

Web shell threat hunting with Azure Sentinel and Microsoft Threat Protection

techcommunity.microsoft.com/t5/azure-sentinel/web-shell-threat-hunting-with-azure-sentinel-and-microsoft/ba-p/1448065

June 9, 2020

Possible web shell installation
This alert is part of incident (3)

Actions

Severity: Medium
Category: Persistence
Technique: T1100: Web
Detection source: EDR
Detection technology: Behavioral

Attack Summary

Attacker IP: 51. ...

Attacker user agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.3; WOW64; Trident/7.0; .NET4.0E; .NET4.0C; .NET CLR 3.5.30729; .NET CLR 2.0.50727; .NET CLR 3.0.30729)

Webshell installed: /_layouts/15/fonts.aspx

Victim site: SharePoint - 33174

TimeGenerated [UTC]	DisplayName	AlertName
5/5/2020, 5:36:55.000 PM	Possible web shell installation	Possible web shell in
...		
2020-05-05T17:36:55Z	Possible web shell installation	Possible web shell installation
		web script was written in a folder containing a
		ender Advanced Threat Protection

Tom McElroy, Rob Mead – Microsoft Threat Intelligence Center

In this blog we use Azure Sentinel to enrich the investigation of endpoint web shell alerts from Microsoft Defender Advanced Threat Protection (MDATP) by correlating with additional data sources, such as W3CIIS log. We then show how Azure Sentinel's Security Orchestration Automation and Response (SOAR) capabilities can be used to automatically develop and share new insights from these logs back with MDATP, triggered by one of its own alerts.

A 'web shell' is a persistence mechanism and backdoor often leveraged by attackers on web servers. Usually this involves an attacker writing a script to a web application directory and using it to execute commands from a remote location. In the last 18 months the profile of web shell based attacks has increased, due in part to a number of vulnerabilities reported in web applications such as [CVE-2019-0604](#) or [CVE-2019-16759](#). Multiple threat actors have also been seen to leverage web shells in recent campaigns, Microsoft has reported on [GALLIUM](#) and [other groups](#) earlier in the year. Web shells are an attractive remote access method for attackers since they often sit on internet facing infrastructure and their short content length can make writing anti-virus signatures for them difficult.

The scenario below has been created to form the basis of our investigation and enrichment. An attacker will first exploit CVE-2019-0604, a patched SharePoint vulnerability, to place a web shell file on the server. The attacker will then leverage the shell to copy itself into another directory to provide a redundant backdoor.

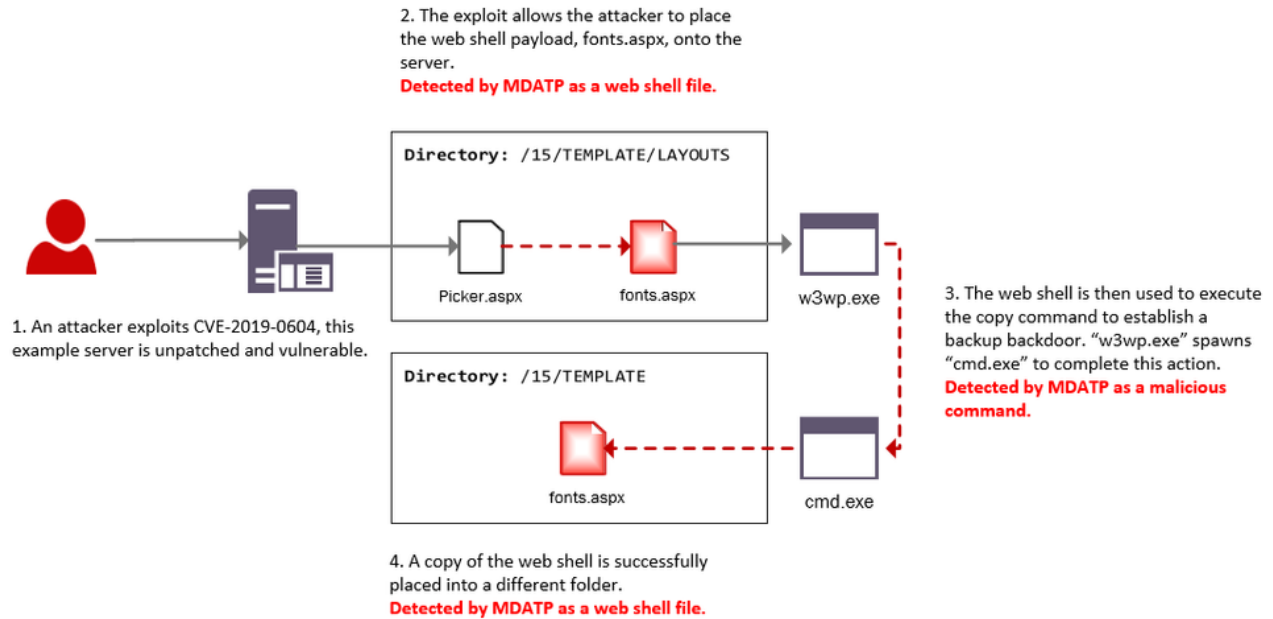



Diagram showing where MDATP alerts trigger during our investigation.

