISLog where a known Exchange URI path is observed, this provides a list of hostnames that are running Exchange. This list of host names can then be used to aggregate information from the alerts in the SecurityAlert table.

```
W3CIISLog
| where csUriStem has_any("/owa/auth/", "/ecp/healthcheck.htm", "/ews/exchange.asmx")
| summarize by computer=tolower(Computer)
| join kind=leftouter (
SecurityAlert
| extend alertData = parse_json(Entities)
| mvexpand alertData
| where alertData.Type == "host"
| extend computer = iff(isnotempty(alertData.DnsDomain),
tolower(strcat(tostring(alertData.HostName), "." ,
tostring(alertData.DnsDomain))),tolower(tostring(alertData.HostName)))
| summarize Alerts=dcount(SystemAlertId), AlertTimes=make_list(TimeGenerated),
AlertNames=make_list(AlertName) by computer
) on computer
| project ExchangeServer=computer, Alerts, AlertTimes, AlertNames
```

Completed. Showing results from the last 24 hours.

ExchangeServer	Alerts...	AlertTimes	AlertNames
exchange-01.contoso.com	9	["2021-03-19T02:50:47.0440000Z","2...	["Possible web shell installation","Suspicious proc...
exchange-02.contoso.com	8	["2021-03-18T19:56:56.9860000Z","20...	["Sensitive credential memory read","Possible we...
mail-server-01.contoso.com	7	["2021-03-19T08:25:16.9150000Z","20...	["Possible web shell installation","Sensitive creden...

The results of the query provide insights into whether additional security alerts beyond web shell alerts have been observed on the host. Following deployment of a web shell it's highly likely the threat actor will begin to execute further commands on the server, triggering additional alerts. In the above example three Exchange servers were observed with security alerts.

This same technique can be used to locate other web applications within the network that use common or predictable web paths.

W3CIISLog Analysis

W3CIISLog provides detailed logging on actions performed on Microsoft Internet Information Servers (IIS). Even when an Endpoint detection alert is not available, it is possible to explore W3CIISLogs for indicators of compromise. W3CIISLog can also provide additional insights

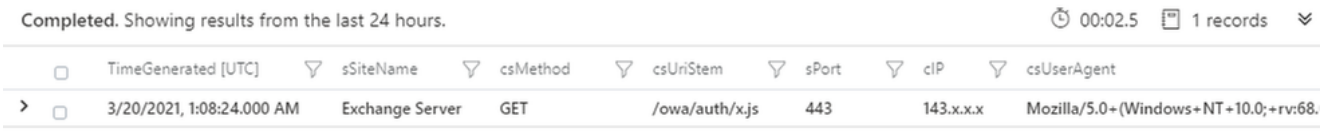
into which hosts in the network are web application servers.

Note: As part of the original Microsoft HAFNIUM blog post, several hunting and detection queries were created to search for artefacts specific to the use of recent vulnerabilities.

Identifying generic exploitation activity

If the URI associated with the vulnerable file on the server is known, a query can be constructed to identify log entries that match the URI pattern. W3CIIS logging stores the URI in the column named “csUriStem”, the below query can be used to search for a specific URI in logs and provide information on which clients have accessed them. Local IP addresses have been removed.

```
W3CIISLog
| where TimeGenerated > ago(3d)
| where not(ipv4_is_private(cIP))
//Insert potentially exploited URI here
| where csUriStem =~ "/owa/auth/x.js"
| project TimeGenerated, sSiteName, csMethod, csUriStem, sPort, cIP, csUserAgent
```

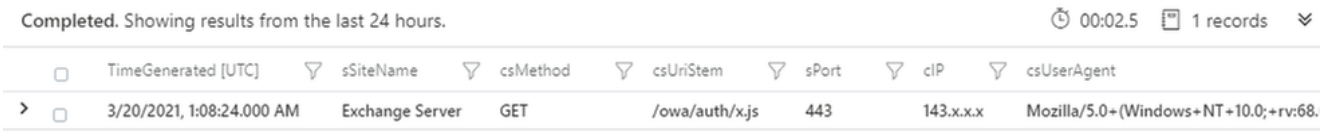


For HAFNIUM attacks observed by MSTIC an indicator feed has been made available (CSV, JSON). A detection query, that will check for the presence of indicators in multiple data sources, has also been made available by the Azure Sentinel team. The detection can be found here, and IOC’s released as feeds by MSTIC can be found in this directory.

The recent Exchange vulnerabilities do not need to be targeted at a specific file. Analysis of automated exploitation tools online shows that many randomise the filenames used; this means that no legitimate user will visit these files as they do not exist on the server. As these filenames are randomly generated, static string matching cannot be used.

