PEAK Threat Hunting Workshop - Exercise Guide

This exercise guide is designed to be used in conjunction with the *PEAK Threat Hunting—Hands-On* workshop. During the workshop, a series of exercises are interspersed with the presentation. Participants can refer to this document to assist during the workshop and as a reference afterward.

The final section of this document also includes links to more information about PEAK, including the framework document itself, presentation videos, and other resources.

NOTE: Be sure to set the time range to "All Time" for each exercise in this workshop!

Exercise #1: Hypothesis-Driven Hunting for Encoded PowerShell Commands

For this exercise, our hunting hypothesis is as follows:

Adversaries are using PowerShell with Base64 encoded obfuscations to hide their commands and avoid detection.

Command-line obfuscation makes strings and patterns within commands and scripts more difficult to identify and analyze. Adversaries often use PowerShell and some commonly abused obfuscations to hide their commands and avoid detection.

Many types of command obfuscation can be used to hide PowerShell commands, but the most common is probably encoding malicious commands using the Base64 algorithm. That's what we'll focus on in this example.

The PowerShell `-encodedcommand` parameter accepts a base64 string that can be used to hide commands. In addition, it's important to know that PowerShell accepts many different shortened variations of this command (e.g., `-ec`, `-enc`, `-en`, etc).

Action: Find PowerShell Commands Executing on Hosts

For the first exercise, we'll begin by searching for any PowerShell command just to get a sense of the different types of commands we see in our environment (and ensure that we have this data). We'll be using process creation logs generated by Microsoft Sysmon. Also, for demonstration purposes, we'll focus entirely on processes called "powershell.exe", though in

reality, we know that threat actors sometimes call PowerShell via different names (another attempt at obfuscation).

What PowerShell commands are executing in our environment?

```
index=main
sourcetype="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
EventCode=1 process_name="powershell.exe"
| stats count by host user CommandLine
| sort - count
```

Inspect the results and see if you can answer the following questions:

1. What systems have PowerShell executing on them?
2. Are there any encoded commands in the CommandLine fields of the events?
    a. If so, can you decode them?

Beware of legitimate activity! Many vendor tools and corporate environments use Base64 encoded commands for benign purposes. Just because an encoded command is present doesn't mean it's necessarily malicious.

Bonus Action: Examine variations of the -encodedcommand parameters

The previous search returned **all** instances of PowerShell executing in the environment, but our hunt is only concerned with those that contained some variation of the `-encodedcommand` parameter. We can use the built-in `rex` command to narrow down our results.

`Rex` uses regular expressions to extract data from certain fields into new fields. In this example, we can create a regular expression that finds (most of) the valid ways of invoking `-encodedcommand`, then extract those into a new field called `enc`. We can then create a quick report based on that field (including a cool sparkline that will show a timeline of when each command occurred). Any event that didn't contain the `-encodedcommand` parameter or one of its variants will not have the `enc` field added and thus will be left out of the report automatically.

Here's an updated version of the original search that will include only those events containing encoded base64 data, along with a timeline of when they occurred on each host.

What variations of -encodedcommand are being used, and on which hosts?

```
index=main
sourcetype="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
EventCode=1 process_name="powershell.exe"
```

```
| rex field=CommandLine "(?<enc>
(?i)-en?c?o?d?e?d?c?o?m?m?a?n?d?\s('|\")?)"
| stats sparkline earliest(_time) as et, latest(_time) as lt count by
host user enc
| convert ctime(et), ctime(lt)
```

Exercise #2: Create a Simple SMB Baseline

SMB is most well-known as a file transfer protocol, but Windows also uses it for other things, including:

- Printer sharing
- Network browsing
- General host-to-host remote communication

It is important to understand what “typical” SMB traffic looks like on your network, which is where a baseline comes in. The exact components of a baseline vary depending on what you're baselining, but for SMB, you might begin by answering a few basic questions such as:

- How many/which hosts provide SMB services?
- How many/which hosts consume SMB services?
- How many SMB sessions do most consumers participate in?
- How many bytes are sent/received via SMB by each host?

Action: Baseline SMB Servers and Data Transfer Volumes

In this exercise, we'll answer some of these questions and create a statistical summary of SMB traffic volume between clients and servers.

What SMB servers are present in the environment?

```
index=main sourcetype=stream:smb dest_port=445
| stats  count by dest_ip
| sort -count
```

How many SMB servers are active in the environment?

How much data does each system transfer (uploads + downloads)?

```
index=main sourcetype=stream:smb
| stats sum(bytes) as count by src_ip, dest_ip
| sort -count
```

Each line of this output represents the total traffic volume (in bytes) between an SMB client and server, regardless of direction. Are there any individual client/server pairs that seem unusual?