We will focus our investigation on two areas:

Web shell installation: A web shell file is placed on the server and the code it contains or the behaviours it exhibits result in an MDATP alert. In this scenario the alert will contain details of the potential shell (e.g. c:\mywebapp\webshell.aspx).

Web shell activity: The web server executes a series of suspicious commands that look like they might be web shell activity, and result in an MDATP alert. In this scenario the alert will contain details of the suspicious processes executed.

 **Possible web shell installation**
 This alert is part of incident (3)

Actions ▾

Severity: Medium
 Category: Persistence
 Technique: T1100: Web Shell
 Detection source: EDR
 Detection technology: Behavioral
 Detection status: Detected

Example EDR alert for

Description
 A suspicious web script was written in a folder containing a web application. An attacker might be attempting to install a web shell and establish persistent access.

web shell installation in the MDATP Security Center.

Connecting the data sources

MDATP alerts can be connected to Azure Sentinel using the built in connector. Once enabled, alerts are written to the SecurityAlert table.

<input type="checkbox"/>	TimeGenerated [UTC]	DisplayName	AlertName
<input checked="" type="checkbox"/>	5/5/2020, 5:36:55.000 PM	Possible web shell installation	Possible web shell installation
...			
	TimeGenerated [UTC]	2020-05-05T17:36:55Z	
	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	
>	Entities	[{"Sid": "4", "DnsDomain": "msticsptest.net", "HostName": "sps-app-0", "IsD	

Example of an MDATP

alert in Azure Sentinel.

The evidence associated with an alert in this table can be found in the 'Entities' field. This is where we'll need to look to extract the details of suspicious files and processes.

Additional visibility into web servers can be enabled by configuring collection of web server logs in Azure Sentinel. Although in this example we discuss collection from IIS web servers, data could also be collected from other server technologies such as Apache or Tomcat to enable similar investigations. For more detail on configuring IIS log collection for Azure Sentinel see here.

In Azure Sentinel, the W3CIIS table contains details of requests made to the web server, including the source IP, the request page and the user agent used.

<input type="checkbox"/>	TimeGenerated [UTC]	sSiteName	sIP	cs...	csUriStem	csUriQuery	*	clP	csUserAgent
<input type="checkbox"/>	4/6/2020, 10:49:13.000 AM	SharePoint Central Administration v4	10.0.3.5	GET	/_layouts/15/cui.js	rev=hkelvuz%...	80	10.0.2.4	Mozilla/4.0+(compatible;+MSIE+7.0...
<input type="checkbox"/>	4/6/2020, 10:49:13.000 AM	SharePoint Central Administration v4	10.0.3.5	GET	/_layouts/15/ima...	rev=23	80	10.0.2.4	Mozilla/4.0+(compatible;+MSIE+7.0...
<input type="checkbox"/>	4/6/2020, 10:49:13.000 AM	SharePoint Central Administration v4	10.0.3.5	GET	/_layouts/15/ima...	rev=23	80	10.0.2.4	Mozilla/4.0+(compatible;+MSIE+7.0...
<input type="checkbox"/>	4/6/2020, 10:49:13.000 AM	SharePoint Central Administration v4	10.0.3.5	GET	/_layouts/15/ima...	rev=23	80	10.0.2.4	Mozilla/4.0+(compatible;+MSIE+7.0...
<input type="checkbox"/>	4/6/2020, 10:49:13.000 AM	SharePoint Central Administration v4	10.0.3.5	GET	/_layouts/15/blan...	rev=ZaOXZlEo...	80	10.0.2.4	Mozilla/4.0+(compatible;+MSIE+7.0...

Azure

Sentinel's W3CIIS log table.

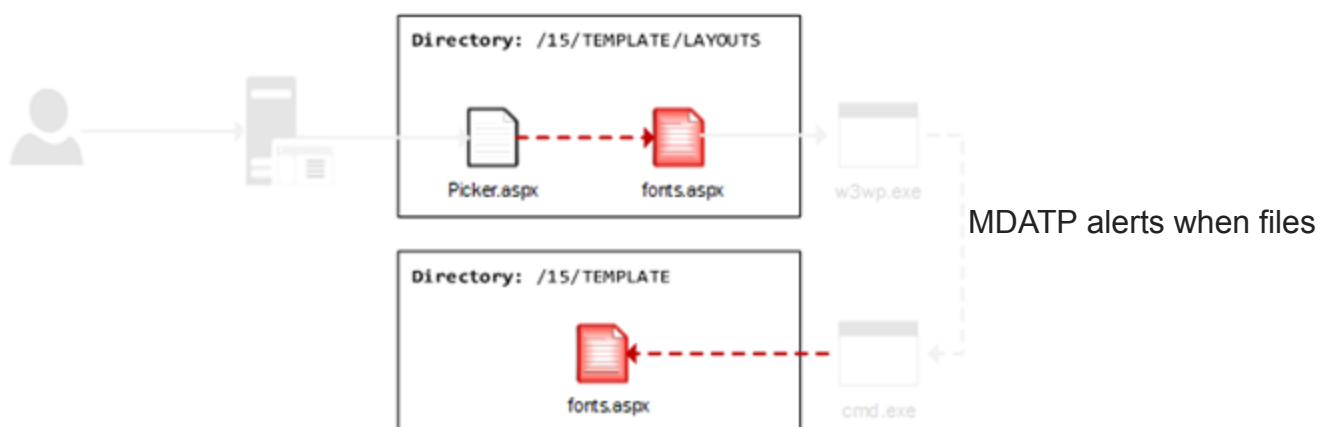
Investigating the alerts and generating new insights

By connecting MDATP alerts and relevant data sources to Azure Sentinel, we can gain additional insights and enrich our investigation. In the case of a web shell, we want to be able to answer questions such as how it got there and where the attack came from.