The Kusto “matches_regex” function can be used to perform regular expression matching on URI’s. The below example extracts events where the URI matches files associated with the exploitation of CVE-2021-27065 from W3CIISLog.

```
W3CIISLog
| where TimeGenerated > ago(3d)
| where not(ipv4_is_private(cIP))
| where (csUriStem matches regex @"\/owa\/auth\/[A-Za-z0-9]{1,30}\.js") or (csUriStem
matches regex @"\/ecp\/[A-Za-z0-9]{1,30}\.(js|flt|css)")
| project TimeGenerated, sSiteName, csMethod, csUriStem, sPort, cIP, csUserAgent
```



The previous queries can be limited when the files being exploited are commonly accessed. They would produce many candidate attacker IP addresses, making analysis challenging.

Using the recent Exchange vulnerabilities as an example, Microsoft has seen malicious automated tools released publicly that are being used to exploit the Exchange vulnerabilities. These tools are designed to only visit specific URIs on the server that are required to perform the exploit. This activity differs from normal and legitimate Administrator or User application browsing activity and if observed should be investigated.

It is possible to craft a query that uses basic statistical analysis to identify instances where a client has visited a disproportionately high number of exploit-related URI's when compared to other URIs on the site., The query below calculates the total number of suspicious URIs that have been visited by each user, it then calculates the total number of URIs visited by the user. Where the number of exploit related URIs is a significant proportion of URIs visited, a result is returned. By default, the query requires over 90% of the URIs visited by the user to be suspicious.

```
let timeRange = 7d;
//Calculate number of suspicious URI stems visited by user
W3CIISLog
| where TimeGenerated > ago(timeRange)
| where not(ipv4_is_private(cIP))
| where (csUriStem matches regex @"\/owa\/auth\/[A-Za-z0-9]{1,30}\.js") or (csUriStem
matches regex @"\/ecp\/[A-Za-z0-9]{1,30}\.(js|flt|css)") or (csUriStem =~
"/ews/exchange.asmx")
| extend userHash = hash_md5(strcat(cIP, csUserAgent))
| summarize susCount=dcount(csUriStem), make_list(csUriStem), min(TimeGenerated),
max(TimeGenerated) by userHash, cIP, csUserAgent
| join kind=leftouter  (
//Calculate unique URI stems visited by each user
W3CIISLog
| where TimeGenerated > ago(timeRange)
| where not(ipv4_is_private(cIP))
| extend userHash = hash_md5(strcat(cIP, csUserAgent))
| summarize allCount=dcount(csUriStem) by userHash
) on userHash
//Find instances where only a common endpoint was seen
| extend containsDefault = iff(list_csUriStem contains "/ews/exchange.asmx", 1, 0)
//If we only see the common endpoint and nothing else dump it
| extend result = iff(containsDefault == 1, containsDefault+susCount, 0)
| where result != 2
| extend susPercentage = susCount / allCount * 100
| where susPercentage > 90
| project StartTime=min_TimeGenerated, EndTime=max_TimeGenerated, AttackerIP=cIP,
AttackerUA=csUserAgent, URIsVisited=list_csUriStem,
suspiciousPercentage=susPercentage, allUriCount=allCount, suspiciousUriCount=susCount
```

While this query is designed to detect recent Exchange exploit activity, it can be easily adapted to other exploit chains if the pages or URIs used are known.

Rare Client File Access

A previously published hunting query can be used to detect instances where resources on a server are requested by a single client – a behaviour that should be investigated in the context of web shell exploits. After the actor creates web shell on the server, it's likely that they will be the only user to access the file to complete their intended objective.

Investigating the Attacker

In the previous blog post covering SharePoint exploitation, a Jupyter Notebook Guided Investigation is provided. This notebook can also be used to investigate on-prem Exchange compromises within your environment.

The notebook extracts alerts from Microsoft 365 Defender related to web shell activity, these can then be enriched with information from W3CIIS to identify the attacker IP and User Agent. The attackers IP and User Agent can be used to hunt through multiple log sources for potential post-compromise activity.

After the attacker details have been identified, the notebook can be used to locate files that were accessed by the attacker prior to the web shell being installed. The notebook will also locate the first instance that the attacker visited the server.

Azure-Sentinel-Notebooks/Guided Investigation - MDE Webshell Alerts.ipynb at master · Azure/Azure-Se...

Instructions for getting the notebook up and running can be found in the original blog post, under the title "Building out the Investigation using Jupyter Notebooks".

You can stay up to date with the latest information at https://aka.ms/exchangevulns.