Describe the distribution of SMB data usage across all systems

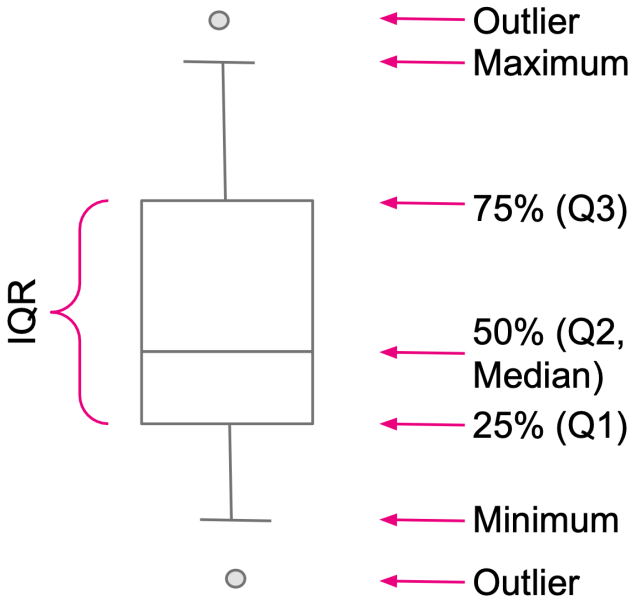
```
index=main sourcetype=stream:smb
| stats sum(bytes) as count by src_ip, dest_ip
| stats min(count) as min, max(count) as max, median(count) as median,
perc25(count) as Q1, perc75(count) as Q3, mean(count) as mean,
stdev(count) as stdev
```

The result of this search will be seven numbers that describe the distribution of the data points (traffic volume between clients and servers) in the data set. While the numbers themselves form a valuable part of the baseline, they are also directly related to finding suspicious outliers, as we'll see in the next exercise.

Exercise #3: Detecting Outliers Using a Baseline

With the statistics you computed at the end of **Exercise #2**, you can begin to look for suspicious SMB activity. Specifically, you created a description of the typical SMB data transfer volume, which you can use to find suspiciously large data transfers. We'll look at two different ways to find statistical outliers.

Interquartile Range (IQR)



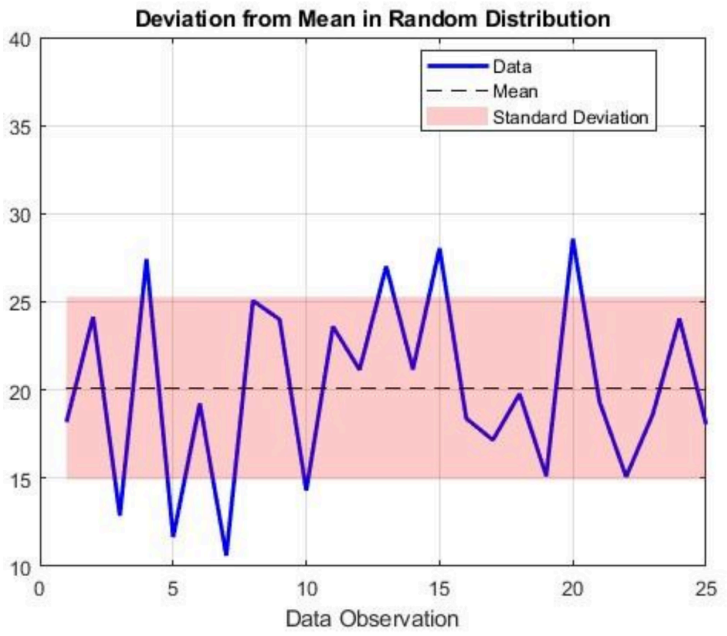
The first method uses the Interquartile Range (IQR). The first five numbers of our statistical description (which statisticians unsurprisingly call a *five-number summary*) describe the distribution of the data points over the entire set and can be visualized using a box plot such as this one.

The diagram labels the data's 25th, 50th, and 75th percentile values (Q1, Q2, and Q3 respectively). The IQR is simply the Q1 value subtracted from Q3 (`IQR = Q3 - Q1`), or the length of the "box". You can use this to compute an outlier threshold by looking for values that are more than (or sometimes less than) the median plus a multiple of the IQR value, typically `1.5 * IQR`.