Acting against the web shell alone may provide a highly motivated attacker with a window of opportunity to start diversifying their access, but by acting against the web shell with MDATP in parallel to identifying the attacker's IP and user agent in Azure Sentinel a more developed picture of the attacks can be formed.

Web shell installation

In the attack scenario there are two instances where a web shell file is placed onto the server; when the attacker uses Picker.aspx to deploy the initial shell and when the attacker copies the file into a new directory. Both actions trigger MDATP alerts for the web shell file which contain a filename for the shell that we can further investigate. The diagram below highlights where in the example scenario this activity takes place.



are written to the system.

The first step is to extract web script filenames from alerts in the Azure Sentinel SecurityAlerts table. The information is stored in a JSON entity, we need to parse that to extract its name and location. The query below will perform the extraction and display the

time the event was generated, the filename which is the name of the shell and the directory that the shell is in.

Web shell file query, Part 1/2

```
let timeWindow = 3d;
//Script file extensions to match on, can be expanded for your environment
let scriptExtensions = dynamic([".php", ".jsp", ".js", ".aspx", ".asmx", ".asax",
".cfm", ".shtml"]);
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
```

The alert has been parsed to provide the file name that triggered the alert, 'fonts.aspx', and the location of the file. The web shell has been installed on a SharePoint server and written to the publicly accessible 'Template/Layouts' directory.

TimeGenerated [UTC]	FileName	Directory
5/5/2020, 5:36:56.000 PM	fonts.aspx	C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\TEMPLATE\LAYOUTS
5/5/2020, 5:36:55.000 PM	fonts.aspx	C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\TEMPLATE\LAYOUTS

Parsed alert showing web shell location.

We can now expand our Kusto query to join the extracted file with the W3CIISLog table and find detail about web requests that were made to it. This is easy to do as we have the filename and the time the alert was generated.

Web shell file query, Part 2/2

```
| 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
```

Because only the attacker knows the location of the web shell, we can be confident that requests for it in W3CIIS log are malicious. The results of the query above will provide indicators for the potential attacker - The attacker IP address and user agent.

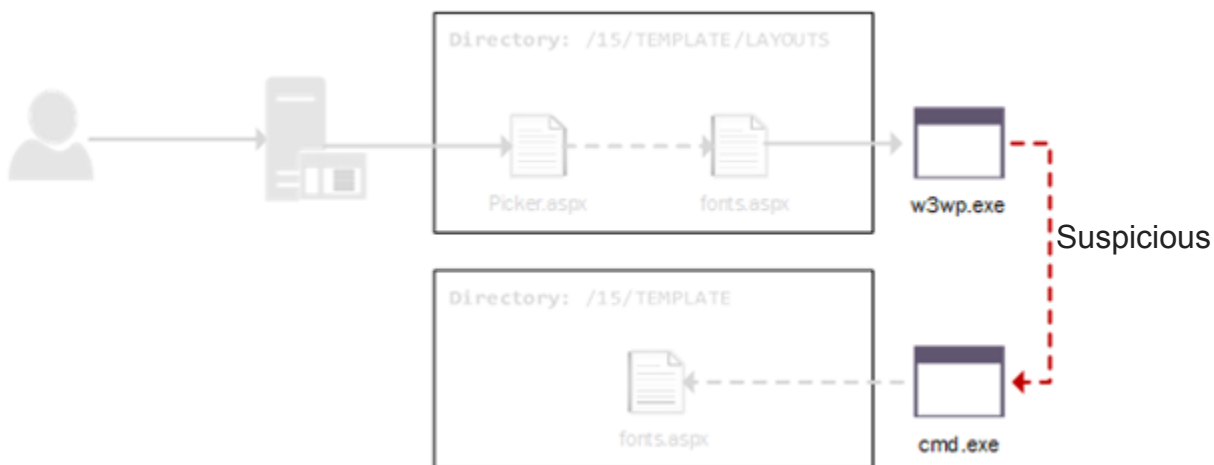
StartTime [UTC]	EndTime [UTC]	AttackerIP	AttackerUserAgent	SiteName	ShellLocation
5/5/2020, 5:33:28.000 PM	5/5/2020, 5:34:47.000 PM	51...	Mozilla/4.0+(compatible;+MSIE+7...	SharePoint - 33174	/_layouts/15/fonts.aspx

Potential attacker IP and user agent identified.

Using these indicators we can investigate other actions the attacker has taken on the webserver. Some of these follow-on queries have been automated in a Notebook covered later in the post, but first we will explore what can be achieved if the alert doesn't provide the web shell filename.

Web shell activity

In the example scenario the attacker uses the web shell to execute a command that copies itself to a backup location. In this case we can investigate with Azure Sentinel based upon the suspicious command execution alerts from MDATP. The diagram below highlights where in the example scenario this activity takes place.



command execution stage.

Multiple MDATP alerts may be received when suspicious commands are executed by an attacker through their web shell. In order to capture all alerts for web shell activity we will look for those that implicate the IIS worker process – w3wp.exe, by searching the evidence in the 'Entities' field. For the purposes of our investigation, we can be confident that alerts involving w3wp.exe are related to web shells.

Command execution query, Part 1/4

```
let timeRange = 3d;
let alerts = SecurityAlert
| where TimeGenerated > ago(timeRange)
| extend alertData = parse_json(Entities), recordGuid = new_guid();
let shellAlerts = alerts
| where ProviderName == "MDATP"
| mvexpand alertData
| where alertData.Type == "file" and alertData.Name == "w3wp.exe"
| distinct SystemAlertId
| join kind=inner (alerts) on SystemAlertId;
```

Due to the behavioural nature of the detection, the alerts do not provide the web shell file name, we need to consider another method of correlating them with the W3CIISLog table. Here we will use the time of command execution in the SecurityAlert table to collect linked web request events from W3CIISLog table, this will give us some candidate files to investigate.

Before linking events we extend the query to extract the suspicious command, the time it was executed and the host it was executed on.

Command execution query, Part 2/4

```
//Script file extensions to match on, can be expanded for your environment
let scriptExtensions = dynamic([".php", ".jsp", ".js", ".aspx", ".asmx", ".asax",
".cfm", ".shtml"]);
let alldata = shellAlerts
| mvexpand alertData
| extend Type = alertData.Type;
//Collect alerts that match our shell criteria
let filedata = alldata
| extend id = tostring(alertData.$id)
| extend ImageName = alertData.Name
| where Type == "file" and ImageName != "w3wp.exe"
| extend imagefileref = id;
//Collect commandline and execution data from the alert
let commanddata = alldata
| extend CommandLine = tostring(alertData.CommandLine)
| extend creationtime = tostring(alertData.CreationTimeUtc)
| where Type =~ "process"
| where isnotempty(CommandLine)
| extend imagefileref = tostring(alertData.ImageFile.$ref);
//Collect information about the host where the command was executed
let hostdata = alldata
| where Type =~ "host"
| project HostName = tostring(alertData.HostName), DnsDomain =
tostring(alertData.DnsDomain), SystemAlertId
| distinct HostName, DnsDomain, SystemAlertId;
//Now we have extracted the file and command line information from the MDATP JSON
object, re-combine them
filedata
| join kind=inner (
commanddata
) on imagefileref
| join kind=inner (hostdata) on SystemAlertId
| project DisplayName, recordGuid, TimeGenerated, ImageName, CommandLine, HostName,
DnsDomain
```

When attackers gain initial access, they often propagate their web shells into other directories, for redundancy and/or to bypass restrictions the web server implements on code executing from the IIS web root. If they do this via a shell command, the name of the shell will appear on the command line. We can extract this value and place it in a column “PossibleShell”.

Command execution query, Part 3/4

```
| extend PossibleShell = iff(CommandLine has_any(scriptExtensions), extract('@'([a-zA-Z0-9\-\_\.]+\.[php|jsp|js|aspx|asmx|asax|cfm|shtml]+)(?:\s|"$)')', 1, CommandLine) ,
"n/a" )
```

In the output below, we can see the extracted commands and time created.

TimeGenerated [UTC]	ImageName	CommandLine	PossibleShell	HostName	DnsDomain
5/5/2020, 5:34:35.000 PM	cmd.exe	"cmd.exe" /c copy "C:\Program Files\Common Fil...	fonts.aspx	sps-app-0	msticsptest.net
5/5/2020, 5:34:36.000 PM	cmd.exe	"cmd.exe" /c copy "C:\Program Files\Common Fil...	fonts.aspx	sps-app-0	msticsptest.net

We can now expand the query to find the IP address and user agent of our attacker based upon the time of the suspicious command execution. By generating a time key we can join the SecurityAlert and W3CIISLog tables. This join will allow us to extract access to script files on the server during a 1 minute window of the command executing – this will provide us with possible web shell files and which client IP’s and user agents were accessing them at the time. On busy web servers large numbers of scripts will be accessed throughout the day increasing the chances of a false positive. To help mitigate this the query will only present scripts that have had less than 3 distinct client IP addresses visit them.