For example, in our SMB data, we are probably most concerned with data transfers that are abnormally large. We can compute the IQR outlier threshold like this:


```
iqr_threshold >= median + (1.5 * (Q3 - Q1))
              >= 5194 + (1.5 * 8183)
                >= 5194 + 12274.5
                    >= 17468.5
                     >= 17469
```

Standard Deviation



Another common method of detecting outliers using statistics relies on the *standard deviation*, often notated with the Greek letter σ (the lower-case "sigma"). A dataset's standard deviation is the average distance each data point is from the overall mean. For outlier detection, a threshold of either two or three times the standard deviation is common.

The dotted line in this diagram (source: Wikimedia Commons) is the dataset's mean (average value). The shaded area represents three standard deviations above and three standard deviations below the mean. Any point within the shaded region is considered an inlier, while those outside are considered outliers.

We can compute the standard deviation outlier threshold for our SMB dataset like so:

$$\texttt{stddev\_threshold} >= \text{mean} + 3\sigma$$
$$>= 22938930.45 + (3 * 170434936.31)$$
$$>= 22938930.45 + 511304808.93$$
$$>= 534243739.38$$
$$\mathbf{>= 543243739}$$

Action: Compare Observed Traffic to the Baseline

Now that you have computed the outlier thresholds using both the IQR and standard deviation methods use them to look for anomalous data transfers.

Which systems transferred significantly more bytes? [IQR method]

```
index=main sourcetype=stream:smb
| stats sum(bytes) as count by src_ip, dest_ip
| where count >= 17469
```

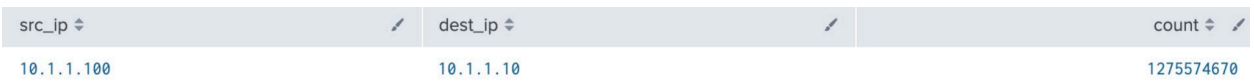
Which systems transferred significantly more bytes? [Stddev method]

```
index=main sourcetype=stream:smb
| stats sum(bytes) as count by src_ip, dest_ip
| where count >= 534243739
```

Note that each outlier method gave different results. This is normal, and choosing the "best" outlier method can be tricky. In many cases, the easiest way is to try them both and see which gives the most manageable set of results. Also, remember that each has a built-in multiplier (1.5 for the IQR method and 3 for the standard deviation method). You can feel free to adjust this multiplier as necessary. If you increase them a bit, you'll get fewer outliers, and more if you decrease them.

Bonus Action: Dig Deeper Into the Anomaly

While searching for outliers above, you may have noticed that one exceptionally large transfer appeared in both outlier lists:



This transfer is by far the most significant outlier in the data set, making it a good candidate for closer examination.

You can use the following search to identify the specific files transferred between these two hosts and their sizes in bytes:

What files were transferred between the hosts?

```
index=main sourcetype=stream:smb dest_port=445 src_ip="10.1.1.100"
| stats sum(bytes) as size by src_ip, dest_ip, filename
| sort -size
```

You can get a similar breakdown by file *type* (e.g., PDF, EXE, etc) rather than by individual filenames. This is particularly useful when there are likely to be too many filenames to review manually:

What types of files were transferred?

```
index=main sourcetype=stream:smb dest_port=445 src_ip="10.1.1.100"
| rex field=filename "(?<file_ext>(\..{3})$)"
| stats sum(bytes) as volume by file_ext
| sort -volume
```

Either way, if you investigate this anomaly, you will notice that one of the files is called *NTDS.DIT*, which is the name of the Active Directory database, and thus extremely concerning to find in SMB traffic!

PEAK Threat Hunting Resources

The PEAK Threat Hunting Framework (PDF eBook)

Bianco, D., Fetterman, R., & Morrone, S., *The PEAK Threat Hunting Framework*, Splunk, October 2023, https://splk.it/PEAK

Presentations

Bianco, David, *Achieving PEAK Performance: Introducing the PEAK Threat Hunting Framework*, SecurityOnion Conference, October 2023, https://youtu.be/GX2FKM18oxk

Bianco, David, *I Screwed Up Threat Hunting a Decade Ago and Now We're Fixing it With PEAK*, RSAC 2024, April 2024, https://youtu.be/NOUWauZ_qRs

Other Resources

Splunk operates a community Slack server with a channel dedicated to PEAK threat hunting. Visit https://splk.it/slack to sign up for a free account. After logging in, join the *#peak-threat-hunting* channel. This is a great way interact not only with the authors of PEAK but also with threat hunters and PEAK users around the world.

The SURGe team also maintains a repository of sample PEAK hunts on GitHub (https://github.com/splunk/peak). The DGA hunt from the workshop's M-ATH section as well as other hunts can be found there.