Command execution query, Part 4/4

```

let lookupWindow = 1m;
let lookupBin = lookupWindow / 2.0;
let distinctIpThreshold = 3;
let commandKeyedData = filedata
//Time key the command line ready for join with web log
| join kind=inner (
commanddata
) on imagefileref
| join kind=inner (hostdata) on SystemAlertId
| project recordGuid, TimeGenerated, ImageName, CommandLine, TimeKey =
bin(TimeGenerated, lookupBin), HostName, DnsDomain
| extend Start = TimeGenerated;
//Baseline page access by unique IP to later limit results
let baseline = W3CIISLog
| where TimeGenerated > ago(timeRange)
| project-rename SourceIP=cIP, PageAccessed=csUriStem
| summarize dcount(SourceIP) by PageAccessed
| where dcount_SourceIP <= distinctIpThreshold;
// Time key join malicious command with web log to find correlated events
commandKeyedData
| join kind=inner (
W3CIISLog
| where TimeGenerated > ago(timeRange)
| where csUriStem has_any(scriptExtensions)
| extend splitUriStem = split(csUriStem, "/")
| extend FileName = splitUriStem[-1] | extend firstDir = splitUriStem[-2] | extend
TimeKey = range(bin(TimeGenerated-lookupWindow, lookupBin), bin(TimeGenerated,
lookupBin),lookupBin)
| mv-expand TimeKey to typeof(datetime)
| summarize StartTime=min(TimeGenerated), EndTime=max(TimeGenerated) by
Site=sSiteName, HostName=sComputerName, AttackerIP=cIP,
AttackerUserAgent=csUserAgent, csUriStem, filename=toString(FileName),
toString(firstDir), TimeKey
) on TimeKey, HostName
| where (StartTime - EndTime) between (0min .. lookupWindow)
| extend attackerP = pack(AttackerIP, AttackerUserAgent)
| summarize Site=makeset(Site), Attacker=make_bag(attackerP) by csUriStem, filename,
toString(ImageName), CommandLine, HostName
| project Site, ShellLocation=csUriStem, ShellName=filename, ParentProcess=ImageName,
CommandLine, Attacker, HostName
| join kind=inner (baseline) on $left.ShellLocation == $right.PageAccessed

```

As we can see in the image below the query was able to successfully link the commands that were executed through a potential web shell on the server, this allows us to get the attackers IP address and their user agent for follow on queries.

Site	ShellLocation	CommandLine	Attacker	HostName	dcount_SourceIP
[\"SharePoint - 33174\"]	/_layouts/15/fonts.aspx	cmd.exe /c copy C:\Program Files\Co...	[\"51:00000000000000000000000000000000;Mozilla/4.0+(compatible;+MSIE+7.0;+...	sps-app-0	1
[\"SharePoint - 33174\"]	/_layouts/15/fonts.aspx	cmd.exe /c copy C:\Program Files\Co...	[\"51:00000000000000000000000000000000;Mozilla/4.0+(compatible;+MSIE+7.0;+...	sps-app-0	1

Attacker IP and user agent for follow-on queries.

We will take these new insights on the attack and use them to identify other activity on the compromised web server. We could do this through additional queries, but to help speed the process up we have created a notebook that automatically enriches and reports interesting attacker activity.

Building out the investigation using Jupyter Notebooks

Jupyter notebooks are a great way to pull these investigation steps together in one place for analysis. If you are new to Jupyter notebooks and would like to understand how it can help with Azure Sentinel investigations [Ian Hellen](#) wrote a [series of blogs](#) covering the topic.

In the example below we built a notebook wrapping each of these steps into a single analysis flow and have added some additional analytics to help understand what the attacker may have been doing on your server.

The notebook uses the Kusto client built into [MSTICPy](#) to enable it to communicate directly with your LogAnalytics server, we also use other MSTICPy dependencies to manipulate and investigate the data we recover. If you do not have MSTICPy configured in your environment already then go ahead and “pip install msticpy”. More detailed instructions about installation and setup can be found in the [documentation](#).

Upon loading into the notebook you will see a number of instructions that cover its use, begin by importing dependencies. Then configure, connect and authenticate with your LogAnalytics instance.

Once configured the notebook provides two different investigations to choose from, these match the two scenarios we discussed earlier in the post. You can investigate alerts that are based on a web shell file being detected or you can begin an investigation into alerts where suspicious command execution from a likely web shell have been detected. Each investigation will automate the previous queries and prepare the data for further enrichment in the notebook.

To investigate a suspicious file alert, you will start with the cell “Begin File Investigation” to be presented with the option to choose which potential web shell file you would like to investigate. The cell will automatically retrieve MDATP alert entities from your server, automating the manual hunting queries we executed earlier.

You can see in the image below that a web shell called “fonts.aspx” was dropped to the server, matching the output of our manual hunting queries.

Please select a webshell to investigate before you continue:

	TimeGenerated	filename	directory
0	2020-05-05 17:36:55	fonts.aspx	C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\TEMPLATE\LAYOUTS

fonts.aspx

To investigate a suspicious command execution alert, you must first select the command you wish to investigate. Running the “Begin Command Investigation” cell will produce a list of suspicious commands detected by MDATP for you to choose from, again this information will be pulled directly from your server. You will need to select a GUID associated with the command you wish to investigate. As we discovered earlier in our investigation while running manual hunting queries, you can see in the image below that commands have been executed to copy a web shell file into different locations.

	recordGuid	TimeGenerated	creationtime	imagename	commandline	DisplayName
0	672950de-0731-43c1-a3fb-230c08bb0864	2020-05-05 17:34:36	2020-05-05T17:29:14.6333745Z	cmd.exe	"cmd.exe" /c copy C:\Program Files\Common Files\microsoft shared\Web Serber Extensions\15\TEMPLATES	A suspicious web script was created
1	52601eb7-1e66-4b6e-a493-c019387852a4	2020-05-05 17:34:35	2020-05-05T17:29:14.6333745Z	cmd.exe	"cmd.exe" /c copy C:\Program Files\Common Files\microsoft shared\Web Serber Extensions\15\TEMPLATES	A suspicious web script was created

While this is similar to the Kusto query we wrote earlier, this time the script will perform a check to see if the file has only been accessed by a single client IP, this is a further step to help narrow down candidate web shell files. The web shell files will often only be accessed by an attacker and no other users on the server, therefore the number of IP addresses accessing them will often be a single IP. If the attacker is rotating their infrastructure regularly you can change the threshold at the top of the cell by changing the variable named “access_threshold”. The higher you increase this number the higher the risk of false positives, but on a production server this threshold can likely be raised quite high.

Regardless of the type of alert you chose to investigate (file or command) we now have candidate files to investigate, we can enrich them with access information to provide us with a potential attacker IP and user agent. In the below image you can see that we have processed the candidate files and found potential attacker IP’s and user agents. If the “access_threshold” has been increased you may see files that have been accessed by legitimate users at a similar time as the command was executed, especially if the server is busy or the attack happens during core working hours.

Candidate Attacker IP Addresses

The following attacker IP addresses accessed the webshell during the alert window.

AttackerIP	AttackerUserAgent	SiteName
51 [redacted]	Mozilla/4.0+ (compatible;+MSIE+7.0;+Windows+NT+6.3;+WOW64;+Trident/7.0;+.NET4.0E;+.NET4.0C;+.NET+...	SharePoint - 33174 /_layouts/

Select Investigation Parameters

Attacker To Investigate

Select parameters to investigate, the default selection is the earliest access within the alert window:

51 [redacted] Mozilla/4.0+(compatible;+MSIE+7.0;+Windows Investigate Both

Previous file access window

To determine what files were accessed immediately before the shell, please pick the window we'll use to look back:

30m

Here you can choose the attacker IP or user agent you want to investigate, or you can choose to investigate both. You will also need to specify a look-back file access time window as the notebook will try to determine which files the attacker accessed immediately before the web shell. By default it will look-back over 30 minutes of data before the suspected web shell execution. Once we have chosen these parameters it is time to continue executing the notebook to generate a report.

We have successfully taken an MDATP alert and enriched it with information that has allowed us to find both the web shell location and the attacker's information.

When dealing with a rapidly evolving incident, security teams can now act not only against the web shell, but the IP and user agent selectors generated that can be used to block or further investigate the attacker.

We are in a position where we can choose to remove the attacker's access in parallel to the web shell removal. This effectively switches the focus of our investigation to the attacker and not the tools they have used. When countering hands on keyboard attacks from both advanced persistent threats and sophisticated cybercrime groups, understanding the attacker and their behavior is critical to successful eviction from the network. Removing the attacker's ability to communicate with their tools also removes any early warning the attacker may see and will hamper their ability to persist. The notebook contains a file history analytic that will allow the identification of files that may have been used to compromise the server so the entry point can also be investigated.

The notebook provides two built-in analytics to help get your investigation started. As well as reporting the attacker IP, user agent, the location of the shell and the website impacted, the notebook contains two analytics that it will generate a report on the following:

- **File History:** This is a list of the files on the server accessed immediately before the web shell file was placed by the attacker. The option “previous file access window” can be used to configure how far the script looks back through the logs. These files are good leads to continue your investigation, in a scenario where a malicious web shell file was detected on the server you may be able to use this analytic to identify the page that was exploited immediately before the web shell is deployed by the actor.
- **Earliest Server Access:** This analytic looks back across your server logs and determines the earliest point that the attacker IP and/or user agent accessed your server, this is limited by a time range and is configurable in the initial config & connect section. By default, the notebook will look-back 3 days, but this can be easily expanded.

Below is an example of the summary report the notebook generated about the attacker. We can see that prior to their first web shell access they had made a request to “Picker.aspx”. This file is strongly associated with the patched SharePoint vulnerability [CVE-2019-0604](#).

Attack Summary

Attacker IP: 51. [REDACTED]

Attacker user agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.3; WOW64; Trident/7.0; .NET4.0E; .NET4.0C; .NET CLR 3.5.30729; .NET CLR 2.0.50727; .NET CLR 3.0.30729)

Webshell installed: /_layouts/15/fonts.aspx

Victim site: SharePoint - 33174

File history

The files the attacker IP "51. [REDACTED]" accessed prior to the webshell installation were:

	TimeAccessed	SiteName	ServerIP	FilesTouched	AttackerP
0	2020-05-05 17:29:18+00:00	SharePoint - 33174	10.0.3.5	/sites/MyTeamsSiteCollection/_layouts/15/home.aspx	51. [REDACTED]
1	2020-05-05 17:29:18+00:00	SharePoint - 33174	10.0.3.5	/sites/MyTeamsSiteCollection/_layouts/15/home.aspx	51. [REDACTED]
2	2020-05-05 17:29:20+00:00	SharePoint - 33174	10.0.3.5	/_layouts/15/Picker.aspx	51. [REDACTED]
3	2020-05-05 17:29:35+00:00	SharePoint - 33174	10.0.3.5	/_layouts/15/fonts.aspx	51. [REDACTED]

Earliest access

In the last 30 days the earliest known access to the server from the attacker IP was:

	TimeAccessed	Site	FileAccessed
0	2020-05-05 17:29:18+00:00	SharePoint - 33174	/sites/MyTeamsSiteCollection/_layouts/15/home.aspx

In this scenario we have taken multiple alerts from MDATP and provided enrichments from W3CIIS Logging to identify the attackers IP and user agent, these can now be used to further investigate the attackers actions on the server. We have been able to automate components of that investigation in a notebook and have provided a workflow that analysts and responders can use. The notebook also provides a foundation that can be built upon, with

some knowledge of Kusto and Python it's possible for security teams to tailor the notebook to their environment. We'll explore such an expansion next and look at how we can communicate our findings back to MDATP.

Providing insights back to MDATP

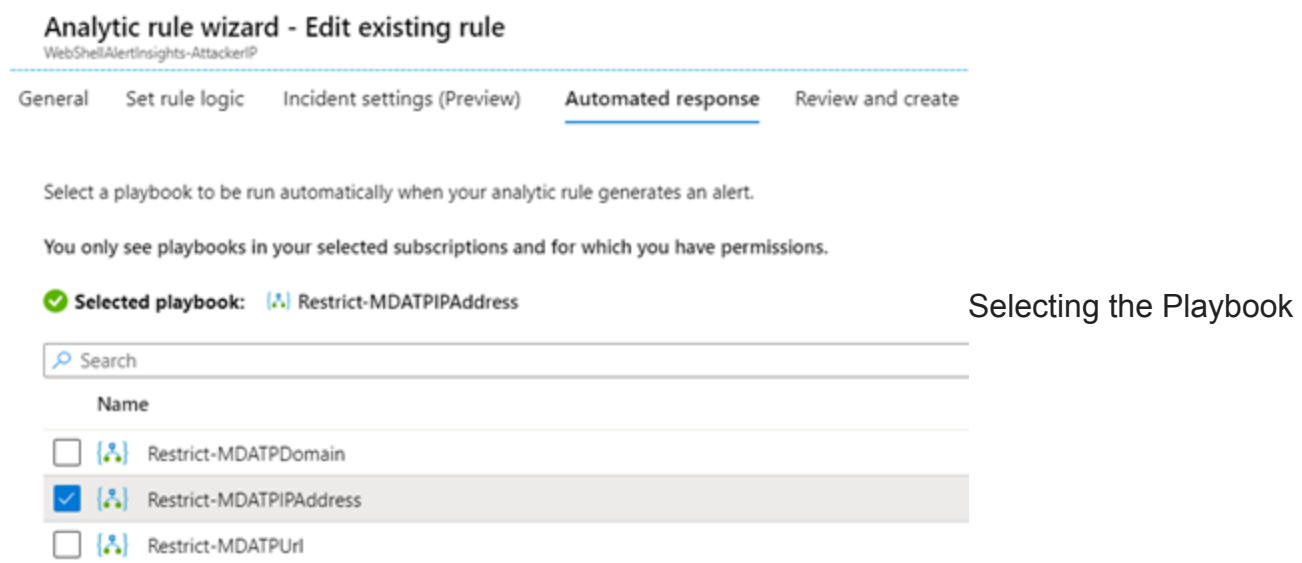
By writing an analytic query in Azure Sentinel we can automate the generation of some of these insights. When written as an analytic, the query below can be scheduled to run periodically. It extracts web shell files from MDATP alerts over the last hour, correlates them with historic data from the W3CIIS log and creates a new incident, enriched with the details of attacks leveraging the shell including the IP address of the attacker.

```

let alertTimeWindow = 1h;
let logTimeWindow = 7d;
// Define script extensions that suit your web application environment - a sample are
provided below
let scriptExtensions = dynamic([".php", ".jsp", ".js", ".aspx", ".asmx", ".asax",
".cfm", ".shtml"]);
let alertData = SecurityAlert
| where TimeGenerated > ago(alertTimeWindow)
| where ProviderName == "MDATP"
// Parse and expand the alert JSON
| extend alertData = parse_json(Entities)
| mvexpand alertData;
let fileData = alertData
// Extract web script files from MDATP alerts - our malicious web scripts - candidate
web shells
| where alertData.Type =~ "file"
| where alertData.Name has_any(scriptExtensions)
| extend FileName = tostring(alertData.Name), Directory =
tostring(alertData.Directory);
let hostData = alertData
// Extract server details from alerts and map to alert id
| where alertData.Type =~ "host"
| project HostName = tostring(alertData.HostName), DnsDomain =
tostring(alertData.DnsDomain), SystemAlertId
| distinct HostName, DnsDomain, SystemAlertId;
// Join the files on their impacted servers
let webshellData = fileData
| join kind=inner (hostData) on SystemAlertId
| project TimeGenerated, FileName, Directory, HostName, DnsDomain;
webshellData
| join (
// Find requests that were made to this file on the impacted server in the W3CIISLog
table
W3CIISLog
| where TimeGenerated > ago(logTimeWindow)
// Restrict to accesses to script extensions
| where csUriStem has_any(scriptExtensions)
| extend splitUriStem = split(csUriStem, "/")
| extend FileName = splitUriStem[-1], HostName = sComputerName
// Summarize potential attacker activity
| summarize count(), StartTime=min(TimeGenerated), EndTime=max(TimeGenerated),
RequestUserAgents=make_set(csUserAgent), RequestMethods=make_set(csMethod),
RequestStatusCodes=make_set(scStatus), RequestCookies=make_set(csCookie),
RequestReferers=make_set(csReferer), RequestQueryStrings=make_set(csUriQuery) by
AttackerIP=cIP, SiteName=sSiteName, ShellLocation=csUriStem, tostring(FileName),
HostName
) on FileName, HostName
| project StartTime, EndTime, AttackerIP, RequestUserAgents, SiteName, ShellLocation,
RequestMethods, RequestStatusCodes, RequestCookies, RequestReferers,
RequestQueryStrings, RequestCount = count_
// Expose the attacker ip address as a custom entity
| extend timestamp=StartTime, IPCustomEntity = AttackerIP

```


When creating the analytic, as part of Azure Sentinel's Security Orchestration Automation and Response (SOAR) capabilities we can also specify an automated response - A playbook to run when incidents are generated. In the below example we have selected the Playbook 'Restrict-MDATPIPAddress'




Analytic rule wizard - Edit existing rule
WebShellAlertInsights-AttackerIP




General Set rule logic Incident settings (Preview) **Automated response** Review and create

Select a playbook to be run automatically when your analytic rule generates an alert.

You only see playbooks in your selected subscriptions and for which you have permissions.

Selected playbook:  Restrict-MDATPIPAddress

Search

Name	
<input type="checkbox"/>	 Restrict-MDATPDomain
<input checked="" type="checkbox"/>	 Restrict-MDATPIPAddress
<input type="checkbox"/>	 Restrict-MDATPUrl

Selecting the Playbook

'Restrict-MDATPIPAddress'.

This playbook extracts IP Address information from the enriched incidents created by our analytic, calls the MDATP indicator API and submits them as custom IoCs for 'AlertAndBlock'. Once submitted, MDATP will create new alerts where this IP address is observed in the environment and block endpoints from connecting to it for 90 days.

A similar analytic and automated response could also be configured for the scenario where we link suspicious commands to web shells based on a time key. Playbooks can also be triggered manually, if preferred through the incident investigation experience.

Enriching the Incident

Microsoft Threat Protection (MTP) orchestrates across endpoint, identity, email and applications to stop attack sprawl and auto-heal enterprise assets. Its built-in intelligence and automation combine signals across Microsoft ATP products into prioritized incidents enabling security professionals to determine the full scope and impact of the threat.

The installation and usage of a web shell is often only a small part of the overall attack. Web shells are used as beachheads to perform follow on actions such as credential theft, lateral movement and data exfiltration. Each of these actions may generate signal in other Microsoft products such as Azure ATP or Microsoft Cloud App Security and MTP will create an incident that covers them all. As well as web shell artifacts from MDATP this might include compromised identities, or details of cloud applications such as OneDrive that were used to exfiltrate sensitive data from the network.

By enriching individual MDATP alerts with Azure Sentinel log sources we have shown that we can build out the picture of the intrusion with details of the attacker and their infrastructure. Extending this enrichment to an MTP incident allows us to take this further and perform compound enrichments based upon the full scope of the attack.

Working with an incident involving a web shell we can enrich endpoint alerts with the IP address and user agent from web server logs as before. With the incident we might also get compromised identities used in the attack from Azure ATP. We can combine these identities with the IP address and user agent insights to fingerprint attacker activity inside other Azure Sentinel logs sources – such as the SignIn logs or OfficeActivity.

In a follow up post we will describe how upcoming integration between MTP and Azure Sentinel will allow us to enrich based upon the incident rather than individual alerts and explore new scenarios that demonstrate the power of this combination.

Summary

By connecting MDATP alerts and relevant log sources to Azure Sentinel we combined signals to further investigate web shell installation and activity alerts and gained additional high value insights – the attacker IP address and user agent. Using Jupyter notebooks we performed an investigation based upon these insights to build out the big picture of the intrusion and identify the vulnerability that allowed the web shell to be installed. Using Sentinel's SOAR capabilities, we went further and automated the generation of these insights, sharing them back with MDATP through the indicator API for further action – such as new alerting and blocking. A number of resources relevant to the blog were added to the Azure Sentinel github and can be found below:

Kusto Query: [Web shell file enrichment query](#).

Kusto Query: [Web shell command enrichment query](#).

Kusto Detection: [Malicious alert linked we request](#)

Jupyter Notebook: [Web shell investigations using MDATP](#)